Índex d'Annexos

ÍNDEX D'ANNEXOS.....1

.ANNEX A: MANUAL DE L'APLICACIÓ TD MANAGER GUI.....2

.Descripció de l'aplicació.....2

.Connexió amb la placa TD1204.....4

.Inicialització i configuració.....5

.Transmissió de missatges per SIGFOX.....7

.Aplicacions del GPS.....8

.Aplicacions de l'acceleròmetre.....10

.Connexió amb el servidor central de SIGFOX.....11

.ANNEX B: CODI FONT DEL MÒDUL TDMANAGER.PY.....12

.ANNEX C: CODI FONT DEL MÒDUL TDMANAGERGUI.PY.....18

.ANNEX C: CODI FONT DEL MÒDUL WEBREQUESTS.PY.....42

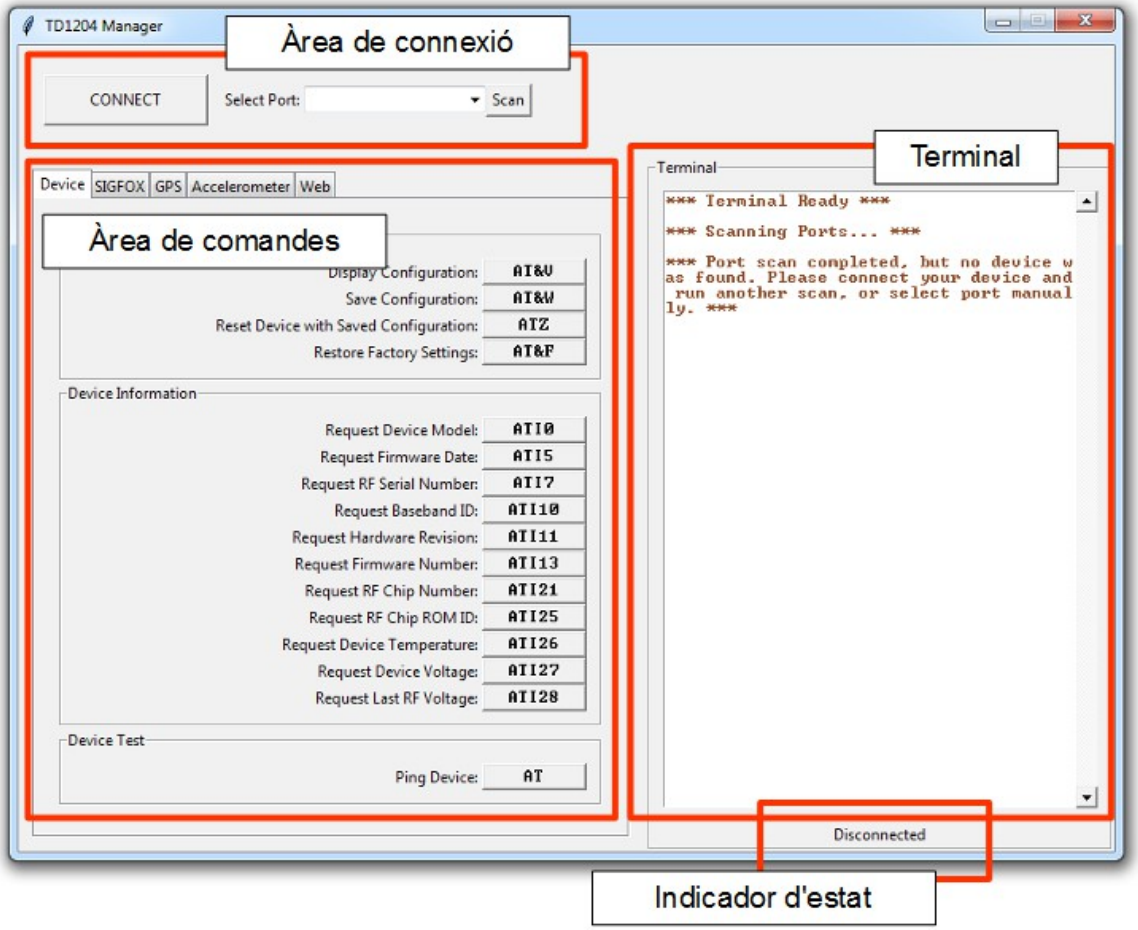
.Annex A: Manual de l'aplicació TD Manager GUI

.Descripció de l'aplicació

L'aplicació TD1204ManagerGUI és una interfície gràfica que serveix per interactuar amb una placa de desenvolupament TD1204 a través de l'enviament de comandes AT a aquesta per un port sèrie.

La finestra principal consta de diverses parts:

- Àrea de connexió: Gestiona la detecció de la placa i la connexió amb aquesta.
- Àrea de comandes: Permet enviar comandes AT a la placa; les comandes estan organitzades en pestanyes.
- Terminal: Mostra les comunicacions entre la placa i l'aplicació, així com els missatges de l'aplicació a l'usuari.
- Indicador d'estat de la placa: Mostra l'estat de la placa.



En el terminal, distingirem tres tipus de missatges, que s'indiquen amb diferent color i caràcters de marca:

- Sistema (color marró, amb la marca "***"): Missatges de l'aplicació a l'usuari.
- Enviat a la placa (color negre, amb la marca ">>>"): Comandes enviades de l'aplicació a la placa.
- Rebut de la placa (color verd, amb la marca ":"): Missatges emesos per la placa al PC.

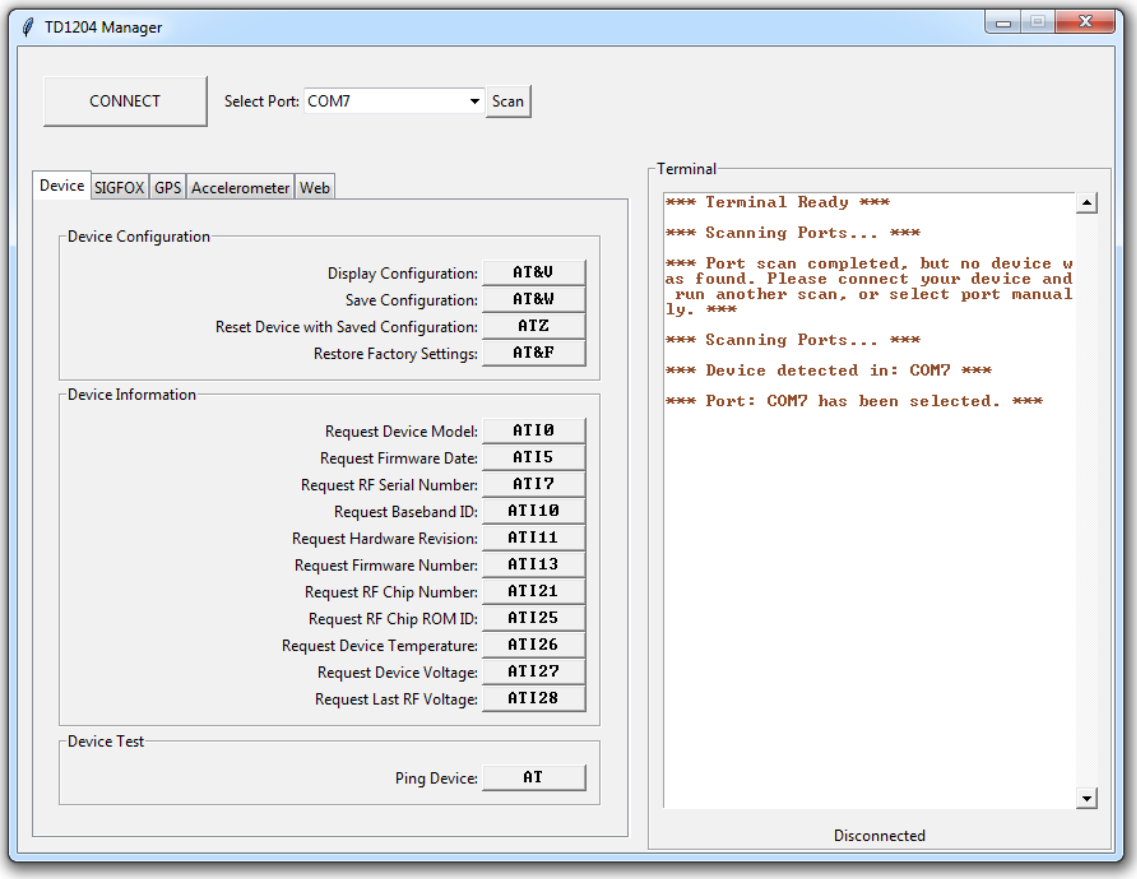
.Connexió amb la placa TD1204

Connectem la placa a un port USB del PC utilitzant el cable FTDI. Els drivers del cable FTDI simularan un port sèrie a través del port USB.

En arrencar, l'aplicació detectarà automàticament el port sèrie simulat en el qual està connectada la placa, o bé la podem detectar prement el botó "Scan" de l'àrea de connexió.

També podem seleccionar el port COM manualment en el desplegable de l'àrea de connexió.

Un cop seleccionat el port, premem el botó "CONNECT" per iniciar la connexió amb la placa.

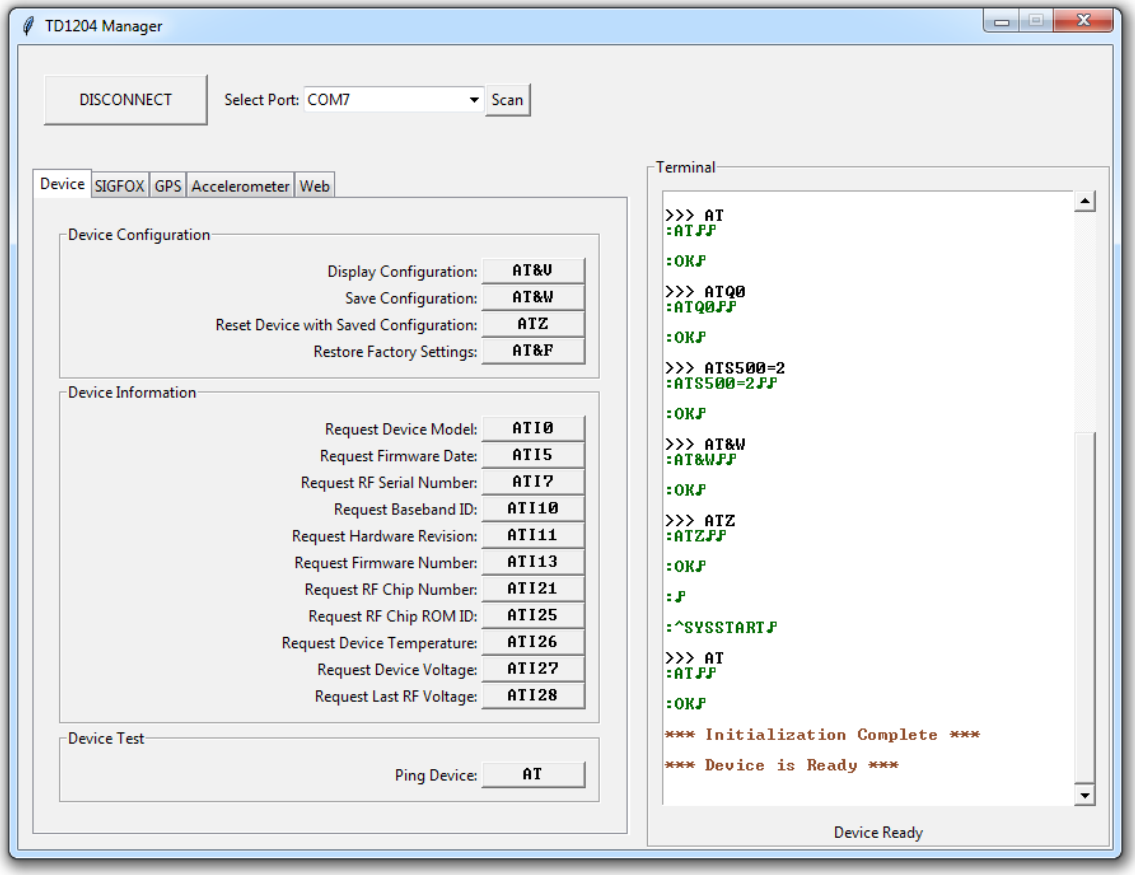


.Inicialització i configuració

Immediatament després de connectar, l'aplicació efectua una seqüència de configuració sobre la placa.

Esperem uns segons a que finalitzi aquesta seqüència, l'aplicació ens ho indicarà a través del terminal ("Initialization Complete; Device is Ready"), i l'indicador d'estat de la placa passarà de "Device Busy" a "Device Ready".

A partir d'aquest moment podem enviar comades AT a la placa.



Per configurar la placa, anem a la pestanya "Device" de l'àrea de comandes. Les comandes disponibles estan dividides en tres seccions:

- Device Configuration: Ens permet veure la configuració de la placa, desar-la, reiniciar la placa, o bé carregar la configuració de fàbrica.

- Device Information: Ens permet enviar requeriments d'informació dels diferents paràmetres de la placa, que es retornaran pel terminal.

- Device Test: Ens permet enviar un "ping" a la placa per verificar que la connexió està establerta i que la placa respon correctament. En aquest cas, la placa retornarà "OK" pel terminal.

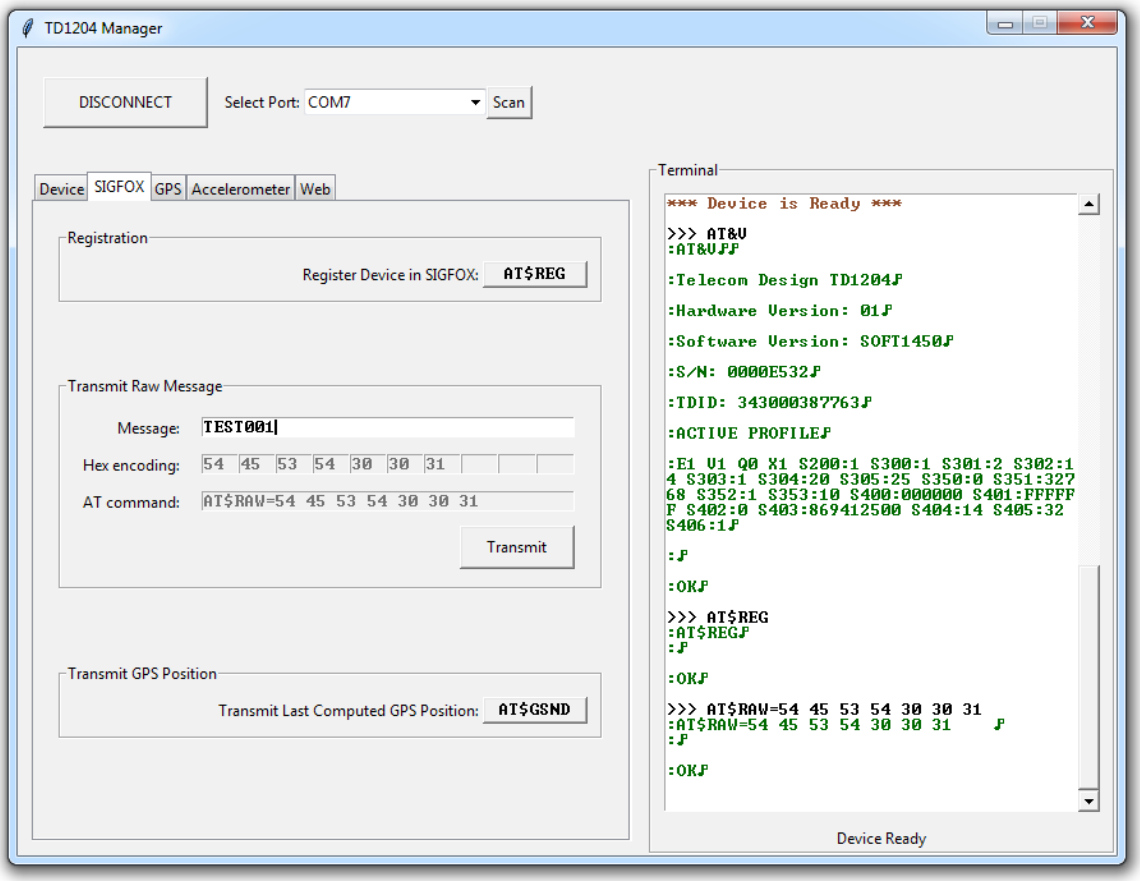
.Transmissió de missatges per SIGFOX

Per utilitzar SIGFOX anem a la pestanya "SIGFOX" de l'àrea de comandes.

La primera vegada que utilitzem un dispositiu TD1204, l'haurem d'enregistrar en la xarxa SIGFOX prement el botó "Register Device in SIGFOX".

Un cop enregistrat el dispositiu, podrem enviar missatges, amb les limitacions que imposa SIGFOX de 140 missatges per dia.

Introduïm el missatge que volem enviar, que pot tenir un màxim de 10 caràcters ASCII, en la casella "Message". La casella "AT command" ens mostrarà la comanda SIGFOX corresponent. Per transmetre el missatge, premem "Transmit", i esperem a que l'aparell respongui "OK" pel terminal (pot trigar uns segons).



.Aplicacions del GPS

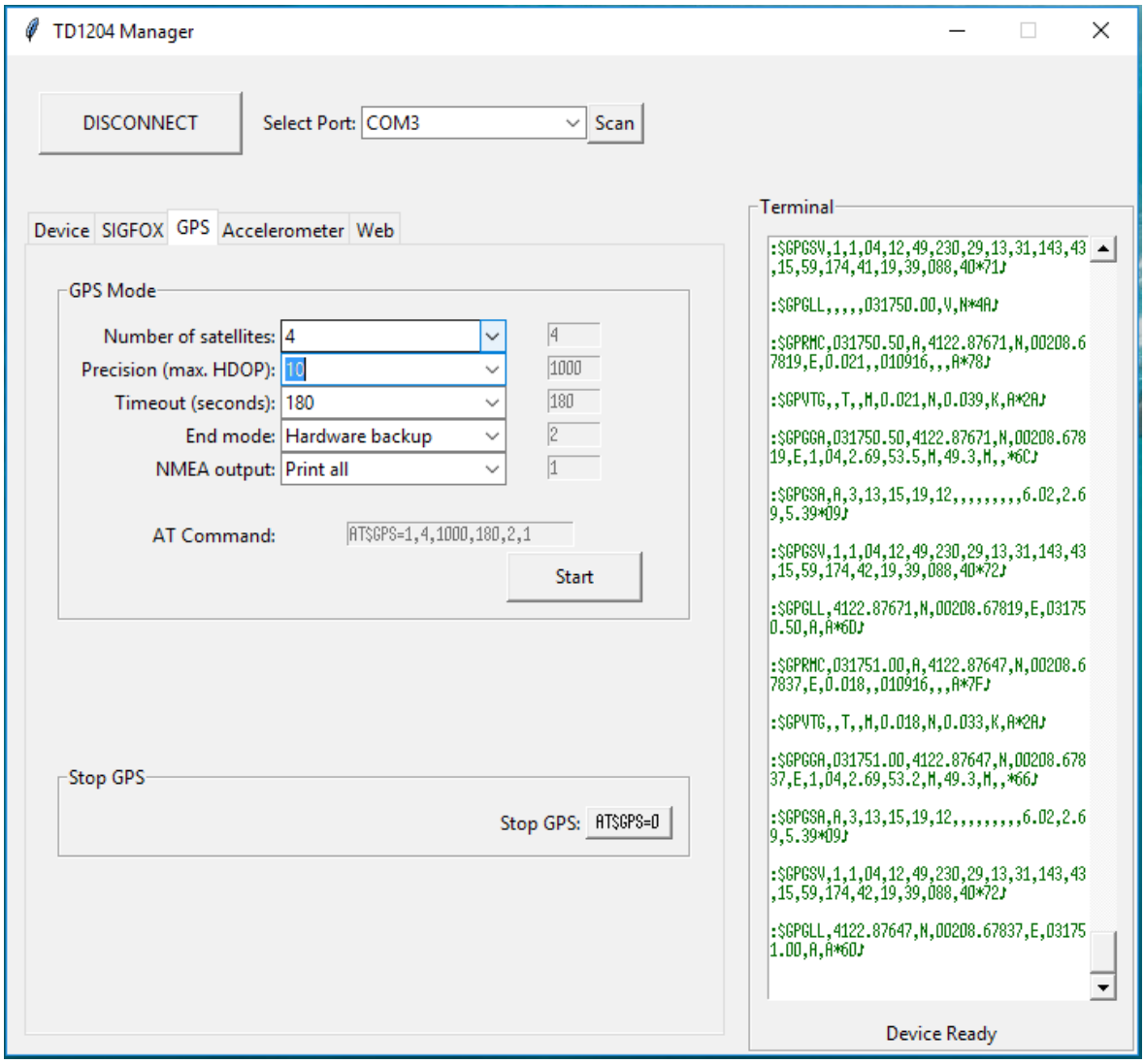
Per utilitzar el GPS de la placa TD1204 anem a la pestanya "GPS" de l'àrea de comandes.

Seleccionem els paràmetres per la geolocalització a través de GSP:

- Number of satellites: Nombre de satèl·lits mínim que ha de detectar la placa per efectuar la geolocalització.
- Precision (max. HDOP): Precisió, basada en la HDOP màxima acceptable.
- Timeout (seconds): Temps màxim per efectuar la geolocalització, si se supera aquest temps sense haver obtingut les coordenades, la cerca GPS finalitzarà igualment.
- End mode: Especifica què farà el gestor de GPS un cop finalitzada la cerca; es pot apagar, o passar a mode espera.
- NMEA output: Especifica les dades que retornarà la placa durant el procés de geolocalització (pot donar tots els càlculs NMEA, o només les coordenades finals; o bé no donar cap dada).

Per interrompre el procés de geolocalització abans de que finalitzi, podem prémer el botó "Stop GPS".

Un cop s'ha aconseguit una geolocalització efectiva, podem enviar les coordenades a través de la xarxa SIGFOX. Per fer-ho, anem a la pestanya "SIGFOX" i premem "Transmit Last Computed GPS Position".


TD1204 Manager
DISCONNECT
Select Port: COM3
Scan
Device
SIGFOX
GPS
Accelerometer
Web
GPS Mode
Number of satellites: 4
Precision (max. HDOP): 10
Timeout (seconds): 180
End mode: Hardware backup
NMEA output: Print all
4
1000
180
2
1
AT Command: AT$GPS=1,4,1000,180,2,1
Start
Stop GPS
Stop GPS: AT$GPS=0
Terminal
:$GPGSV,1,1,04,12,49,230,29,13,31,143,43,15,59,174,41,19,39,088,40*71
:$GPGLL,,,,,031750.00,V,N*4A
:$GPRMC,031750.50,A,4122.87671,N,00208.67819,E,0.021,,010916,,,A*78
:$GPVTG,,T,,M,0.021,N,0.039,K,A*2A
:$GPGGA,031750.50,4122.87671,N,00208.67819,E,1,04,2.69,53.5,M,49.3,M,,*6C
:$GPGSA,A,3,13,15,19,12,,,,,,,,,6.02,2.69,5.39*09
:$GPGSV,1,1,04,12,49,230,29,13,31,143,43,15,59,174,42,19,39,088,40*72
:$GPGLL,4122.87671,N,00208.67819,E,031750.50,A,A*6D
:$GPRMC,031751.00,A,4122.87647,N,00208.67837,E,0.018,,010916,,,A*7F
:$GPVTG,,T,,M,0.018,N,0.033,K,A*2A
:$GPGGA,031751.00,4122.87647,N,00208.67837,E,1,04,2.69,53.2,M,49.3,M,,*66
:$GPGSA,A,3,13,15,19,12,,,,,,,,,6.02,2.69,5.39*09
:$GPGSV,1,1,04,12,49,230,29,13,31,143,43,15,59,174,42,19,39,088,40*72
:$GPGLL,4122.87647,N,00208.67837,E,031751.00,A,A*6D
Device Ready

.Aplicacions de l'acceleròmetre

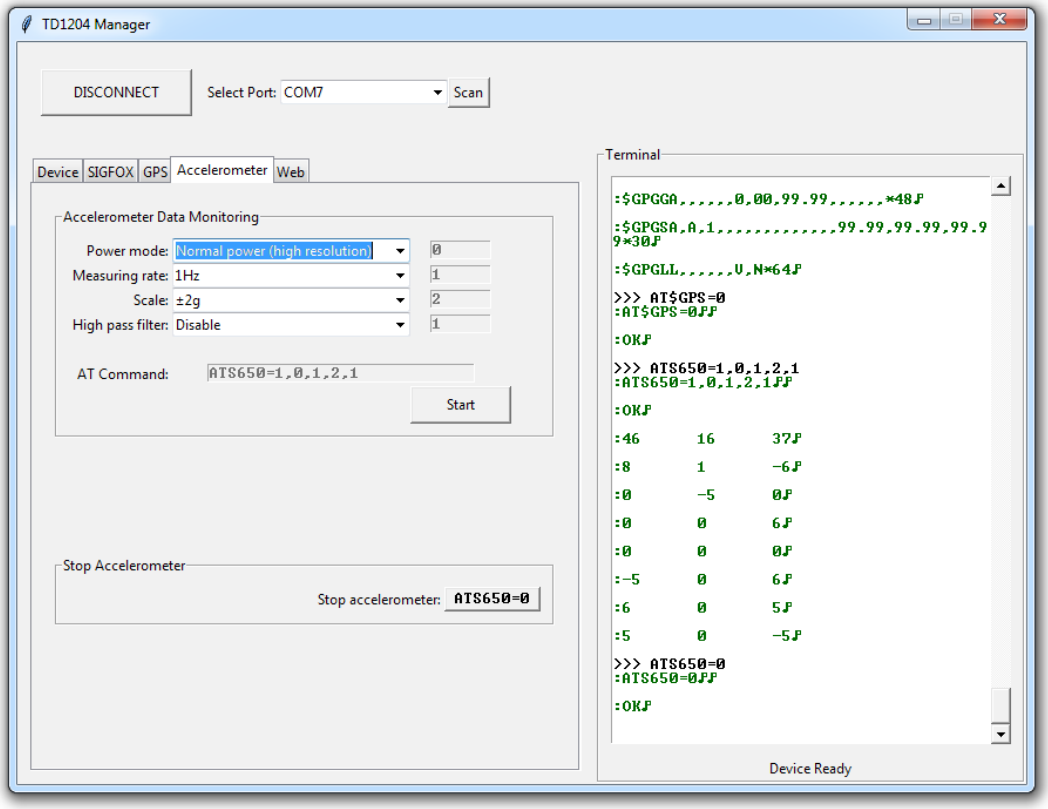
Per utilitzar l'acceleròmetre que incorpora la placa TD1204 anem a la pestanya "Accelerometer" de l'àrea de comandes.

Seleccionem els paràmetres de l'acceleròmetre:

- Power mode: Consum de potència de l'acceleròmetre, que
- Measuring rate: Periodicitat de la lectura.
- Scale: Escala de la lectura.
- High pass filter: Activació o desactivació del filtre passa alts.

Finalment premem "Start" per iniciar la lectura de l'acceleròmetre. La placa començarà a mostrar la lectura pel terminal de l'aplicació, amb la periodicitat que hem assignat.

Finalment per aturar la lectura de l'acceleròmetre, premem el botó "Stop aceelerometer".


TD1204 Manager
DISCONNECT
Select Port: COM7
Scan
Device
SIGFOX
GPS
Accelerometer
Web
Accelerometer Data Monitoring
Power mode: Normal power (high resolution) 0
Measuring rate: 1Hz 1
Scale: ±2g 2
High pass filter: Disable 1
AT Command: ATS650=1,0,1,2,1
Start
Stop Accelerometer
Stop accelerometer: ATS650=0
Terminal
:$GPGGA,,,,,,0,00,99.99,,,,,,*48
:$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99,99.99*30
:$GPGLL,,,,,,V,N*64
>>> AT$GPS=0
:AT$GPS=0
:OK
>>> ATS650=1,0,1,2,1
:ATS650=1,0,1,2,1
:OK
:46 16 37
:8 1 -6
:0 -5 0
:0 0 6
:0 0 0
:-5 0 6
:6 0 5
:5 0 -5
>>> ATS650=0
:ATS650=0
:OK
Device Ready

.Connexió amb el servidor central de SIGFOX

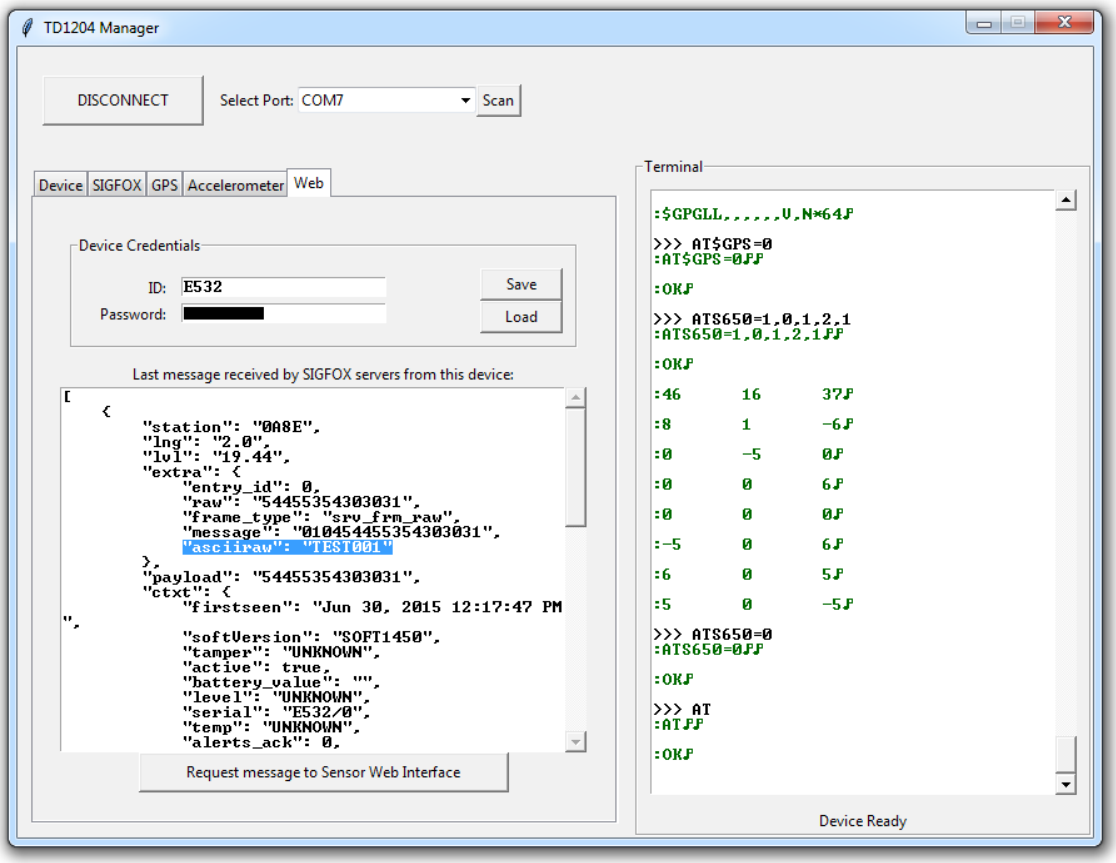
Des de la mateixa aplicació, podem contactar amb el servidor central de SIGFOX a través d'internet per verificar que un missatge que hem transmès per radio (a través de SIGFOX) s'ha rebut a la central.

Per fer-ho, anem a la pestanya "Web" de l'àrea de comandes.

Identifiquem el nostre dispositiu amb el codi d'usuari i contrasenya proporcionats pel fabricant, escrivint-los a les caselles "ID" i "Password" respectivament, o bé carregant-les des d'un arxiu amb el botó "Load".

Finalment premem el botó "Request message to Sensor Web Interface". Al cap d'un moment, en el requadre de text apareixeran les dades que SIGFOX proporciona sobre l'últim missatge que hem enviat.

Si es tractava d'un missatge de text ASCII, podem veure el contingut descodificat del missatge en el camp "asciiraw".



.Annex B: Codi font del mòdul TDManager.py

```
#from threading import Thread
#import os

import time
import serial
import serial.tools.list_ports


SENT="sent"
RECEIVED="received"
SYSTEM="system"
WAITING="waiting"
CONNECTED="connected"
DISCONNECTED="disconnected"
PERIOD=0.01



# Each AT command is an object of ATCom class:

class ATCom():
      def   __init__(self,   code="",   ack="OK",   pause=0.5,   *args,
**kwargs):
            self.code=code
            self.ack=ack
            self.pause=pause  #Pause after command, in seconds

      def get(self, param=""):
            if type(param) == type([]):
                  joined = ""
                  for element in param:
                        joined += element + " "
                  param=joined
            return self.code + param

      def getpause(self):
            return self.pause
```

```
    def getack(self):
        return self.ack



# List of AT commands:

ActivateEcho = ATCom("ATE1")
Ping = ATCom("AT")
ActivateResultCodes = ATCom("ATQ0")
ActivateTransmit = ATCom("ATS500=2")
SaveConfig = ATCom("AT&W")
Reset = ATCom("ATZ", ack="SYSSTART", pause=8)

ShowConfig = ATCom("AT&V")
RestoreFactory = ATCom("AT&F")

RequestModel = ATCom("ATI0")
RequestFirmwareDate = ATCom("ATI5")
RequestRFSerial = ATCom("ATI7")
RequestBasebandID = ATCom("ATI10")
RequestHardwareRev = ATCom("ATI11")
RequestFirmwareNum = ATCom("ATI13")
RequestRFChipNum = ATCom("ATI21")
RequestRFChipROMID = ATCom("ATI25")
RequestTemp = ATCom("ATI26")
RequestVolt = ATCom("ATI27")
RequestRFVolt = ATCom("ATI28")

SigfoxPresent = ATCom("SFP")
SigfoxID = ATCom("SFID")
SigfoxVersion = ATCom("SFv")

Register = ATCom("AT$REG")
TransmitRaw = ATCom("AT$RAW=")
TransmitGPS = ATCom("AT$GSND")

StopGPS = ATCom("AT$GPS=0")
GPSmode = ATCom("AT$GPS=1")

StopAccel = ATCom("ATS650=0")
AccelData = ATCom("ATS650=1")
```

```
# TDManager class encapsulates all the functionality
# of the interactions with the device through a COM port
# It will store all the traffic messages in a FIFO

class TDManager(serial.Serial):
      def __init__(self, *args, **kwargs):
            serial.Serial.__init__(self, *args, **kwargs)

            self.busy=False
            self.nextack=""
            self.stopwatch=False
            self.msgFIFO=[]

            self.PERIOD=PERIOD

            self.SENT=SENT
            self.RECEIVED=RECEIVED
            self.SYSTEM=SYSTEM

            self.ERROR="ERROR"

            self.WAITING=WAITING
            self.CONNECTED=CONNECTED
            self.DISCONNECTED=DISCONNECTED
            self.status=self.DISCONNECTED

      # Send a message (string) directly to the port:
      def psend(self, message="", ack="", pause=0.1):
            if not(self.isOpen()):
                  self.WriteToMsgFIFO("No Connection")
            elif self.busy:
                  self.WriteToMsgFIFO("Device    is    busy.    Please,
wait...")
            else:
                  self.WriteToMsgFIFO(message, self.SENT)
                  self.write((message+"\n").encode("ascii"))
```

```
                if ack != "":
                      self.nextack=ack
                      self.busy=True
                      self.status = self.WAITING
                else:
                      time.sleep(pause)

      # Send an AT Command (with its parameters) to the port:
      def psendcode(self, ATC, Param=""):
            self.psend(message=ATC.get(Param),  pause=ATC.getpause(),
ack=ATC.getack())

      # Watch the port by reading it periodically
      # and send received lines to terminal:
      def watch(self):
            self.status = self.CONNECTED
            while not self.stopwatch:
                  message=self.readline().decode("ascii")
                  if message!="":
                        self.WriteToMsgFIFO(message, self.RECEIVED)
                        if self.busy and (self.nextack in message):
                              self.busy=False
                              self.nextack=""
                              self.status=self.CONNECTED
                        elif self.busy and (self.ERROR in message):
                              self.busy=False
                              self.nextack=""
                              self.status=self.CONNECTED
                              self.WriteToMsgFIFO("ERROR:  Command  was
not accepted by the device.")
                  time.sleep(self.PERIOD)
            self.stopwatch=False
            self.status=self.DISCONNECTED

      # Stop watching the port and then close it:
      def stop(self):
            self.stopwatch=True
            time.sleep(self.PERIOD)
            self.busy=False
            self.close()

      # Set COM port to manage
```

```
    def setport(self, comport):
        self.port=comport

    # Returns whether the device is busy
    def isbusy(self):
        return self.busy

    # Send a list of AT commands to the device
    def FIFOpsendcode(self, comlist=[]):
        for element in comlist:
            self.psendcode(element)
            while self.isbusy():
                time.sleep(self.PERIOD)
            if not self.isOpen():
                break

    def GetStatus(self):
        return self.status

    def WriteToMsgFIFO(self, message="", msgtype=""):
        if msgtype==="":
            msgtype=self.SYSTEM
        print((message, msgtype))
        self.msgFIFO.append((message,msgtype))

    def GetMsgFIFO(self):
        messages=self.msgFIFO
        self.msgFIFO=[]
        return messages

    def DeviceInit(self):
        self.WriteToMsgFIFO("Initializing     Device,     Please
Wait...")

        self.FIFOpsendcode([ActivateEcho,                    Ping,
ActivateResultCodes, ActivateTransmit,
            SaveConfig, Reset, Ping])

        if self.isOpen():
            self.WriteToMsgFIFO("Initialization Complete")
            self.WriteToMsgFIFO("Device is Ready")
        else:
```

```
                self.WriteToMsgFIFO("Disconnected,   Initialization
Cancelled")



# Routine to create and return a TDManager instance
# with the configuration set for a TD1204 device.
# The COM port is not provided so the port will not be open.
# To open the port must provide port number with the method:
# TDManager.port(port_number); then call PortManager.open()

def CreateTDManager():
     port = TDManager(
          baudrate=9600,
         parity=serial.PARITY_NONE,
         stopbits=serial.STOPBITS_ONE,
         timeout=1,
         bytesize=serial.EIGHTBITS
     )
     return port
```

.Annex C: Codi font del mòdul TDManagerGUI.py

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog

import time
from threading import Thread

import serial.tools.list_ports

import binascii

import webrequests
import os
import json

import TD1204Manager as td




# The GUI period will be a bit slower than the TD manager to avoid
conflicts
PERIOD= 1.2 * td.PERIOD


# Terminal is a subclass of the tkinter text widget
# Terminal objects receive and display messages (strings)
# using different formats for: SENT, RECEIVED, SYSTEM

class Terminal(tk.Text):

      def __init__(self, *args, **kwargs):
            tk.Text.__init__(self, *args, **kwargs)
            self.config(state=tk.DISABLED)

            self.SENT = td.SENT
            self.RECEIVED = td.RECEIVED
```

```
            self.SYSTEM = td.SYSTEM

            self.config(font=("Terminal",11))
            self.tag_config(self.SENT, foreground="black")
            self.tag_config(self.RECEIVED, foreground="dark green")
            self.tag_config(self.SYSTEM, foreground="sienna4")

            self.tprint("Terminal Ready")

      # Display message (print on terminal):
      def tprint(self, message="", messagetype=""):
            if messagetype==self.SENT:
                  message= ">>> " + message + "\n"
                  mtag=self.SENT
            elif messagetype==self.RECEIVED:
                  message= ":" + message + "\n"
                  mtag=self.RECEIVED
            else:
                  message="*** " + message + " ***" + "\n\n"
                  mtag=self.SYSTEM

            self.config(state=tk.NORMAL)
            self.insert(tk.END, message, mtag)
            self.see(tk.END)
            self.config(state=tk.DISABLED)

      # Gets the messages from a TDmanager object as a list
      # and prints them on the terminal
      def RequestMessages(self, port):
            messages=port.GetMsgFIFO()
            for element in messages:
                  self.tprint(element[0], element[1])



# This class will display the status of the Port Manager as a label
class StatusLabel(tk.Label):
      def __init__(self, *args, **kwargs):
            tk.Label.__init__(self, *args, **kwargs)
            self.status=tk.StringVar()
            self.config(textvariable=self.status)
```

```
            self.status.set("No Device")
      def SetDisconnected(self):
            self.status.set("Disconnected")
      def SetConnected(self):
            self.status.set("Device Ready")
      def SetWaiting(self):
            self.status.set("Device Busy")
      def SetError(self):
            self.status.set("Device Error")
      def RequestStatus(self, port):
            tdstatus=port.GetStatus()
            if tdstatus==td.WAITING:
                  self.SetWaiting()
            elif tdstatus==td.CONNECTED:
                  self.SetConnected()
            elif tdstatus==td.DISCONNECTED:
                  self.SetDisconnected()
            else:
                  self.SetError()



# Port Selector Class

class PortSelector(tk.Frame):
      def __init__(self, terminal=None, *args, **kwargs):
            tk.Frame.__init__(self, *args, **kwargs)
            self.terminal=terminal
            self.ports=[]
            self.portnames=[]
            self.portslabel=tk.Label(self, text="Select Port:")
            self.portbox=ttk.Combobox(self,    values=self.portnames,
state="readonly")
            self.scanbutton=tk.Button(self,              text="Scan",
command=self.scan)
            self.portslabel.pack(side=tk.LEFT)
            self.portbox.pack(side=tk.LEFT)
            self.scanbutton.pack(side=tk.LEFT)

      def scan(self):
            self.terminal.tprint("Scanning Ports...")
            self.ports = list(serial.tools.list_ports.comports())
```

```
            self.portnames=[]
            index=0
            detection=False
            for port in self.ports:
                  self.portnames.append(port[0])
                  self.portbox.config(values=self.portnames)
                  if "FTDIBUS" in port[2]:
                        self.terminal.tprint("Device detected in: " +
port[0])
                        self.portbox.current(newindex=index)
                        detection=True
                  index +=1
            if detection:
                  self.terminal.tprint("Port: " + self.get() + " has
been selected.")
            else:
                  self.terminal.tprint("Port scan completed, but no
device was found. Please connect your device and run another scan,
or select port manually.")

      def get(self):
            try:
                  return self.portnames[self.portbox.current()]
            except:
                  self.terminal.tprint("No valid port selected")



# This class creates a file manager that saves objects in json
format.
# We will use it to save and load the device credentials.
class SaveLoad():
      def __init__(self):
            self.filepath      =      os.path.dirname(__file__)      +
"\\Credentials"
      def save(self, values=""):
            jsonvalues=json.dumps(values)
            try:
                  fout=tk.filedialog.asksaveasfile(mode="w",
                        defaultextension=".dcr",            initialdir
=self.filepath,
```

```
                        initialfile="TD1204",    title="Save    Device
Credentials on File")
                  json.dump(jsonvalues, fout)
                  fout.close()
            except:
                  print("Failed to save file.")
      def load(self):
            fin=tk.filedialog.askopenfile(mode="r",
initialdir=self.filepath, title="Load Device Credentials File")
            filedata = json.load(fin)
            fin.close()
            filedata= json.loads(filedata)
            return filedata




#### FUNCTIONS ##############################################

# Tells the td manager to start the initialization routine for the
device
# It will get called immediately after opening the port
def DeviceInit(port):
      port.DeviceInit()

# Tells the td manager to start watching the port for messages
received
def ActivateWatch(port):
      port.watch()

# Convert ascii text to a list of hex code strings
# That can be transmitted through SIGFOX
def HexList(text):
      hexlist=[]
      for character in text:
            hexcode=binascii.hexlify(character.encode("ascii"))
            hexlist.append(hexcode.decode("ascii"))
      return hexlist


# Tells the terminal and status GUI elements to start watching the
TD manager
```

```
# for status and messages received
def GUIWatch(port, term, status):
      global GUIWatchFlag
      while GUIWatchFlag:
            term.RequestMessages(port)
            status.RequestStatus(port)
            time.sleep(PERIOD)




##### GUI COMMAND FUNCTIONS ##################################

# These functions will be called by
# the interactive elements of the GUI.

def PingPort():
      port.psendcode(Ping)

def TransmitTest():
      port.psendcode(TransmitRaw, HexList("TEST"))

def OpenPort():
      if not port.isOpen():
            try:
                  term.tprint("Opening Port: " + COMport.get())
                  port.setport(COMport.get())
                  port.open()
                  term.tprint("Port Open")
                  # Start up the port watcher on a different thread
                  # that will continue to run on the background
                  threadwatch      =      Thread(target=ActivateWatch,
args=(port,))
                  threadwatch.start()
                  # Call routine for the device initialization:
                  threadinit        =        Thread(target=DeviceInit,
args=(port,))
                  threadinit.start()
                  ConnectButtonText.set("DISCONNECT")
            except:
                  term.tprint("ERROR: Failed to Open Port")
```

```
        else:
                port.stop()
                term.tprint("Disconnected")
                ConnectButtonText.set("CONNECT")

def SaveCredentials():
        credentials={"ID":IDvar.get(), "Pass":Passvar.get()}
        filemanager.save(credentials)

def LoadCredentials():
        credentials=filemanager.load()
        IDvar.set(credentials["ID"])
        Passvar.set(credentials["Pass"])


# Functions for the test buttons used for debugging

def Test1():
        port.psend(message="AT")

def Test2():
        port.psend(message="AT")





##### GUI CLASSES ##########################################

# Classes that we will use on the design of GUI elements

# Displays a label with an AT command name and a button.
# Pressing the button will send the command to the device.
class ATFrame(tk.Frame):
        def __init__(self, ATCommand, labeltext, *args, **kwargs):
                tk.Frame.__init__(self, *args, **kwargs)
                self.ATCommand=ATCommand
                self.label=tk.Label(self, text=labeltext)
                self.label.pack(side=tk.LEFT)
                self.button=tk.Button(self, text=ATCommand.get(),
```

```
                command=self.ButtonPress,                    width=9,
font=("Terminal",10))
          self.button.pack(side=tk.LEFT)
     def ButtonPress(self):
          port.psendcode(self.ATCommand)

# Places a series of tkinter entry fields next to each other in a
container.
# We use it to display the hex conversion values in the raw transmit
frame
class Multientry():
     def __init__(self, container, number, *args, **kwargs):
          self.fields=[]
          self.vars=[]
          for index in range(0,number):
               self.vars.append(tk.StringVar())
               self.fields.append(tk.Entry(container))
               self.fields[index].config(width=2,
font=("Terminal",10), state=tk.DISABLED)

     self.fields[index].config(textvariable=self.vars[index])
               self.fields[index].pack(side=tk.LEFT,     fill=tk.X,
expand=True, anchor=tk.W)
     def set(self, values):
          for index in range(0,len(self.fields)):
               if index < len(values):
                    self.vars[index].set(values[index])
               else:
                    self.vars[index].set("")
     def get(self):
          values = []
          for index in range(0, len(self.fields)):
               values.append(self.vars[index].get())
          return values


##### GUI CONSTRUCTION ##############################################
```

```
# *** Basic Structure ***

root=tk.Tk()
root.title("TD1204 Manager GUI")
root.resizable(0,0)


# Main Frame

mainframe=tk.Frame(root)
mainframe.pack(fill=tk.BOTH, expand=True)


# Button Frame
# Contains the interactions to manage the connection to the device

bframe=tk.Frame(mainframe, padx=20, pady=20)
bframe.pack(fill=tk.BOTH, expand=True)

ConnectButtonText = tk.StringVar()
ConnectButtonText.set("CONNECT")
ConnectButton=tk.Button(bframe, command=OpenPort,
      textvariable=ConnectButtonText, width=15, padx=8, pady=8)
ConnectButton.pack(side=tk.LEFT)



# Panels Container
# Main area on the app, will contain the terminal and interaction
elements

panels=tk.PanedWindow(mainframe, orient=tk.HORIZONTAL)
panels.pack(fill=tk.BOTH, expand=True)



# Test Frame
# For debugging and testing commands
''''
testframe=tk.Frame(mainframe)
testframe.pack(fill=tk.X, expand=True)
```

```
testb=tk.Button(testframe, text="PING", command=PingPort)
testb.pack(side=tk.LEFT)

testb2=tk.Button(testframe, text="TEST 1", command=Test1)
testb2.pack(side=tk.LEFT)

testb3=tk.Button(testframe, text="TEST 2", command=Test2)
testb3.pack(side=tk.LEFT)
'''



# Interaction Frame

iframe=tk.Frame(panels, padx=10, pady=10)
panels.add(iframe)


# Terminal Frame

tframe=tk.LabelFrame(panels, text="Terminal")
panels.add(tframe)

tframe1=tk.Frame(tframe, padx=10, pady=10)
tframe2=tk.Frame(tframe)
tframe1.pack()
tframe2.pack()

term = Terminal(tframe1, width=40, height=40)
term.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

termscroll=tk.Scrollbar(tframe1)
termscroll.pack(side=tk.RIGHT, fill=tk.Y)

term.config(yscrollcommand=termscroll.set)
termscroll.config(command=term.yview)

status = StatusLabel(tframe2)
status.pack(side=tk.BOTTOM)
```

```
# Notebook

notebook = ttk.Notebook(iframe)
notebook.pack(fill=tk.BOTH, expand=True)

configframe=tk.Frame(notebook, padx=20, pady=20)
notebook.add(configframe, text="Device")

transmitframe=tk.Frame(notebook, padx=20, pady=20)
notebook.add(transmitframe, text="SIGFOX")

gpsframe=tk.Frame(notebook, padx=20, pady=20)
notebook.add(gpsframe, text="GPS")

accelframe=tk.Frame(notebook, padx=20, pady=20)
notebook.add(accelframe, text="Accelerometer")

webframe=tk.Frame(notebook, padx=20, pady=20)
notebook.add(webframe, text="Web")



# *** Config Frame Construction ***

deviceconfigframe=tk.LabelFrame(configframe,            text="Device
Configuration", padx=10, pady=10)
deviceconfigframe.pack(side=tk.TOP,     fill=tk.X,     anchor=tk.NW,
expand=True)

ConfigButtons = []
ConfigButtons.append(ATFrame(td.ShowConfig,                "Display
Configuration:", deviceconfigframe))
ConfigButtons.append(ATFrame(td.SaveConfig,  "Save  Configuration:",
deviceconfigframe))
ConfigButtons.append(ATFrame(td.Reset,  "Reset  Device  with  Saved
Configuration:", deviceconfigframe))
ConfigButtons.append(ATFrame(td.RestoreFactory,   "Restore   Factory
Settings:", deviceconfigframe))

for item in ConfigButtons:
```

```
    item.pack(anchor=tk.NE)


requestsframe=tk.LabelFrame(configframe, text="Device Information",
padx=10, pady=10)
requestsframe.pack(side=tk.TOP, fill=tk.X, anchor=tk.NW,
expand=True)

RequestsButtons = []
RequestsButtons.append(ATFrame(td.RequestModel, "Request Device
Model:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestFirmwareDate, "Request
Firmware Date:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestRFSerial, "Request RF
Serial Number:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestBasebandID, "Request
Baseband ID:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestHardwareRev, "Request
Hardware Revision:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestFirmwareNum, "Request
Firmware Number:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestRFChipNum, "Request RF Chip
Number:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestRFChipROMID, "Request RF
Chip ROM ID:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestTemp, "Request Device
Temperature:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestVolt, "Request Device
Voltage:", requestsframe))
RequestsButtons.append(ATFrame(td.RequestRFVolt, "Request Last RF
Voltage:", requestsframe))

for item in RequestsButtons:
    item.pack(anchor=tk.NE)


devicetestframe=tk.LabelFrame(configframe, text="Device Test",
padx=10, pady=10)
devicetestframe.pack(side=tk.TOP, fill=tk.X, anchor=tk.NW,
expand=True)

devicetest=ATFrame(td.Ping, "Ping Device:", devicetestframe)
```

```
devicetest.pack(anchor=tk.NE)



# *** Transmit Frame ***

registerframe=tk.LabelFrame(transmitframe,      text="Registration",
padx=10, pady=10)
registerframe.pack(side=tk.TOP,       fill=tk.X,       anchor=tk.NW,
expand=True)

registerdevice=ATFrame(td.Register,  "Register  Device  in  SIGFOX:",
registerframe)
registerdevice.pack(anchor=tk.NE)



# *** Raw Transmit Frame Construction ***

rawframe = tk.LabelFrame(transmitframe, text="Transmit Raw Message",
padx=10, pady=10)
rawframe.pack(side=tk.TOP, fill=tk.X, anchor=tk.NW, expand=True)


lmessage = tk.Label(rawframe, text="Message:")
lhex = tk.Label(rawframe, text="Hex encoding:")
latc = tk.Label(rawframe, text="AT command:")

emessage = tk.Entry(rawframe, font=("Terminal",10))
hexframe = tk.Frame(rawframe)
eatc  =  tk.Entry(rawframe,  state=tk.DISABLED,  font=("Terminal",10),
width=36)

bsend = tk.Button(rawframe, text="Transmit")

lmessage.grid(   column=0, row=1, sticky=tk.E, padx=4, pady=4)
lhex.grid(       column=0, row=2, sticky=tk.E, padx=4, pady=4)
latc.grid(       column=0, row=3, sticky=tk.E, padx=4, pady=4)
emessage.grid(   column=1, row=1, sticky=tk.EW, padx=10, pady=4)
```

```
hexframe.grid(   column=1, row=2, sticky=tk.EW, padx=10, pady=4)
eatc.grid(       column=1, row=3, sticky=tk.EW, padx=10, pady=4)
bsend.grid(      column=1,  row=4,  sticky=tk.SE,  padx=10,  pady=4,
      ipadx=16, ipady=4)

rawframe.columnconfigure(1, weight=1)
rawframe.rowconfigure(4, weight=1)

HexEntry = Multientry(hexframe, 10)



# *** Raw Transmit Functionality ***

# Assignation of StringVars:
inputmessage = tk.StringVar()
emessage.config(textvariable=inputmessage)

rawtosend = tk.StringVar()
eatc.config(textvariable=rawtosend)

# When the entry message text changes, calculate and place the
values on HexEntry:
def PlaceValuesTransmitRaw(*args, **kwargs):
      # Clear fields
      rawtosend.set("")
      HexEntry.set([])
      message=inputmessage.get()

      # Verify the message can be encoded in ascii
      try:
            message.encode("ascii")
      except:
            term.tprint("ERROR: Only ASCII characters can be encoded
for raw transmition")
            return

      # Check the length of the message
      if len(message)>10:
            term.tprint("ERROR:   Message   too   long   for   raw
transmition, maximum length is 10 bytes")
            message=message[0:10]
```

```
      HexEntry.set(HexList(message))
      rawtosend.set(td.TransmitRaw.get(HexList(message)))

inputmessage.trace("w", PlaceValuesTransmitRaw)


# Button sends Transmit Raw AT Command to device

def SendTransmitRaw():
      port.psendcode(td.TransmitRaw, HexEntry.get())

bsend.config(command=SendTransmitRaw)




# *** Transmit GPS Position ***


transmitGPSframe=tk.LabelFrame(transmitframe,   text="Transmit   GPS
Position", padx=10, pady=10)
transmitGPSframe.pack(side=tk.TOP,      fill=tk.X,      anchor=tk.NW,
expand=True)

GPStransmit=ATFrame(td.TransmitGPS,  "Transmit  Last  Computed  GPS
Position:", transmitGPSframe)
GPStransmit.pack(anchor=tk.NE)




# *** Web Interface Frame ***

IDvar=tk.StringVar()
Passvar=tk.StringVar()

credentialsframe=tk.LabelFrame(webframe, text="Device Credentials")
credentialsframe.pack(fill=tk.X, padx=10, pady=10)
```

```
credentialsframe1=tk.Frame(credentialsframe)
credentialsframe1.pack(side=tk.LEFT, padx=10, pady=10)

credentialsframe2=tk.Frame(credentialsframe)
credentialsframe2.pack(side=tk.RIGHT, padx=10, pady=10)

IDLab=tk.Label(credentialsframe1, text="ID:", padx=10)
PassLab=tk.Label(credentialsframe1, text="Password:", padx=10)

IDField=tk.Entry(credentialsframe1,            font=("Terminal",10),
textvariable=IDvar)
PassField=tk.Entry(credentialsframe1,     font=("Terminal",     10),
textvariable=Passvar)

SaveButton=tk.Button(credentialsframe2,     text="Save",     width=8,
command=SaveCredentials)
LoadButton=tk.Button(credentialsframe2,     text="Load",     width=8,
command=LoadCredentials)

IDField.grid(column=1, row=0)
PassField.grid(column=1, row=1)
IDLab.grid(column=0, row=0, sticky=tk.E)
PassLab.grid(column=0, row=1, sticky=tk.E)
SaveButton.grid(column=2, row=0)
LoadButton.grid(column=2, row=1)


webmessagelabel=tk.Label(webframe,  text="Last  message  received  by
SIGFOX servers from this device:")
webmessagelabel.pack()

webmessageframe=tk.Frame(webframe)
webmessageframe.pack()

messagescroll=tk.Scrollbar(webmessageframe)
messagescroll.pack(side=tk.RIGHT, fill=tk.Y)

messagetext=tk.Text(webmessageframe,   width=50,   font=("Terminal",
10), state=tk.DISABLED)
messagetext.pack(side=tk.LEFT)

messagetext.config(yscrollcommand=messagescroll.set)
```

```
messagescroll.config(command=messagetext.yview)

def requestmessage():
      webinterface=webrequests.WebInterface()
      webinterface.gettoken(IDvar.get(), Passvar.get())
      messagetext.config(state=tk.NORMAL)
      messagetext.delete(0.0, tk.END)
      messagetext.insert(tk.END, webinterface.getmessage())
      messagetext.config(state=tk.DISABLED)

webbutton=tk.Button(webframe, text="Request message to Sensor Web
Interface", width=40, padx=4, pady=4, command=requestmessage)
webbutton.pack()




# *** GPS Frame ***


def GPSObtainParameters(index, value, op):
      GPSATCommand=td.GPSmode.get()
      for i in range(0,5):
            key = str(gpschoice[i].get())
            dictionary=gpsparamdict[i]
            gpsparam[i].set(dictionary[key])
            GPSATCommand+=","+dictionary[key]
      gpsATvar.set(GPSATCommand)


gpsmodeframe=tk.LabelFrame(gpsframe, text="GPS Mode", padx=10,
pady=10)
gpsmodeframe.pack(side=tk.TOP, fill=tk.X, anchor=tk.NW, expand=True)

gpsparamdict=[]
gpsparamli=[]
gpschoice=[]
gpslab=[]
gpscbx=[]
gpsfield=[]
```

```
gpsparam=[]

gpslabtext=["Number of satellites:",
"Precision (max. HDOP):",
"Timeout (seconds):",
"End mode:",
"NMEA output:"]

gpsparamli.append(["4", "5", "6", "7", "8"])
gpsparamli.append(["1 (Very High)", "2", "5", "10", "20 (Low)"])
gpsparamli.append(["60", "180", "300", "Unlimitted"])
gpsparamli.append(["Stop", "Leave on", "Hardware backup"])
gpsparamli.append(["No print", "Print all", "Print only when
finished"])

gpsparamdict.append({"4":"4", "5":"5", "6":"6", "7":"7", "8":"8"})
gpsparamdict.append({"1 (Very High)":"100", "2":"200", "5":"500",
"10":"1000", "20 (Low)":"2000"})
gpsparamdict.append({"60":"60", "180":"180", "300":"300",
"Unlimitted":"65535"})
gpsparamdict.append({"Stop":"0", "Leave on":"1", "Hardware
backup":"2"})
gpsparamdict.append({"No print":"0", "Print all":"1", "Print only
when finished":"2"})

gpsinitial=[0,3,1,2,1]

gpsATvar=tk.StringVar()


for index in range(0,5):
      gpschoice.append(tk.StringVar())

      gpsparam.append(tk.StringVar())

      gpslab.append(tk.Label(gpsmodeframe))
      gpslab[index].config(text=gpslabtext[index])

      gpscbx.append(ttk.Combobox(gpsmodeframe))
      gpscbx[index].config(textvar=gpschoice[index],
values=gpsparamli[index])
      gpscbx[index].current(gpsinitial[index])
```

```
      gpschoice[index].trace("w",GPSObtainParameters)

      gpsfield.append(tk.Entry(gpsmodeframe,       state=tk.DISABLED,
width=6, font=("Terminal", 10)))
      gpsfield[index].config(textvar=gpsparam[index])


# We set the values of the comboboxes again
# to force the tracing function to calculate the parameters
for index in range(0,5):
      gpscbx[index].current(gpsinitial[index])



gpslab[0].grid(column=0, row=0, sticky=tk.E)
gpslab[1].grid(column=0, row=1, sticky=tk.E)
gpslab[2].grid(column=0, row=2, sticky=tk.E)
gpslab[3].grid(column=0, row=3, sticky=tk.E)
gpslab[4].grid(column=0, row=4, sticky=tk.E)

gpscbx[0].grid(column=1, row=0)
gpscbx[1].grid(column=1, row=1)
gpscbx[2].grid(column=1, row=2)
gpscbx[3].grid(column=1, row=3)
gpscbx[4].grid(column=1, row=4)

gpsfield[0].grid(column=2, row=0)
gpsfield[1].grid(column=2, row=1)
gpsfield[2].grid(column=2, row=2)
gpsfield[3].grid(column=2, row=3)
gpsfield[4].grid(column=2, row=4)


gpslab6=tk.Label(gpsmodeframe)
gpsfieldAT=tk.Entry(gpsmodeframe,                  textvar=gpsATvar,
state=tk.DISABLED, width=28, font=("Terminal", 10))
gpslabAT=tk.Label(gpsmodeframe, text="AT Command:")
gpsstart=tk.Button(gpsmodeframe,  text="Start",  width=10,  padx=4,
pady=4)

gpslab6.grid(column=0, row=5)
gpslabAT.grid(column=0, row=6, sticky=tk.E)
```

```
gpsfieldAT.grid(column=1, row=6, columnspan=2)
gpsstart.grid(column=2, row=7)


def GPSStart():
      gpsmodeparam=""
      for i in range(0,5):
            gpsmodeparam += "," + gpsparam[i].get()
      port.psendcode(td.GPSmode, gpsmodeparam)


gpsstart.config(command=GPSStart)



# Stop GPS frame and button
stopGPSframe=tk.LabelFrame(gpsframe,   text="Stop   GPS",   padx=10,
pady=10)
stopGPSframe.pack(side=tk.TOP, fill=tk.X, anchor=tk.NW, expand=True)

stopGPSinteraction=ATFrame(td.StopGPS, "Stop GPS:", stopGPSframe)
stopGPSinteraction.pack(anchor=tk.NE)





# *** Accelerometer Frame ***




def AccelObtainParameters(index, value, op):
      accelATCommand=td.AccelData.get()
      for i in range(0,4):
            key = str(accelchoice[i].get())
            dictionary=accelparamdict[i]
            accelparam[i].set(dictionary[key])
            accelATCommand+=","+dictionary[key]
      accelATvar.set(accelATCommand)
```

```
accelmodeframe=tk.LabelFrame(accelframe, text="Accelerometer Data Monitoring", padx=10, pady=10)
accelmodeframe.pack(side=tk.TOP, fill=tk.X, anchor=tk.NW, expand=True)

accelparamdict=[]
accelparamli=[]
accelchoice=[]
accellab=[]
accelcbx=[]
accelfield=[]
accelparam=[]

accellabtext=["Power mode:",
"Measuring rate:",
"Scale:",
"High pass filter:"]

accelparamli.append(["Normal power (high resolution)", "Low power (low resolution)"])
accelparamli.append(["1Hz", "10Hz", "25Hz", "50Hz"])
accelparamli.append(["±2g", "±4g", "±8g", "±16g"])
accelparamli.append(["Enable", "Disable"])

accelparamdict.append({"Normal power (high resolution)":"0", "Low power (low resolution)":"1"})
accelparamdict.append({"1Hz":"1", "10Hz":"2", "25Hz":"3", "50Hz":"4"})
accelparamdict.append({"±2g":"2", "±4g":"4", "±8g":"8", "±16g":"16"})
accelparamdict.append({"Enable":"0", "Disable":"1"})

accelinitial=[0,0,0,1]

accelATvar=tk.StringVar()


for index in range(0,4):
      accelchoice.append(tk.StringVar())

      accelparam.append(tk.StringVar())
```

```
      accellab.append(tk.Label(accelmodeframe))
      accellab[index].config(text=accellabtext[index])

      accelcbx.append(ttk.Combobox(accelmodeframe))
      accelcbx[index].config(textvar=accelchoice[index],
values=accelparamli[index], width=30)
      accelcbx[index].current(accelinitial[index])
      accelchoice[index].trace("w",AccelObtainParameters)

      accelfield.append(tk.Entry(accelmodeframe,   state=tk.DISABLED,
width=6, font=("Terminal", 10)))
      accelfield[index].config(textvar=accelparam[index])


# We set the values of the comboboxes again
# to force the tracing function to calculate the parameters
for index in range(0,4):
      accelcbx[index].current(accelinitial[index])



accellab[0].grid(column=0, row=0, sticky=tk.E)
accellab[1].grid(column=0, row=1, sticky=tk.E)
accellab[2].grid(column=0, row=2, sticky=tk.E)
accellab[3].grid(column=0, row=3, sticky=tk.E)

accelcbx[0].grid(column=1, row=0)
accelcbx[1].grid(column=1, row=1)
accelcbx[2].grid(column=1, row=2)
accelcbx[3].grid(column=1, row=3)

accelfield[0].grid(column=2, row=0)
accelfield[1].grid(column=2, row=1)
accelfield[2].grid(column=2, row=2)
accelfield[3].grid(column=2, row=3)


accellab6=tk.Label(accelmodeframe)
accelfieldAT=tk.Entry(accelmodeframe,            textvar=accelATvar,
state=tk.DISABLED, width=28, font=("Terminal", 10))
accellabAT=tk.Label(accelmodeframe, text="AT Command:")
```

```
accelstart=tk.Button(accelmodeframe, text="Start", width=10, padx=4,
pady=4)

accellab6.grid(column=0, row=5)
accellabAT.grid(column=0, row=6, sticky=tk.E)
accelfieldAT.grid(column=1, row=6, columnspan=2)
accelstart.grid(column=2, row=7)


def AccelStart():
      accelmodeparam=""
      for i in range(0,4):
            accelmodeparam += "," + accelparam[i].get()
      port.psendcode(td.AccelData, accelmodeparam)


accelstart.config(command=AccelStart)





# Stop accel frame and button
stopaccelframe=tk.LabelFrame(accelframe, text="Stop Accelerometer",
padx=10, pady=10)
stopaccelframe.pack(side=tk.TOP,      fill=tk.X,      anchor=tk.NW,
expand=True)

stopaccelinteraction=ATFrame(td.StopAccel,  "Stop  accelerometer:",
stopaccelframe)
stopaccelinteraction.pack(anchor=tk.NE)





##### EXECUTION #####################################

# Create TD manager object
port = td.CreateTDManager()
```

```
# Tell terminal and status to start watching the TD manager
# It will run on the background, so we create a thread for it
GUIWatchFlag=True
threadGUIwatch = Thread(target=GUIWatch, args=(port,term,status))
threadGUIwatch.start()

# Create file manager
filemanager = SaveLoad()

# Start port scanner
COMport = PortSelector(term, bframe, padx=10, pady=10)
COMport.pack(side=tk.LEFT)
COMport.scan()

# Open main window
root.mainloop()

# After the main window is closed:

# Stop elements from watching TDManager
# By setting a flag that will stop the watch thread
GUIWatchFlag=False

# Tell TDManager to close the port
print(port.GetMsgFIFO())
port.stop()
```

.Annex C: Codi font del mòdul webrequests.py

```
import sensor
import json


class WebInterface(sensor.Device):
      def __init__(self, *args, **kwargs):
            sensor.Device.__init__(self, *args, **kwargs)

      def gettoken(self, tdid, tdpass):
            self.set_device(tdid, tdpass)

      def getmessage(self):
            message = self.get_history(1)
            if message==0:
                  return "ERROR: Failed to retrieve message.\nPlease,
verify your credentials."
            else:
                  return json.dumps(message["items"], indent=4)
```