round solutions

More with wireless ->
More Expertise & More Service

Nano Tracker

Integration Manual

Rev .04 – 20-03-2017

Introduction:

Nano Tracker is a tiny tracking device. It consists of an embedded tracking device and a battery. The purpose of this device is to track people or items discreetly for the purpose of security and safety.

The tracking device can be completely concealed inside clothing, or inside any nonmetallic item to track. It uses a Quad band GPRS modem a SIRF 4 GPS chipset. It is battery powered and programmable. The device has an accelerometer on board to detect when the device is not moving to shut down the modem and put the GPS in sleep mode to save battery power. Battery can be charged via USB. Data is transmitted at preset intervals in HTTP posts. The device can be fully configured and diagnosed by USB or SMS.

Hardware Specifications:

- Quad Band GSM/GPRS modem
- SIRF 4 GPS Chipset: -163dBm sensitivity
- Micro - B USB Interface for configuration and debugging connected to the 2G Telit module
- Onboard Accelerometer for movement detection and power saving
- 3.7V – 320 mAh LI-Ion battery
- Nano SIM Card holder
- Internal memory: 32KB EEPROM
- Ready for integration with Cloud platforms
- Data includes Longitude, Latitude, speed, date, time, and battery voltage.
- Dimensions: 4cm x 2,3cm x 0,8 cm (including GPS antennas)
- UFL Connector for GSM Antenna
- Onboard active GPS antenna
- Modem Emulation via USB – USB connection can be used to communicate with the Telit Module directly and execute AT commands.

Power Saving and Accelerometer:

In the basic functionality of the device, if it stays stationary for more than 10 minutes, then it shuts down the GSM module and the GPS module. To wake it up again, just shake it for a couple of seconds and it will turn the GSM and GPS back on. If you prefer to change this, Round Solutions can provide you with the source code of the microcontroller. Please check section: **Microcontroller Programming**

USB Driver:

The MicroTracker uses FT230XS chip from FTDI chip. The VCP driver can be downloaded from the following link: http://www.ftdichip.com/Drivers/VCP.htm

RSTerm – Terminal Software:

We recommend using the RSTerm terminal software available on our website to download scripts to the MicroTracker. The RSterm can be downloaded from:

http://www.roundsolutions.com/media/pdf/PCB-NT-GE866_rsterm.zip

Select **Python** view from the Top menu and use the below buttons to list, write, read, select, and set as main Python scripts to and on the MicroTracker.

AT#LSCRIPT: Lists the scripts inside the module

AT#WSCRIPT: Opens a file browser and let you choose the file you want to write to the MicroTracker

AT#ESCRIPT=”xxx.pyc” : set the selected script as the main executing script

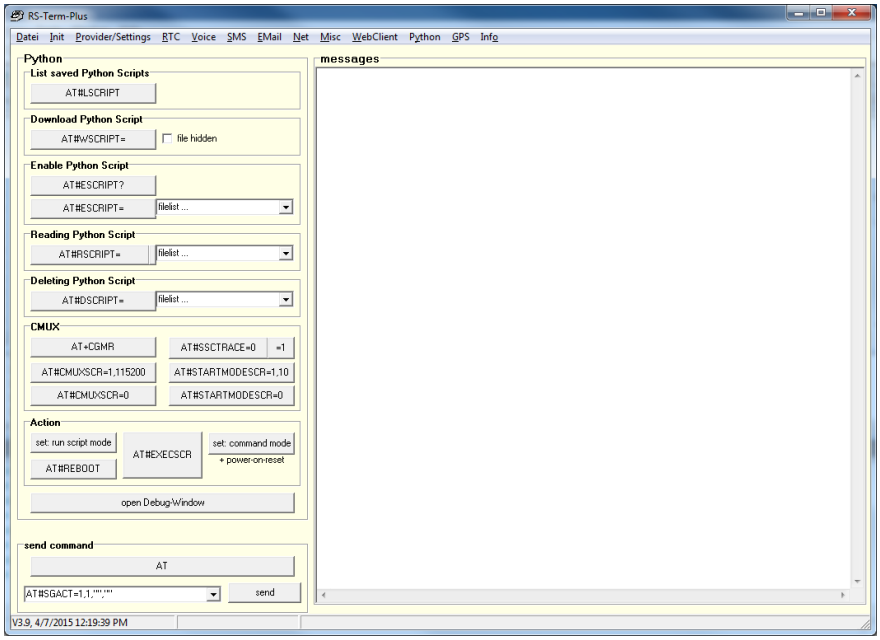
AT#DSCRIPT=”xxx.pyc” : deleted the selected script

AT#EXECSCR: Executes the script selected as the main executing script

AT#STARTMODESCR=1,10 : Executes the script after each boot of the module within 10 seconds if no AT commands are sent.

AT#STARTMODESCR=0: Disables the previously mentioned feature.

Here’s how the **Python** view looks like



Data

Transmission:

We provide a sample tracking application developed in Python for the NanoTracker. You need to download four script files called “ntc.pyc”, “EE_RS.pyo”, “GPS_RS.pyo”, and “MODEM_RS.pyo” to the device using RSterm. After that you need to set “ntc.pyc” as the executing script. Steps to download the scripts are shown at the end of this manual. In this application, data is sent to one of the two Cloud Platforms Telit DeviceWise and Cumulocity using HTTP posts. The server address and the page address are specified in the configuration parameters of the device.

Nanotracker with 3G Modules:

For Nanotrackers with 3G modules (UE866), the user should send the following command AT#PORTCFG=3 and then restart the module. This needs to be done only once to configure the ports to be used properly by the Python script.

Server Side Listener:

A web server can be used as a listener if you plan to host your own server application. A page with server side code receives the HTTP post and parses the messages received from the device. Please contact Round Solutions for the source code. You would need to make changes to parse the data and save it in a database.

API for Python Libraries:

We provide an API that the developers can use to develop their own Python scripts for the device. The API provides easy access to GPS data, easy module configuration, easy connection to server, and read write to the on board EEPROM. The API has three classes: GPS_RS, Modem_RS, and EE_RS

GPS_RS:

Here’s a code snippet that demonstrates how it works:

```
import GPS_RS

m_gpsmanager = GPS_RS.Gps_RS()

m_gpsmanager. initGPS ()   #sets port speed

GPSDt = m_gpsmanager.GetGPSData()  #returns a string of raw GPS data

m_gpsmanager.UpdateGPSParameters()  #reads NMEA messages from the GPS and updates the coordinates of the device

GPSHasFix = m_gpsmanager.GPSHasFix()     #returns 1 if yes and 0 if not

Lastlongitude = m_gpsmanager.GetLongtitude()  # string - format is degrees, decimal minutes

Lastlatitude = m_gpsmanager.GetLatitude()   # string - format is degrees, decimal minutes
```

```
LastDateTime = m_gpsmanager.GetDateTime()  # string - format is yyyy.MM.dd hh:mm:ss

LastSpeed = m_gpsmanager.GetSpeed()  #string - value is in Knots unit

LastHeading = m_gpsmanager.GetHeading()  #string - value is in degrees
```

Modem_RS:

Here's a code snippet that shows how the Modem_RS class works:

```
import MODEM_RS

m_modemmanager = MODEM_RS. Modem_RS ()

 m_modemmanager.ExecuteATCommand("Sample AT Command",TimeOut)  #executes the commands and returns the response

 m_modemmanager.SetAPN("APN Of Operator") # returns 1 is successful and 0 if not

 ret = m_modemmanager.ActivateContext("Username","password")  #returns 1 if context if activated, 0 otherwise

 m_modemmanager.DeactivateContext()

 ret = m_modemmanager.IsContextActive()  #returns 1 of context is active, 0 otherwise

 ret  = m_modemmanager.GetDCD()  #returns 1 if DCD is high

 ret = m_modemmanager.ConnectToServer("Server_Address or IP",TCP_Port)

 m_modemmanager.SendData("sample data to send when connection to the server is established")

 m_modemmanager.DisconnectServer()  #closes the connection to the server

 m_modemmanager.InitSMSSettings()  #call once at the beginning of the script

 m_modemmanager.SendSMS("Number","SMS Contents")
```

EE_RS:

Here's a code snippet that demonstrates how it works:

```
import EE_RS

   m_ee = EE_RS.Ee_RS()

   m_ee.InitEE()

   m_ee.WriteByte(65,0)  #Write Byte 65 at location 0

   m_ee.WriteByte(66,1)  #Write Byte 66 at location 1

   m_ee.WriteByte(ord('C'),2)  #Write Byte 'C' at location 2

   SER.send('Reading Byte at location 0 = '+m_ee.ReadByte(0)+'---\r\n')

   SER.send('Reading Byte at location 1 = '+m_ee.ReadByte(1)+'---\r\n')
```

```
SER.send('Reading Byte at location 2 = '+m_ee.ReadByte(2)+'---\r\n')

StrInMem = m_ee.ReadString(0,3)  #Read a string of 3 characters starting at location 0
```

The address of the memory has a range from 0 to 131072 (128KBytes)

SMS Configuration Example:

To change a variable value by sending a SMS to the device, the following code could be used:

```
#assume that you send a SMS to the device with the following text:  interval:30,
#the comma at the end is necessary to determine the end of the message. Any other character can also be used
def CheckSMS():
   SMSBuffer = m_modemmanager.ExecuteATCommand('AT+CMGL=ALL\r', 20)
   headindex = SMSBuffer.lower().find('interval:')
   if(headindex!=-1):
      tailindex = SMSBuffer.find(',',headindex)
      NewInterval = SMSBuffer[headindex+9:tailindex]
      m_modemmanager.ExecuteATCommand('AT+CMGD=1,4\r', 5)   #deletes the SMS in the modukle
      m_modemmanager.SendSMS('+49123456789',"Interval is updated ") #send back a confirmation SMS
```

Note that it is necessary to save the new value of the variable in a file and read it at next startup. This is done through standard code in Python for file Read/Write.

Connect a TCP socket Example Code

```
import MODEM_RS
import MDM
import MOD
import time
GPRS_USER = ""
GPRS_PASSW = ""
M2MAPN = "surfo2"
m_modemmanager = MODEM_RS.Modem_RS()
m_modemmanager.SetAPN(APN)
if(m_modemmanager.ActivateContext(GPRS_USER,GPRS_PASSW)==1):
        if(m_modemmanager.ConnectToServer("Server_Address",80)==1):
                #send some data to the server
                Pass
```

Building HTTP Post example:

```
 POSTING_PAGE = "somefolder/somepage.aspx"
GPS_SERVER = "www.someserver.com"
 BufferedData = "payload of http post - Data"
 PostHeader = "POST /"+POSTING_PAGE + " HTTP/1.0"
 PostHeader  = PostHeader +  "\r\n"
PostHeader  = PostHeader +  "Content-Type: application/x-www-form-urlencoded"
 PostHeader  = PostHeader +  "\r\n"
PostHeader  = PostHeader +  "Host:"
PostHeader  = PostHeader +  GPS_SERVER
 PostHeader  = PostHeader +  "\r\n"
PostHeader  = PostHeader +  "Content-Length:"
PostHeader  = PostHeader +  str(len(BufferedData))
```

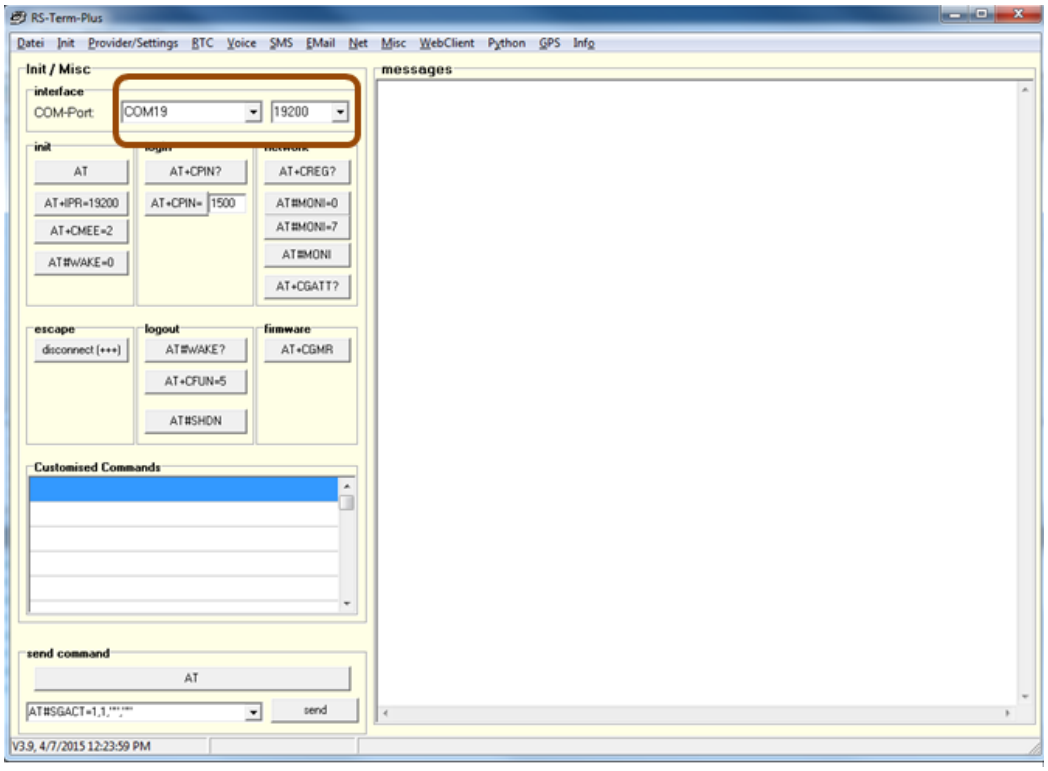
```
 PostHeader  = PostHeader +  "\r\n"
PostHeader  = PostHeader +  "Expect: 100-continue"
PostHeader  = PostHeader +  "\r\n"
 PostHeader  = PostHeader +  "Connection: Close"
 PostHeader  = PostHeader +  "\r\n"
PostHeader  = PostHeader +  "\r\n"
PostHeader  = PostHeader +  BufferedData
```

Device Main Script:

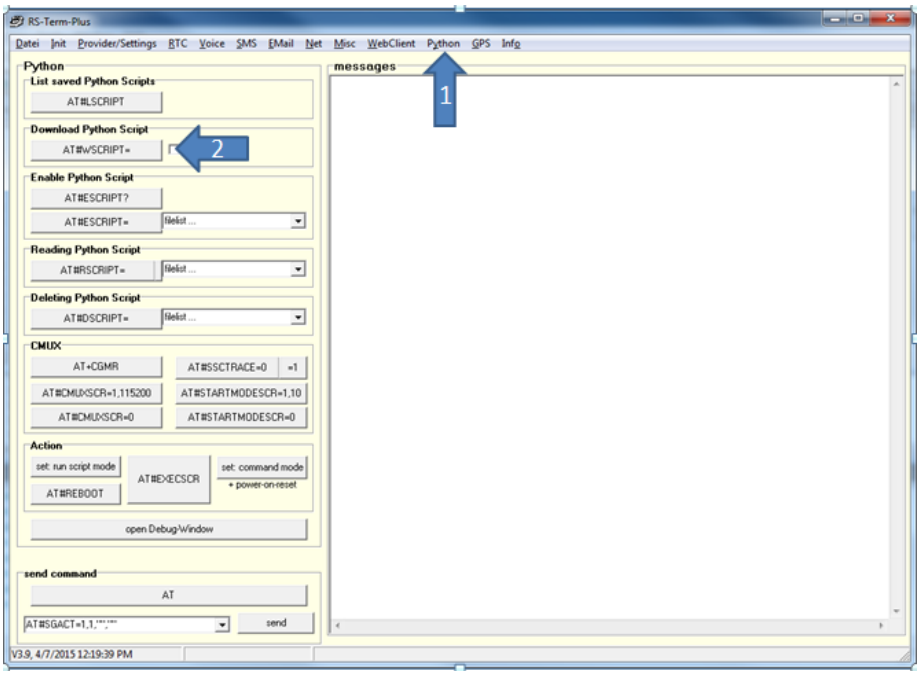
Unless otherwise mentioned, the device doesn't come with a script preloaded. Please contact Round Solutions to get either the Libraries and or the main complied script to download to the device.

Before proceeding, make sure you have successfully installed the USB drivers.

Upon opening RSTerm, make sure you select the proper COM port and set the baud rate as shown below. Click on the button AT and wait for the tracker to reply by OK to make sure you have established communications with the module.



After that click on **Python** on the Top menu, then click AT#WSCRIPT. Note that you cannot overwrite a script with the same name that already exists. You need to delete it first from the module and then download the new one.



Steps to download a new Python script:

Downloading a new script using RSTerm terminal software is very easy. Just follow the following steps after you have successfully connected the device:

1- Go to Python View by clicking on **Python** on the top men
2- Click on **AT#LSCRIPT** to list the scripts downloaded to the device
3- If the script name already exists, make you sure you delete it using **AT#DSCRIPT="scriptname.pyc"**
4- Click on **AT#WSCRIPT** and navigate to the script file you wish to download
5- Once finished, click on **AT#LSCRIPT** again to update the populated lists with the name of the new script
6- Type in the script name in the field next to the button **AT#ESCRIPT=** and then click on the button
7- Click on the button **AT#STARTMODESCR=1,10** to enable running the script upon module boot after 10 seconds
8- Click on **AT#EXECSCR** to execute the script immediately

Microcontroller Programming:

If you wish to change the power saving functionality of the device, Round Solutions can provide you with the source code of the Microcontroller on board. The Nano tracker has a PIC18LF46K22 microcontroller on board. The Microcontroller can turn on and off the Cellular module and the GPS module separately. It communicates with the Accelerometer via Master I2C bus. It can also communicate with the Cellular module via Slave I2C bus.
You need an ICD3 and a connector cable from Tag-Connect:
http://www.tag-connect.com/TC2030-MCP-NL
This connector cable can be purchased from several online components distributors. The list of distributors can be found here:

http://www.tag-connect.com/tag-connect-distributors

The development Environment is MPLABX IDE and XC8 compilers. Both can be downloaded free of charge from Microchip's website: www.microchip.com

The microcontroller controls two on board active switches that powers ON/OFF the Cellular and GPS modules. These switches are controlled by these functions:

Cellular Module:
void TurnXE866OFF()
void TurnXE866ON()

GPS Module:
void TurnGPSOFF()
void TurnGPSON()

The device configures the accelerometer to generate an interrupt when inactivity is detected. The configuration of it is done in the function:

void ConfigureAccelerometer()

Please refer to the datasheet of the accelerometer LIS2HH12TR from STMicroelectronics for more information on this.

The function **bool** CheckIfIdle() returns true if the device has been idle for more than a certain period **IdleTotalTime** in seconds which its value can be changed in the definitions.
The implementation of all the previous mentioned functions can be found in the file user.c

The code activates two timers in the microcontroller TMR0 and TMR4.

TMR4 generates an interrupt every 0.1 seconds and has a call back function that is called every 1 second. This is used for timing purposes.
The function implementation of **void** TMR4_ISR() and of **void** TMR4_CallBack() can be found in file tmr4.c

The call back function toggles on of the LEDs every one second and does some time calculations for the function CheckIfIdle()

TMR0 generates an interrupt every 1 second and has a call back function that is called every 1 second also. This is used for timing purposes and is free for the developer to use.

The Cellular module can communicate with the microcontroller using I2C. The implementation for this is done in file I2C2.c

The function of interest in this file is:
void I2C2_StatusCallback(**I2C2_SLAVE_DRIVER_STATUS** i2c_bus_state)

If the Cellular module writes data to the microcontroller using the I2C bus, the following code is executed:

```
case I2C2_SLAVE_WRITE_COMPLETED:
        ....
        ....
        switch (RegisterAddress)
        {
                case 0x01: LATBbits.LATB2 = I2C2_slaveWriteData; break; //LED 1 address
                case 0x02: LATBbits.LATB1 = I2C2_slaveWriteData;break; //LED 2 address
                case 0x03: break;
        }
```

In the above example, we assigned address 0x01 for LED1 and address 0x02 for LED2. The Cellular module can control these LEDS by writing data to these registers using I2C bus.

If the cellular module requests to read data from the microcontroller, this is the code section that is executed:

```
case I2C2_SLAVE_READ_REQUEST:
  switch (RegisterAddress)
  {
        case 0x01: SSP2BUF = LATBbits.LATB2; break; //return state of LED 1
        case 0x02: SSP2BUF = LATBbits.LATB1;break; //return state of LED2
        case 0x03: break;
        default: SSP2BUF = 0x99;//return a 0x99 to indicate that no valid register address is found
  }
  RegisterAddress++;//to allow reading the next register
```

Same applies here for registers addresses. You can add additional registers to send like accelerometer data, sleep time, etc….

On the cellular module side, you can use the following code to write and read data from the microcontroller using the I2C bus:

```
import IIC

I2C_SDA = 5
I2C_SCL = 6
I2C_ADDR = 0x08
bus = IIC.new(I2C_SDA, I2C_SCL, I2C_ADDR)
status = bus.init()

#Write 0x00 to register with address 0x01
res = bus.readwrite('\x01\x00', 0)

#Read value from register 0x01
res = bus.readwrite('\x01', 0)
```