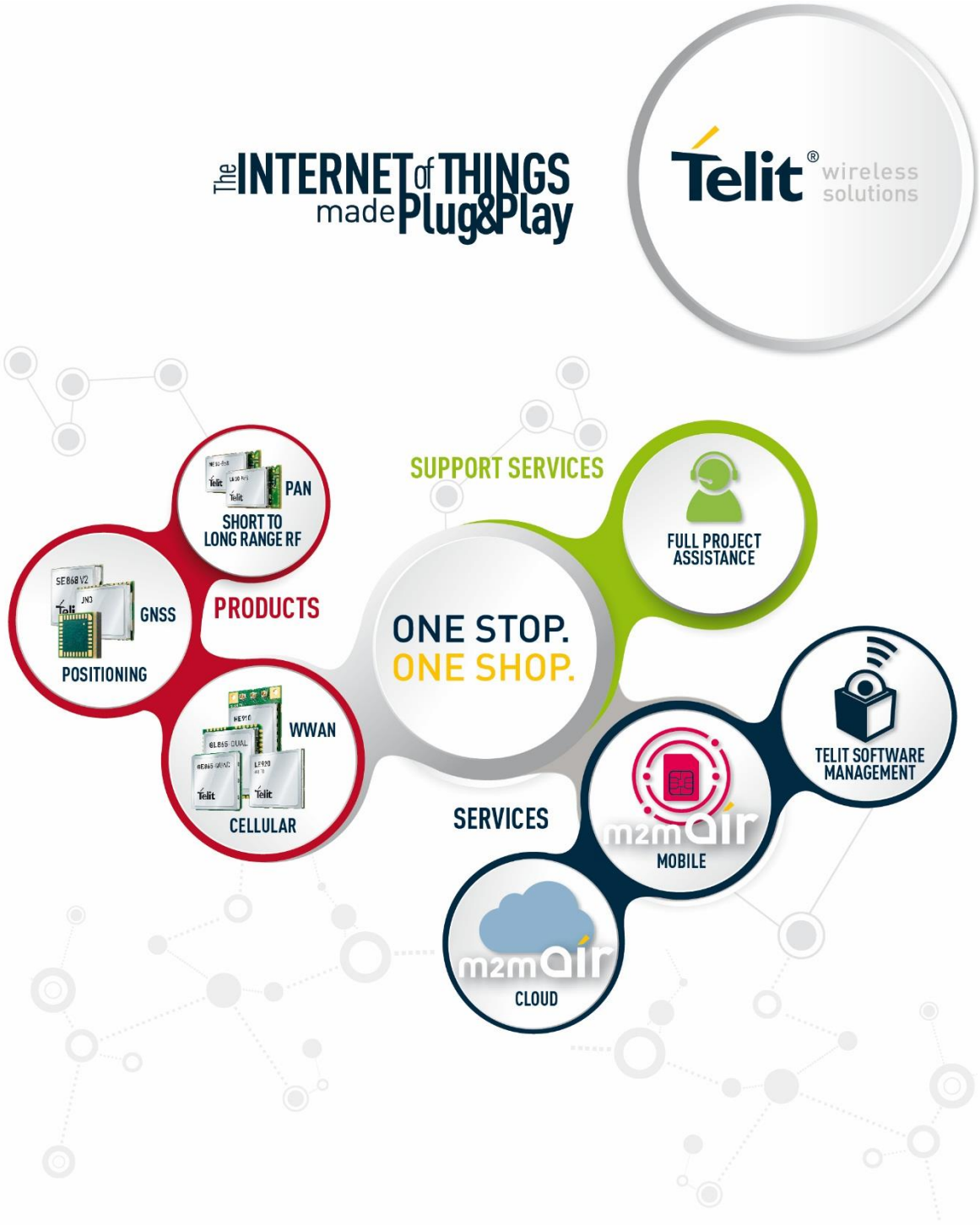


APPZONE PYTHON USER GUIDE

APPLICABILITY TABLE

PRODUCTS

		GE910 SERIES
		UE910 SERIES
		HE910 SERIES
		UL865 SERIES
		UE866 SERIES

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE

LEGAL NOTICE

These Guidelines are general guidelines pertaining to the installation and use of Telit's Integrated Application Development Environment ("IDE"). Telit and its agents, licensors and affiliated companies make no representation as to the suitability of these Guidelines for your particular needs and Telit disclaims any and all warranties, expressed or implied, relating to the contents of this document. Furthermore, this document shall not be construed as providing any representation or warranty with respect to any Telit Product whether referenced herein or not.

It is possible that this document may contain references to, or information about Telit products, services and programs, that are not available in your region. Such references or information shall not be construed to mean that Telit intends to make available such products, services and programs in your area.

HIGH RISK USES

The IDE is not intended for the design of software for use in hazardous environments or environments requiring fail-safe controls, including the operation of nuclear facilities, aircraft navigation or aircraft communication systems, air traffic control, life support, or weapons systems ("High Risk Activities"). Without derogation, Telit, its licensors and its supplier(s) specifically disclaim any expressed or implied warranties with respect to the use of the IDE for development of software for such High Risk Activities and specifically disclaim all liability with respect to such use.

TRADEMARKS

The names Telit, device WISE, device WISE by Telit, secure WISE, secure WISE by Telit, Telit IoT MODULES, Telit IoT PORTAL, Telit IoT PLATFORM, Telit IoT PLATFORMS, Telit IoT CONNECTIVITY, Telit IoT SERVICES and their associated logos are trademarks of Telit Communications PLC, its subsidiaries or affiliates in the United States and/or other countries. Other company or product names are the trademarks of their respective owners. All trademark rights are reserved.

THIRD PARTY RIGHTS

The IDE may contain or may be provided in conjunction with third party software (the "third party software"), including some or all of those detailed in the NOTICES file provided to you during installation of the IDE. You acknowledge that not all third party software detailed in the NOTICES file are necessarily used or provided in the particular version of the IDE you install and use. Refer to the IDE's EULA and the NOTICES file for licensing information pertaining to the third party software.

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

1.1. About This Guide

The document describes the installation and use of AppZone Python. Using AppZone Python you can create, build, and run scripts on a Telit module. You can also create and manage Python scripts, adopting the same style of user interface used for AppZone Python projects.

The User Guide covers different type of Modules listed in the Applicability Table and, in more details, in Modules & SW Ver. Tables.

1.2. Scope

This document provides the guidelines to install and use AppZone Python to create an M2M application and run the application on a Telit module. It describes examples using the +M2M AT commands to configure the module and manage the M2M applications uploaded into the module.

1.3. Audience

This User Guide is intended for users who need to develop a custom application (M2M application or Python script), and run it on a Telit module as an embedded application that uses the services provided by the module itself.

1.4. Contact Information, Support

For general contact, technical support services, technical questions and report documentation errors contact Telit Technical Support at:

TS-EMEA@telit.com

TS-AMERICAS@telit.com

TS-APAC@telit.com

Alternatively, use:

http://www.telit.com/support

For detailed information about where you can buy the Telit modules or for recommendations on accessories and components visit:

http://www.telit.com

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

Telit appreciates feedback from the users of our information.

1.5. Text Conventions

Caution or Warning – Alerts the user to important points about integrating the module, if these points are not followed, the module and end user equipment may fail or malfunction.

Tip or Information – Provides advice and suggestions that may be useful when integrating the module.

All dates are in ISO 8601 format, i.e. YYYY-MM-DD.

1.6. Acronyms & Abbreviations

ADE	Application Development Environment
API	Application Programming Interface
APN	Access Point Name
DTE	Data Terminal Equipment
GCC	GNU Compiler Collection
GPIO	General Purpose Input/Output
GUI	Graphic User Interface
I2C	Inter-Integrated Circuit
NVM	Non-Volatile Memory
PDP	Packet Data Protocol
SDK	Software Development Kit
SMS	Short Message Service
SPI	Serial Peripheral Interface
SSL	Secure Socket Layer
UART	Universal Asynchronous Receiver Transmitter

1.7. Related Documents

[1] UE910 Hardware User Guide, 1vv0301012

[2] Easy Scripts in Python 2.7 80378ST10106A

[3] HE910 Hardware User Guide, 1vv03700925

[4] GE910 Hardware User Guide, 1vv0300962

[5] File System Tool User Guide, 1vv0301192

[6] Telit 3G Modules AT Commands Reference Guide, 80378ST10091A

[7] AT Commands Reference Guide, 80000ST10025A

[8] Running AT Commands Remotely, 80000NT10029A

[9] Easy Script in Python 2.7, 80378ST10106A

[10] Telit website Python's scripts, 1vv0300808

[11] Telit 3G Modules Ports Arrangements, 1vv0300971

[12] GE910 Series Ports Arrangements, 1vv0301049

[13] UL865 Hardware User Guide, 1vv0301050

[14] UE866 Hardware User Guide, 1vv0301157

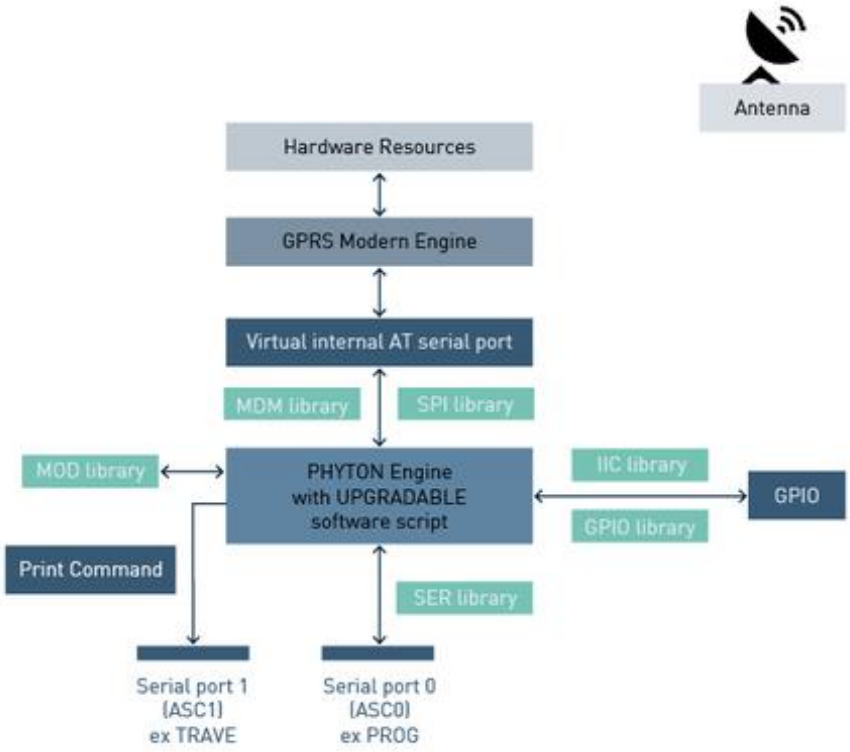
[15] AppZone Python API Reference Guide 80496ST10726A

In the next pages is used the notation [x]/[y] to refer to documents of different modules. You have to see the document in accordance with the module that you are using.

2. APPZONE OVERVIEW

AppZone Python enables programing on the Telit modules, writing the controlling application directly in a high-level open-sourced language: Python. Development in Python is performed using Easy Script®, which is a complete software platform that enables development of m2m applications without additional investment in hardware, license fees, and dependence on other manufacturers‘ technology.

- Built-in interfaces are dedicated to the communication between the Python script and the Telit module.
- These interfaces provide the following functionalities:
- Sending and receiving of the data from the network
- Reading and writing information from the serial port in the case of GPS or other additional devices connected
- Faster handling of general purpose I/O and direct control of pins
- Selecting appropriate timer for customer application



AppZone Python provides the following interfaces:

- MDM – MDM is the most important interface that allows Python script to send and receive data from the network during connections. It is similar to the standard serial port interface of the Telit module. The difference is that this interface is not a real serial port but rather an internal software bridge between Python and the engine handling the internal AT commands.
- MDM2 – The second interface between Python and internal AT command handling, necessary for sending AT commands from a Python script to a mobile device and receiving AT responses when the other interface with this role, standard MDM, is already in use.
- SER – This interface allows a Python script to read and write to the real physical serial port ASC0. When Python© is running, this serial port is available for communication between Python script and an external device, since it's not being used as AT command interface.
- SER2 (non GPS products) – The new interface between Python and internal serial port ASC1 for direct handling. It is used for sending data from a Python script to the serial port ASC1 and

receiving data from the serial port ASC1. This allows the Python application to have access to two serial ports ASC1 and ASC0 (with standard SER interface).

- GPIO – General Purpose Input/Output (GPIO) with Python permits direct control over the pin's MOD. This also serves as an interface between Python and miscellaneous functions of the module such as sleep and counter.
- IIC – A custom software IIC bus that can be mapped over any available GPIO pin.
- SPI – A custom software Serial Protocol Interface bus that can be mapped over any available GPIO pin.

You can develop M2M applications that can address a wide range of different applications, such as remote monitoring and control, security and surveillance, telemetry, location services, billing, and fleet management. The generic user application can access the following modem resources:

- Operating System: Signals, Semaphores, Timers, Dynamic Memory Management, etc.
- HW/SW: GPIO, I2C, UART, SPI, Keypad, File-System, RTC, etc.
- 2G/3G: Communication services.
- Networking: BSD socket support, SSL capabilities.

Refer to documents [1]/[3]/[4]/[13]/[14] for information on module hardware resources.

3. INSTALL APPZONE PYTHON

AppZone Python is an integrated development tool based on Eclipse IDE and running on Windows.

During installation, AppZone Python announces itself as AppZoneIDE.

On the module, AppZone Python and AppZone C services are mutually exclusive. Choose the module type in accordance with your target.

3.1. Hardware and Software Requirements

To install AppZone Python you must:

- Have administrator permissions
- Know which module you will be using

The following table lists the requirements for installing AppZone IDE:

Need to update the table

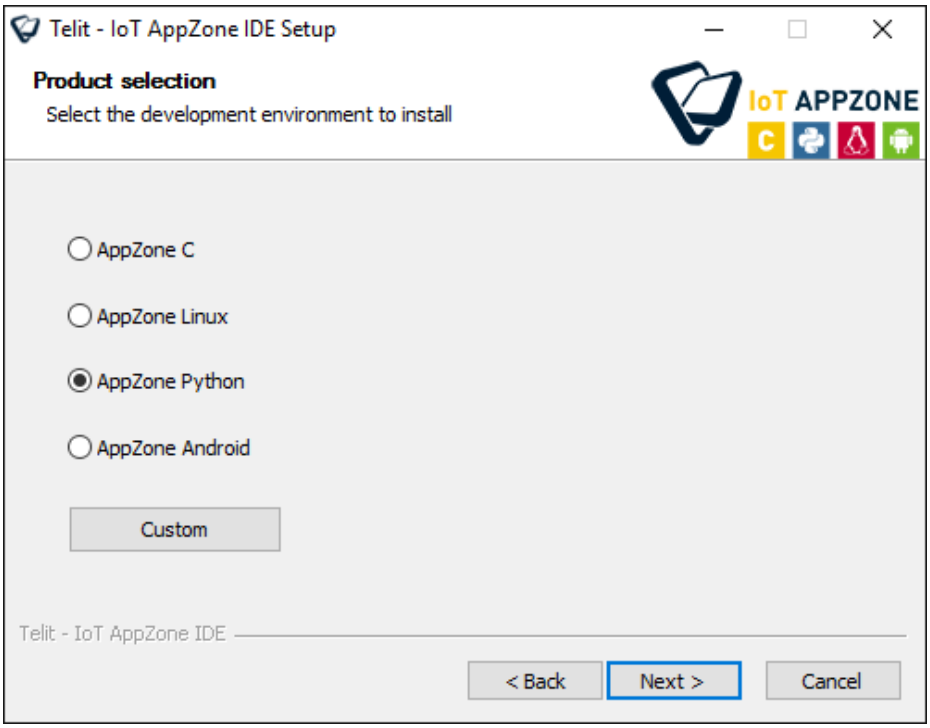
OS	Windows XP, Windows Vista, Windows 7 (32bit or 64bit), Windows 8, Windows 10
RAM	1 Gb
Free disk space	850 MB free disk space
Java SE Runtime	Java™ SE Runtime Environment 32-bit **Note:** 64-bit is not compatible with AppZone IDE.

3.2. Install the IoT AppZone Python IDE

To install the AppZone Python IDE:

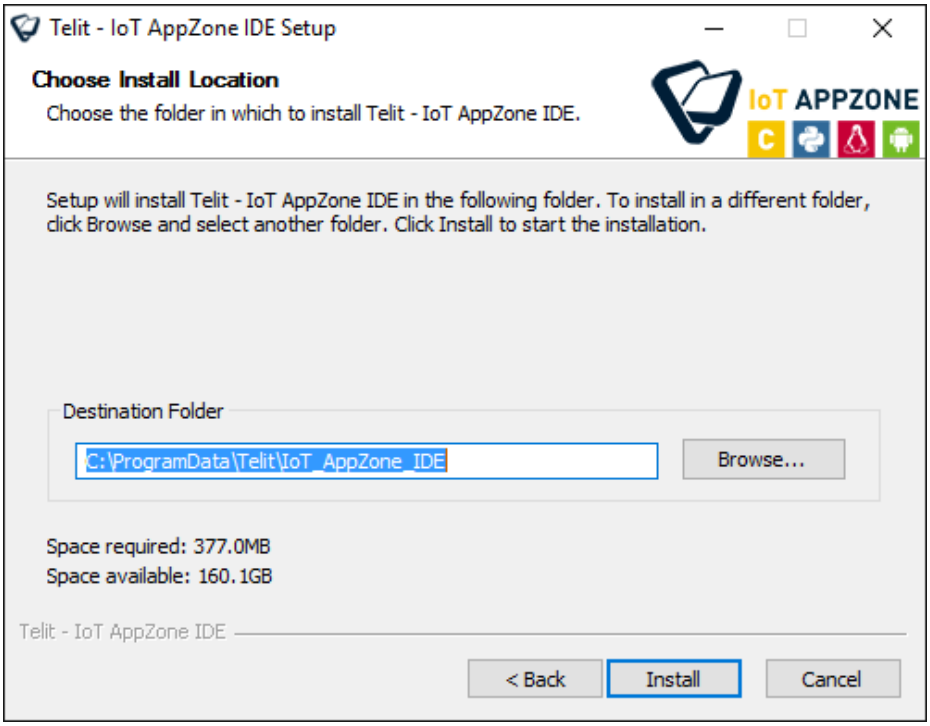
1. Access the Telit portal: http://www.telit.com/support/programming-tools/iotappzone/developers/.
2. Click the AppZone installation file.
3. The installation wizard starts.
4. If Java 32-bit is not yet installed, a message opens notifying you that you must install Java.
   a. Click **OK**. The browser opens with the Java Installation page.
   b. Under **JRE** click **Download**. The JRE download page opens.
   c. Above the table, select **I Agree**.
   d. Click the x86 JRE versions to install.
   e. When the installation ends, click **OK**. The AppZone Python installation windows opens.
5. Click **Next** to continue.
6. Read the license agreement and then click **I Agree** to continue.

7. Select AppZone Python and then click **Next**.



To install more than one development environment, click **Custom** and select the development environments that you want to install.

8. Choose the destination folder in which to install AppZone Python. You can leave the default folder or click **Browse** to select a new destination folder.



9. Click **Install** to begin the installation.

   The installation wizard downloads the files that are required for the development environment that you selected.

10. Click **Finish** to close the AppZone Python installation wizard.

4. GET STARTED

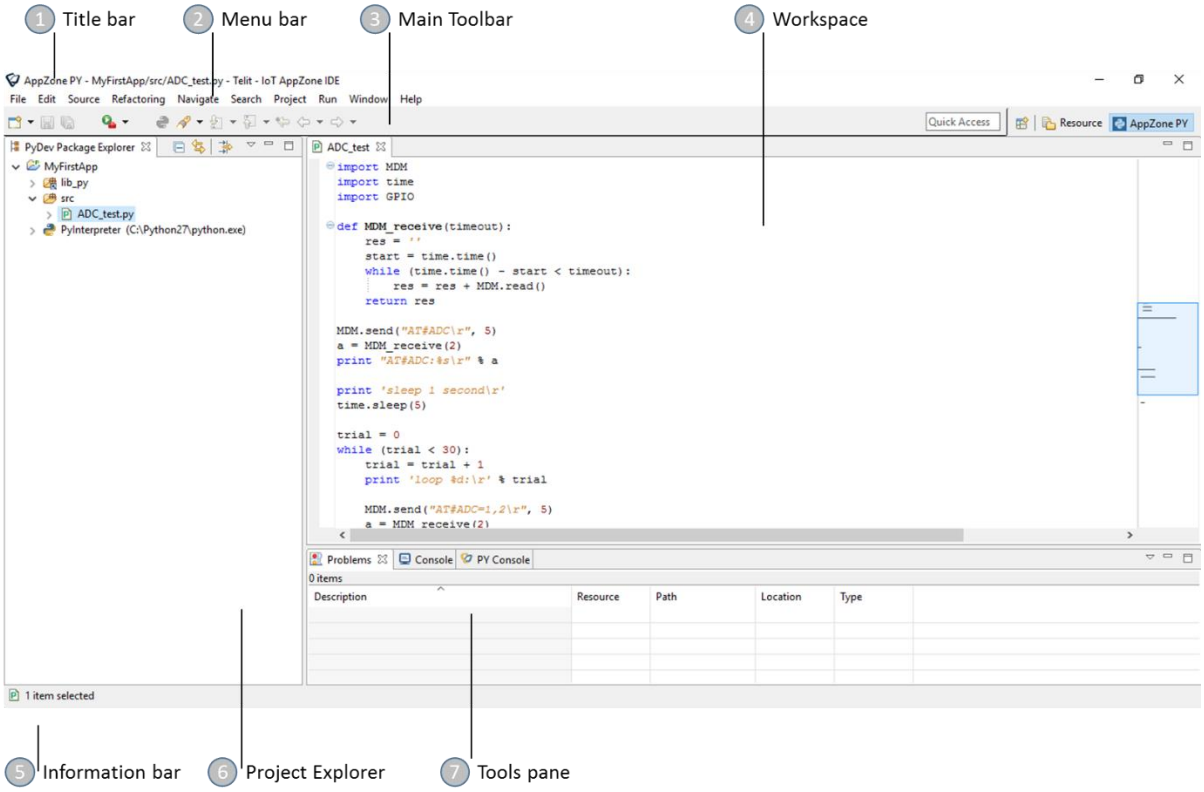
To open the development environment on Windows:

You must have administrator privileges or the development environment will not load.

1. Click **Start > Telit > AppZoneIDE > AppZoneIDE**. The Workspace Launcher window opens.
2. Select the workspace in which you want to work, and then click **OK**.

4.1. Main Window

The following figure describes the main sections of the AppZone Python window:



You can optimize the location of the panes and the views to suit your workflow. You can rearrange the layout of the panes and the views.

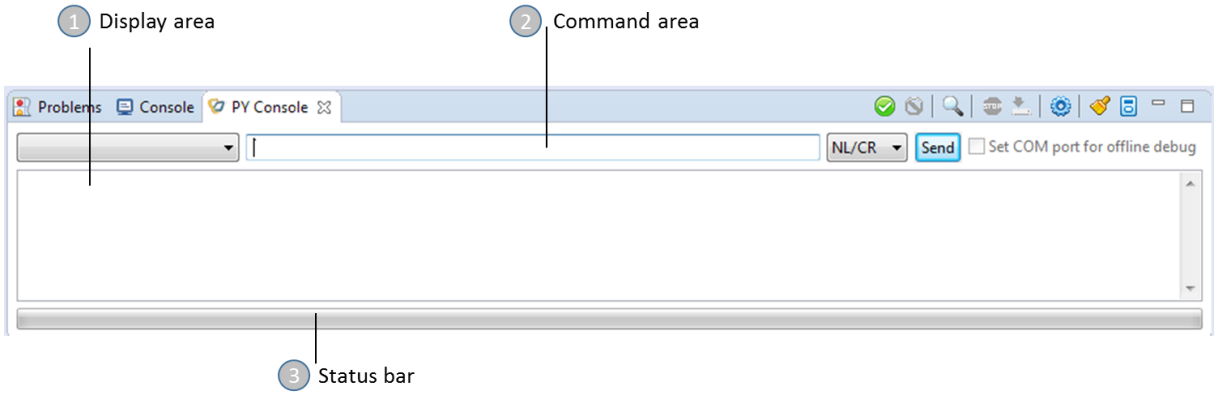
The main window contains the following sections:

1. Titlebar – Displays information on the environment, the current project, and the file that is currently displayed in the workspace in the following format: AppZone Python <*Project name*>/<*File name*>
2. Menu Bar – Provides options that enable you to perform the most common operations on items. Most of these options also appear in the pop-up menus when you right-click items. The menu bar also contains options related to the entire application.
3. Main Toolbar – Provides buttons for navigation and most commonly used operations.
4. Workspace – Displays the details of the file that you select in the Project Explorer pane. In the workspace you define write and edit the code.
5. Information Bar – Displays information on the file that is currently displayed in the workspace in the following format: /<*Project name*>/<*File name*>

6. Project Explorer – Displays all files in the current project.
7. Tools Pane – Displays the following views:
   - Problems – Displays system-generated errors, warnings, or information associated with a resource.
   - Console – Displays a build console that enables viewing the status of the current build.
   - Py Console – Enables exchanging AT Commands and files with the module and displaying the results. For more information, see Py Console.

4.2. Py Console

The Py console enables running Python scripts on the module.



The Py console contains the following areas:

1. Display area – Displays the messages from the module, such as commands results and the connection COM result.
2. Command area – Enables entering commands.
3. Status bar – Displays the progress of download and upload operations.

The following table describes the icons in the Py Console toolbar:

Icon	Name	Description
	Connect COM port	Connects to the COM port that is selected in the Settings.
	Disconnect COM port	Disconnects from the COM port.
	AutoConnect COM Port	Automatically connects to the COM port that is defined in the Settings.
	Abort transfer	Stops transferring the file to the module.
	File Manager	Opens the File Manager, which enables viewing files in the module.
	Settings	Configures the settings that are used to connect to the module.
	Clear Log	Clears the display area.
	Set Ports Simulation	Enables time logging.

5. CREATE APPLICATION

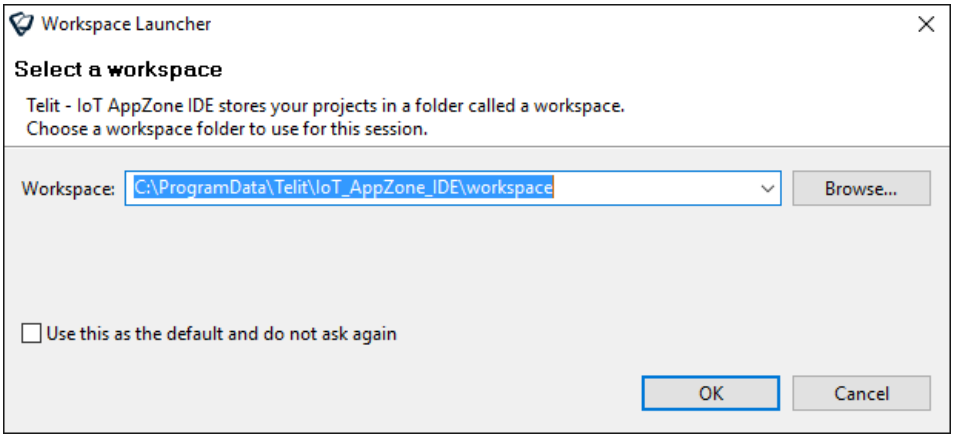
AppZone Python enables managing AppZone applications and provides facilities to write, test (on PC), and upload (to the module) Python scripts. This section describes a guideline to perform the main activities.

5.1. Create an Application

You can create new applications from scratch or use an example application.

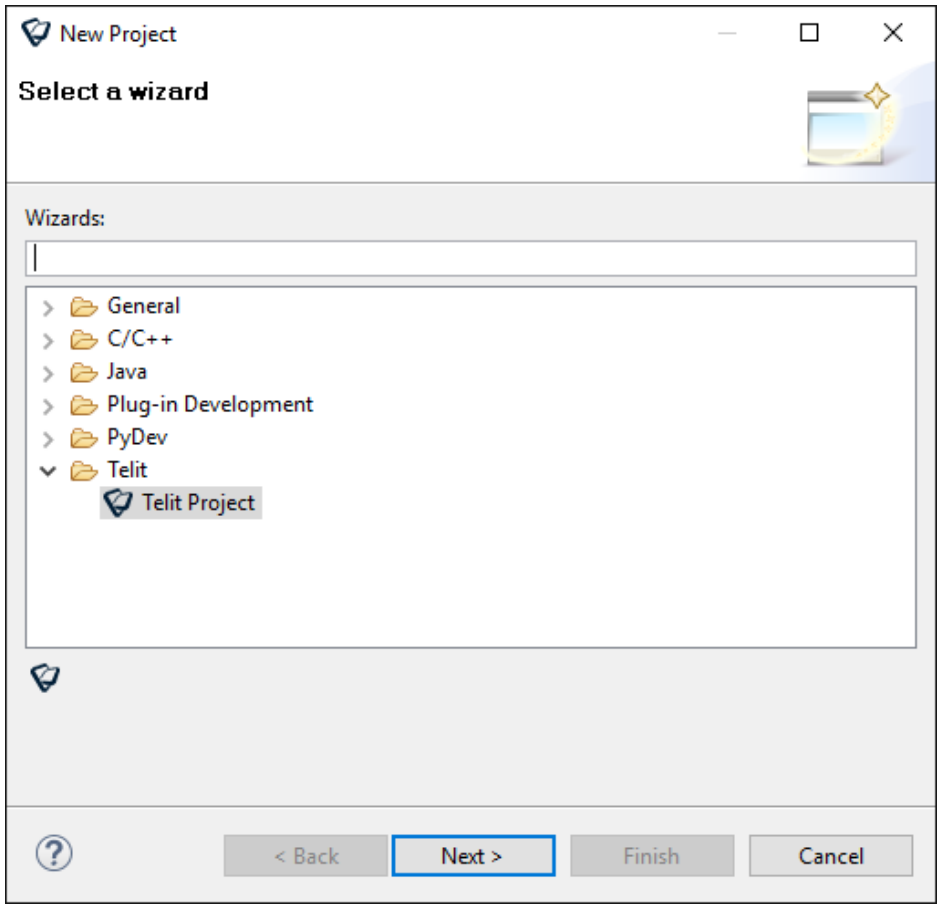
To create a new application:

1. Click **Start > All applications > Telit > AppZoneIDE > AppZoneIDE**. The Workspace Launcher window opens.
2. In the **Workspace** field, type the location of the workspace or click **Browse**.

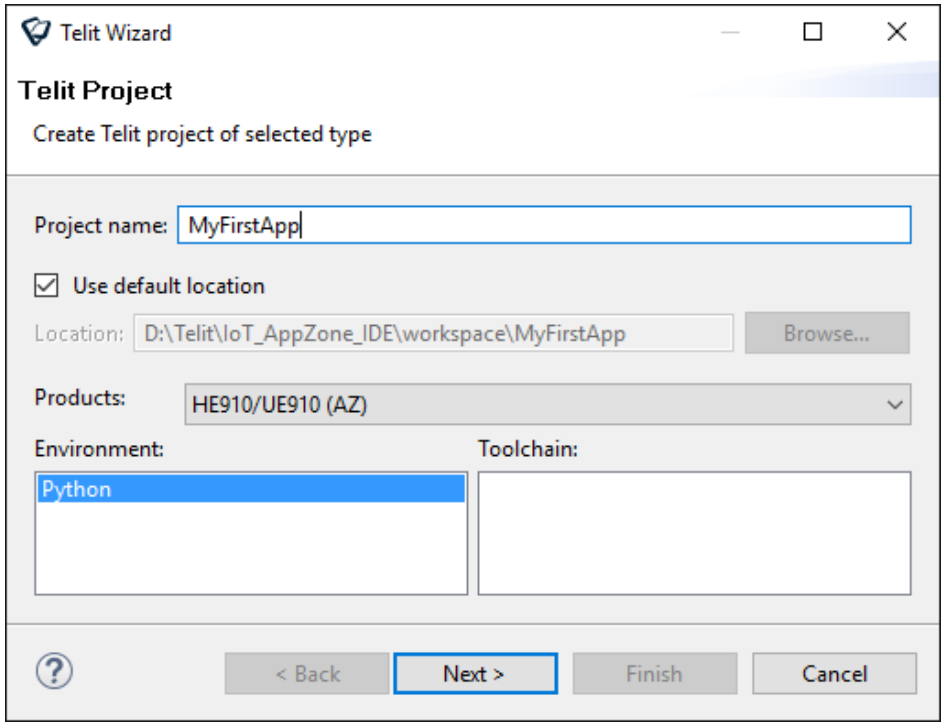


3. Click **OK**. The AppZone Python IDE opens.

4. From the menu select **File > New >** Project. The New Project window opens.



5. To create a new project, expand **Telit** and then select **Telit Project**.
6. Click **Next**. The Telit Project window opens.



a. In the **Project name** field, type a name for the project.

b. To change the default location in which the project is saved:

   i. Clear the **Use default location** checkbox.

ii. In the **Location** field type the new location or click **Browse**.

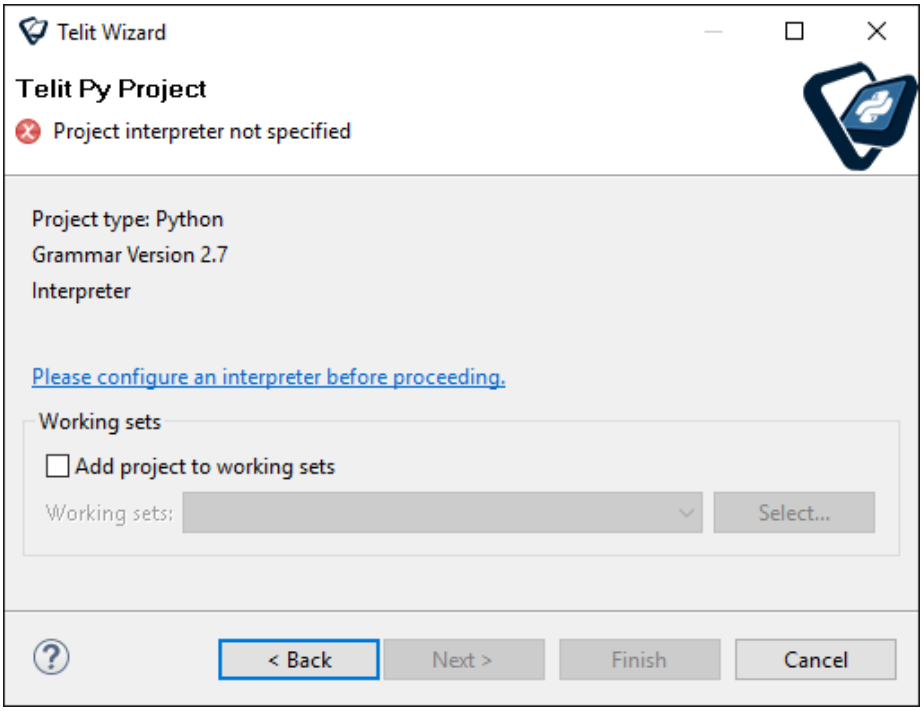
c. In the **Products** field, select the module for which you want to develop the application.

d. In the **Environment** field, select Python.

7. Click **Next**. The Telit Phy Project window opens.

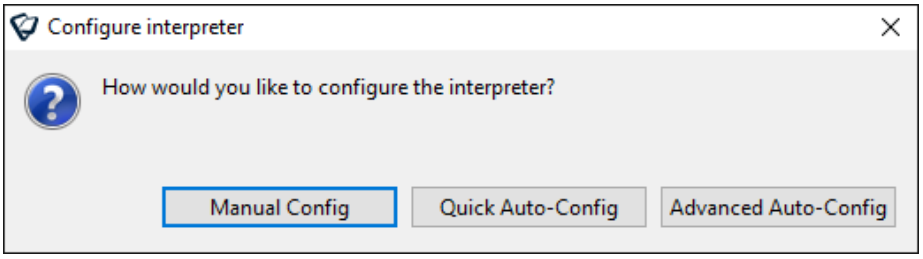
8. If you have not yet configured the Python Interpreter:

a. Click the **Please configure an interpreter before proceeding** link.



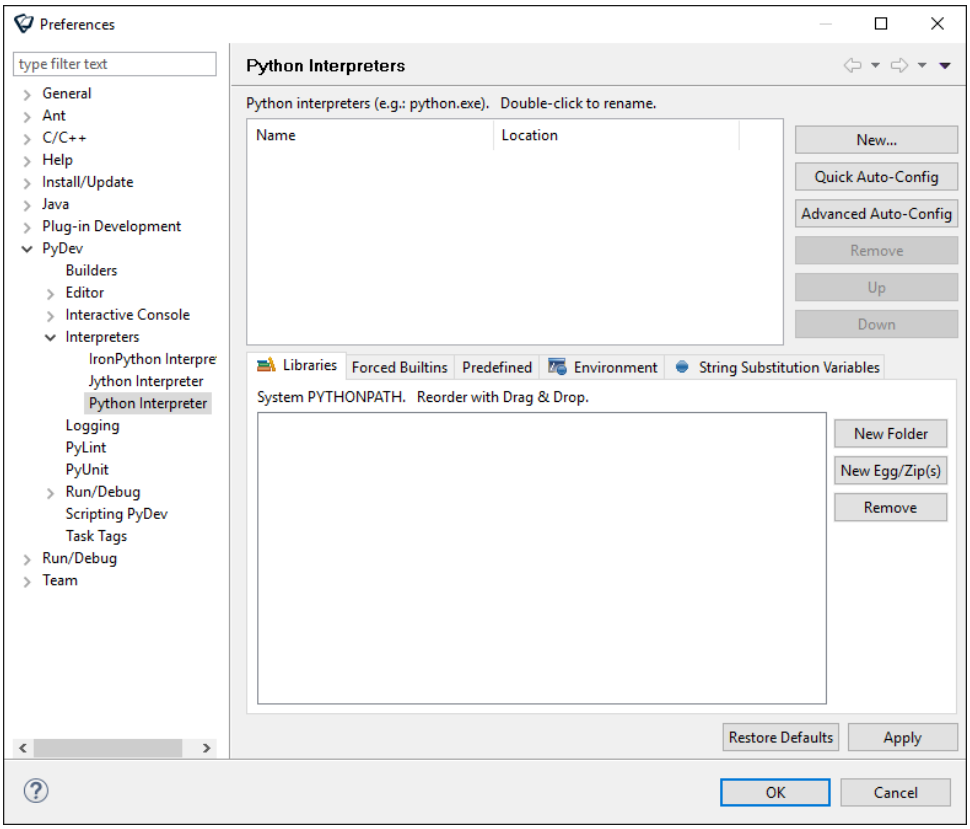
The Configure Interpreter window opens.

b. Click **Manual Config**.



The Preferences window opens with Python Interpreter selected.

c. To define the Python interpreter, click **New**.



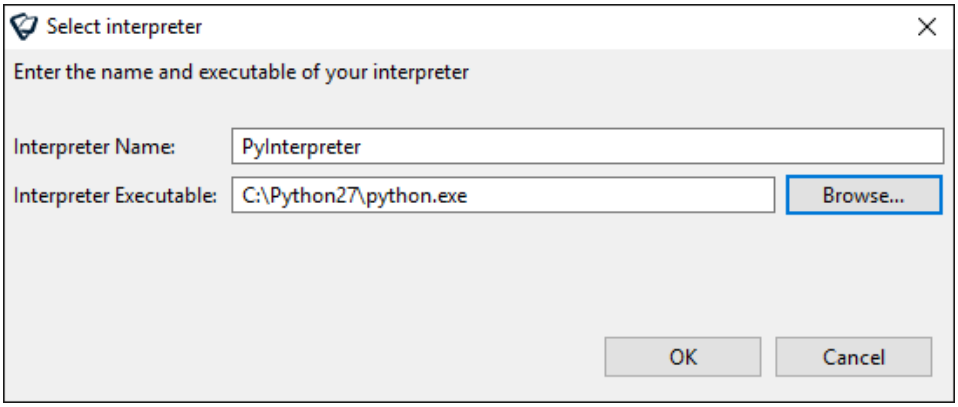
The Select Interpreter window opens.

d. Enter the following information, and then click **OK**:

i. In the Interpreter Name field, type a name for the interpreter.

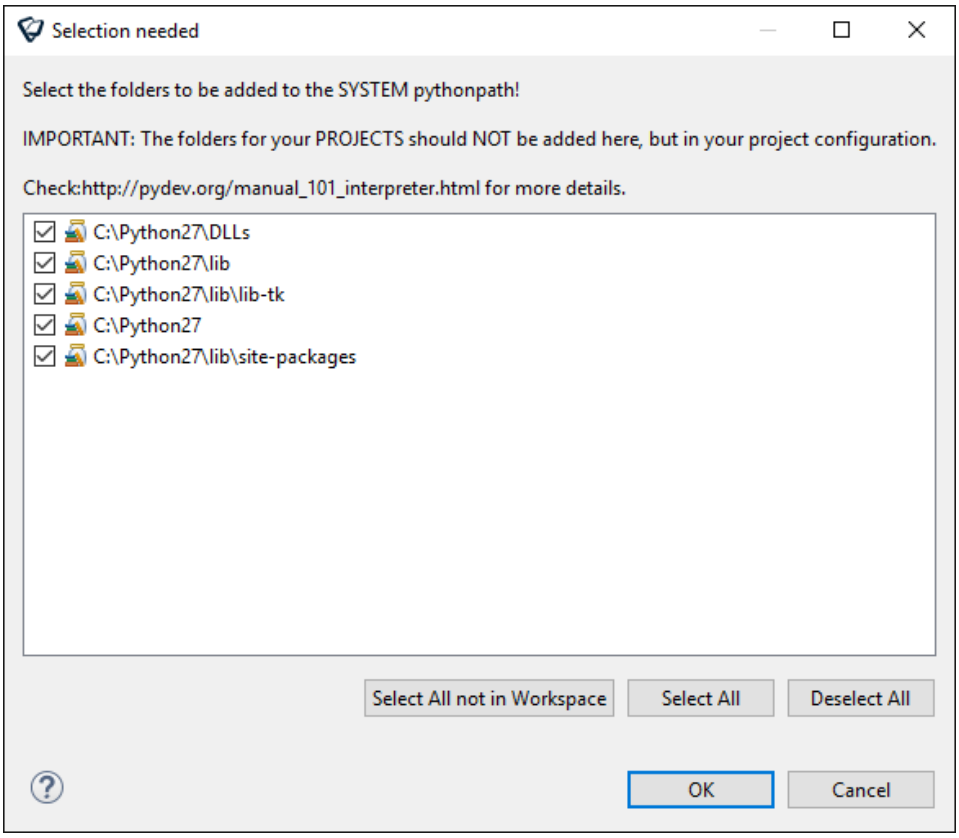
ii. In the Interpreter Executable, click **Browse** and select the complete path to the Python Interpreter.

The default path is: C:\Python27\python.exe.



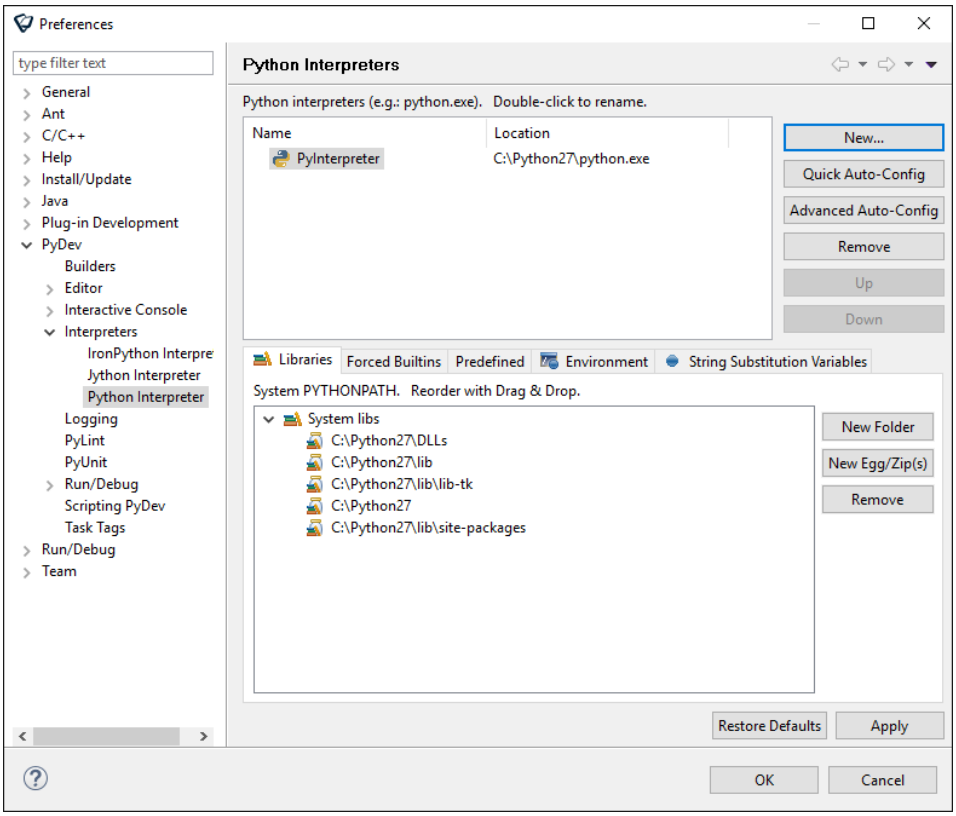
A window enabling to add a selection folder to the Python path opens.

e. Click **OK** to continue.



The new Python interpreter that you defined is added to the interpreters.

f. Click **OK** to close the Python interpreter definition window.



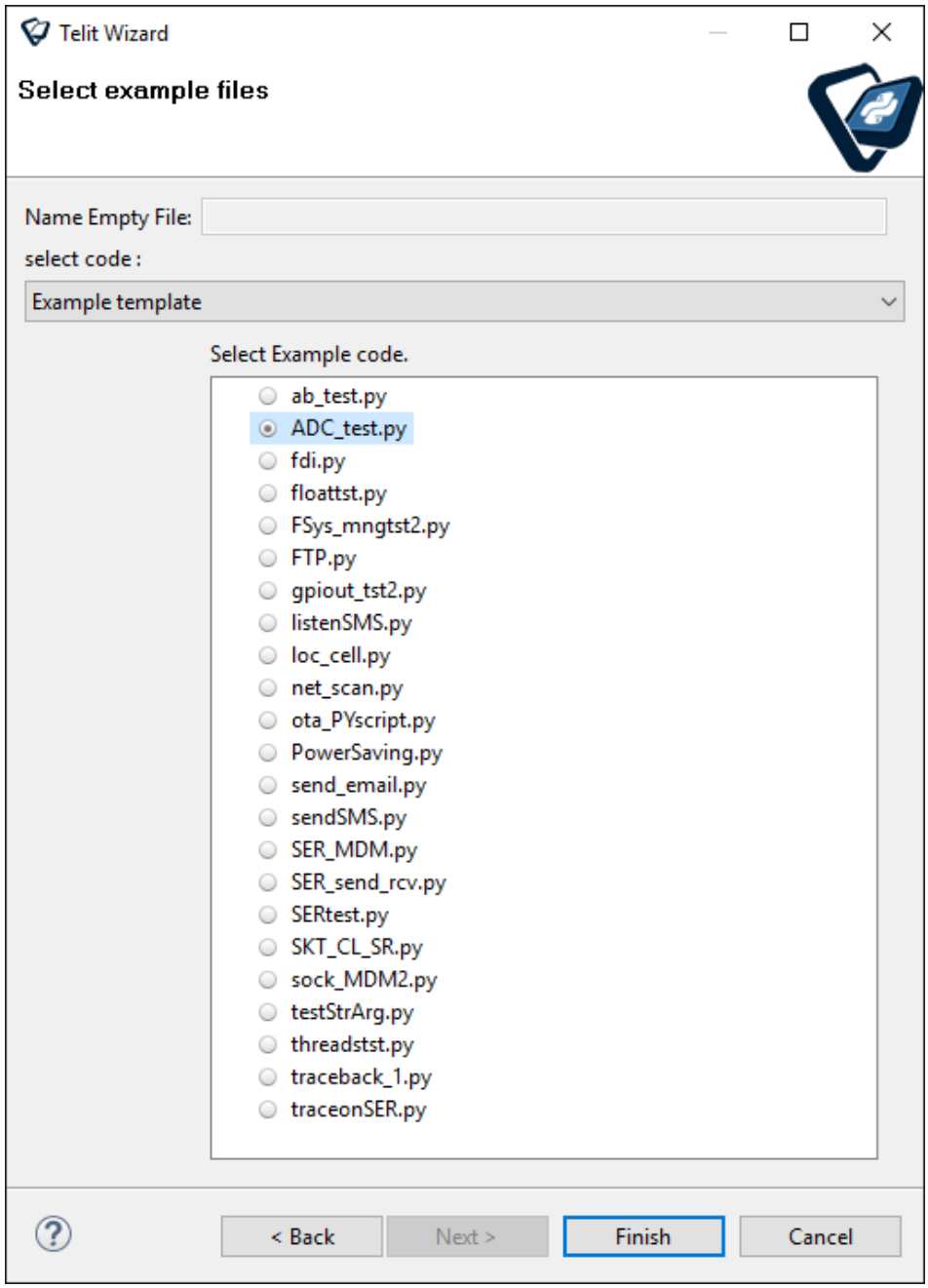
The Python interpreter is defined and the Telit Py Project window opens again.

9. Click **Next**.

   A window enabling to select the project type opens.

10. Select the type of project that you want to create:

    - To create an empty project:
      1. In the New Empty File field, type a name for the project.
      2. From the Select Code list, select **Empty Project**.
    - To create a project based on an example template:
      1. From the Select Code list, select **Example template**.
      2. Select the checkbox next to the code that you want to use as an example. For a description of the sample applications, see Sample Applications.

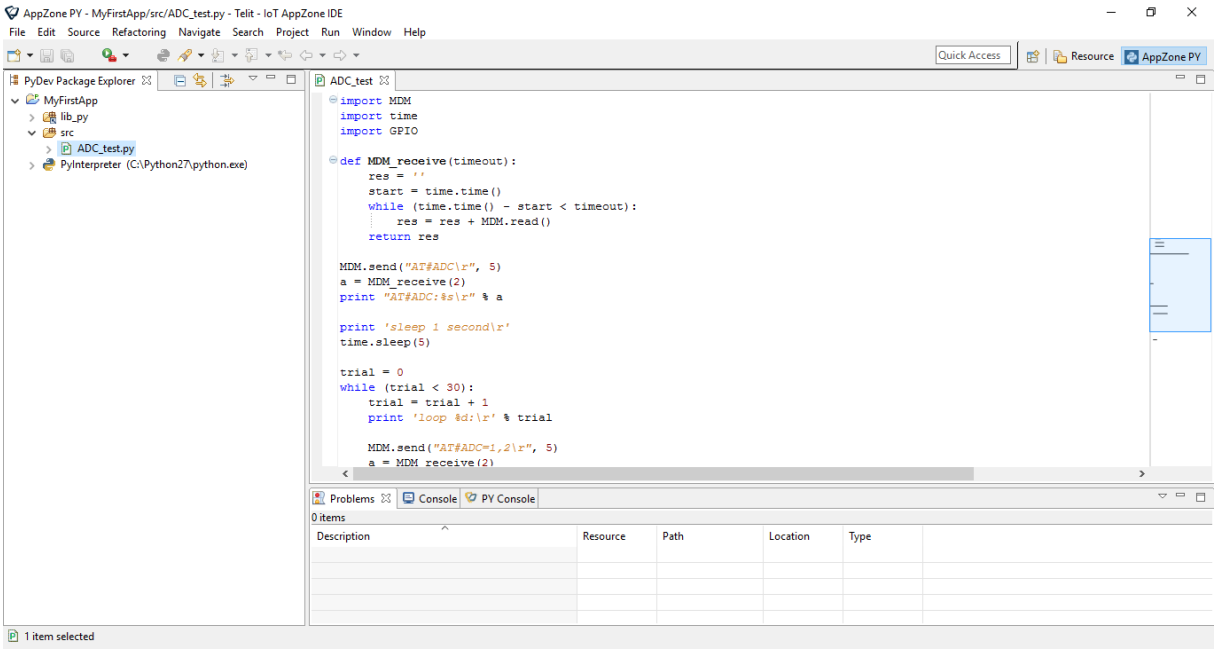


11. Click **Finish**.

12. A Default Eclipse preferences for PyDev window opens. Select your preferences, and then click **OK**.

    The AppZone Python IDE opens.

If you created a project based on example application, AppZone Python contains the source files for that example.



5.2. Sample Applications

Python scripts are text files stored in NVM inside the Telit module. The file system on the module allows to write and read files with different names on one single level (no subdirectories are supported).

You can run on the module a single script at a time.

The Python script is executed in a task inside the Telit module at the lowest priority, making sure this does not interfere with GSM/GPRS/UMTS normal operations. This allows serial ports, protocol stack etc. to run independently from the Python script.

The Python script interacts with the Telit module functionality through build-in interfaces.

The following sections describe the example scripts that are provided with AppZone Python.

5.2.1. ab_test.py

Script description: the script tries to open the “test.bin” file. If the file already exists, the program appends some data; otherwise a new file is created with the “wb” parameter

User inputs: none

Availability: any Telit product supporting Python

5.2.2. ADC_test.py

Script description: the script makes a loop with the following operations (the permitted VMax of the input is 2V):

- the script measures the ADC value at the ADC_IN1 pin, by issuing AT#ADC and AT#ADC=1,2, - using GPIO.getADC prints the result on the debug port - sleep for 0,5 sec

User inputs: none

Availability: any Telit product supporting Python

5.2.3. fdi.py

Script description: un-expected power loss with file left open

User inputs: none

Availability: any Telit product supporting Python

5.2.4. FSys_mngtst2.py

Script description: the script modifies the 5th byte inside the file test2.txt (size 4000 KB) stored in flash

User inputs: none

Availability: any Telit product supporting Python

5.2.5. FTP.py

Script description: the script shows capabilities of Telit embedded FTP doing upload and download of a text file

User inputs: edit these values in the script

- SIMPIN = '***' #SIM PIN
- GPRS_APN = 'xxx.xxxxxxx.xx' # APN, ask your network provider the correct value
- GPRS_USER = “” # GPRS username
- GPRS_PASSW =”” # GPRS password
- FTPSERVER_ADDR = ftp.byte*** # FTP server name
- FTPUSERNAME = '*****' # FTP username
- FTPPASSWORD = '*****' # FTP password
- FTPFILECONTENT = 'testing 123' # string to be sent the server

Availability: any Telit product supporting Python

5.2.6. gpiout_tst2.py

Script description: the script sets the GPIO5 as an output and cycles between 1 and 0 values with a clock period of approximately 2.4 seconds.

User inputs: none.

Availability: any Telit product supporting Python

5.2.7. gpsfence.py

Script description: the script find the geographical position using the gps. When the module move away from this position for a distance bigger than the fence radius, it sends a sms. User inputs: edit these values in the script

FENCE_RADIUS = 50 # outside this radius send an sms

TEL_NUMBER = '1111111111' # to be changed, tel number to which send an sms when outside the fence

Availability: any Telit product supporting Python and GPS

5.2.8. listenSMS.py

Script description: the script receives an unsolicited result code if a SMS is received and print it on the debug com port. Then the script collects the content of the SMS received and prints it on the debug com port.

User inputs: the user should know the mobile number of the SIM card used by the module.

Availability: any Telit product supporting Python

5.2.9. loc_xE910.py

Script description: the script try to find the geographical position, connecting to opencellid.org and sending the values of the module mmc, mnc, cellid and lac. In response the opencellid site return the latitude and longitude.

User inputs: edit these values in the script

```
GPRS_APN = 'xxx.xxxxxxx.xx'          # APN, ask your network provider
                                     the correct value
GPRS_USER = “”                       # GPRS username
GPRS_PASSW =””                       # GPRS password
```

Availability: xE910 supporting Python

5.2.10. otaGE.py

Script description: the script is a simple OTA (Over the Air) update of a python script chosen in the SMS sent to the module. Inside the text of the SMS it is written the name of the python file to be downloaded from a FTP server. After the download, the file is saved, enabled and a reboot of the module follows. All the steps are printed on the debug com port. A SMS message is sent back from the module with the result of the update procedure.

User inputs: the user should know the mobile number of the SIM card used by the module, and send an SMS with the name of the python file to update in it.

Availability: any Telit product supporting Python except the xE910.

5.2.11. ota_xE910.py

Script description: the script is a simple OTA (Over the Air) update of a python script chosen in the SMS sent to the module. Inside the text of the SMS it is written the name of the python file to be downloaded from a FTP server. After the download, the file is saved, enabled and a reboot of the module follows. All the steps are printed on the debug com port. A SMS message is sent back from the module with the result of the update procedure. This script is similar to the otaGE.py

User inputs: the user should know the mobile number of the SIM card used by the module and send an SMS with the name of the python file to update in it.

Availability: xE910 supporting Python

5.2.12. PowerSaving.py

Script description: the script does a loop with the following operations:

put the module in power saving mode for 5 minutes and then wakes it up - open a TCP/IP socket and send a string to a server - sleep for 5 seconds.

User inputs: edit these values in the script

```
GPRS_APN = 'xxx.xxxxxxx.xx'          # APN, ask your network provider
                                     the correct value
GPRS_USER = “”                       # GPRS username
GPRS_PASSW =””                       # GPRS password
SERVER_ADDR = 'xxx.xxx.xxx.xxx' # Target server address
```

SERVER_PORT = xxxx # Target server port

stringtosend = 'testing 123\r' # String to be sent the server

Availability: any Telit product supporting Python

5.2.13. send_email.py

Script description: the script sends e-mail using AT#SEMAIL custom command.

User inputs: edit these values in the script

- GPRS_APN = 'xxx.xxxxxxx.xx' # APN, ask your network provider the correct value
- GPRS_USER = “” # GPRS username
- GPRS_PASSW =”” # GPRS password
- SMTP_SERVER = "xxx.xxx.xxx" # SMTP server
- SMTP_USERID = "xxxx@xxx.xxx" # SMTP username
- SMTP_PASSW = "xxxxx" # SMTP password

FROM_EMAIL_ADDR = " xxxx@xxx.xxx " # SMTP From

TO_EMAIL_ADDR = " xxxx@xxx.xxx " # SMTP To

SUBJECT = "xxxxxxxxxxxx" # email subject

BODY = "testing 123" # email text body

Availability: any Telit product supporting Python

We recommend that you use email accounts that use SSL ciphering to maintain security.

5.2.14. sendSMS.py

Script description: the script sends an SMS to a destination number in text mode.

User inputs: edit these values in the script

SMSTO = '072*******' # destination number

SMSTEXT = 'testing 123' # SMS text

Availability: any Telit product supporting Python

5.2.15. SER_MDM.py

Script description: the script shows the basic idea for a bi-directional SER to MDM bridge

User inputs: none.

Availability: any Telit product supporting Python

5.2.16. SER_send_rcv.py

Script description: the script receives and sends data on SER interface. The user should write something in the Port Terminal after “Write text:” string

User inputs: none.

Availability: any Telit product supporting Python

5.2.17. SERtest.py

Script description: the script sends a text string 'TEST' to the serial port (ASC0) every 500ms.

User inputs: none.

Availability: any Telit product supporting Python

5.2.18. SKT_CL_SR.py

Script description: the script executes following steps:

- script configuration (IP address, remote & local ports, APN, PIN, PUK, NET_Operator)
- check SIM status, insert SIM PIN or PUK when needed, and check for network registration - socket configuration and GPRS context activation
- client or server selection based on the input char received ('c' or 's' character to select one of the
- 2 options) ○ start client mode dialing the remote socket (skt1)
  - start server mode configuring firewall and socket listen (skt2)
- when client to server connection is established, every character sent on the serial port of one module is received by the remote module
- suspend the socket with "+++" sequence
- since MDM-SER bridge is active it is possible to send AT commands (AT#SO=1 to reconnect the socket or AT#SH=1 to close the connection)
- the module is rebooted once DCD low status is detected

User inputs: edit these values in the script

- NETWORK= "xxx" # this is the MNO name
- PIN = "" # SIM card PIN
- PUK = "" # SIM card PUK
- NEW_PIN = "" # SIM card PIN
- LOCAL_PORT = xxxxx # Local server port
- REMOTE_IP = "xxx.xxx.xxx.xxx" # Remote server address
- SUBNET_MASK = "255.0.0.0"
- REMOTE_PORT = "xxxx" # Remote server port
- APN = "xxx.xxxxxxxx.xx" # APN, ask your network provider the correct value

Availability: any Telit product supporting Python

5.2.19. sock_MDM2.py

Script description: the script sends data to a server through a TCP socket on MDM while using MDM2 for other AT actions

User inputs: edit these values in the script

- GPRS_APN = 'xxx.xxxxx.xx' # APN, ask your network provider
- the correct value

- GPRS_USER = ‘’ ‘’ # GPRS username
- GPRS_PASSW =”” # GPRS password
- SERVER_ADDR = 'xxx.xxx.xxx.xxx' # Target server address
- SERVER_PORT = xxxx # Target server port
- stringtosend = 'testing 123\r' # string to be sent the server

Availability: any Telit product supporting Python

5.2.20. suHE910.py

Script description: the script works similar to the Network Survey command #CSURV, not present in the HE910 module AT commands. The script takes some time to execute, it does two survey, one for GSM operators and one for the UTRAN operators using the AT#MONI command.

User inputs: none.

Availability: HE910 supporting Python

5.2.21. testStrArg.py

Script description: the script is to test the string allocation. It seems that no space is allocated in the names list for strings delimited by '' and ending with \r

User inputs: none

Availability: any Telit product supporting Python

5.2.22. traceback_1.py

Script description: the script gets exception details and formats them.

User inputs: none

Availability: any Telit product supporting Python

5.2.23. traceonSER.py

Script description: the script redirects print output to SER interface. It’s useful to debug scripts without the need to rely on CMUX.

User inputs: none

Availability: any Telit product supporting Python

5.2.24. floattst.py

Script description: the script shows the usage of the float type

User inputs: none

Availability: xE910 supporting Python

5.2.25. sdio.py

Script description: the script package contains 4 files and performs SD card raw data writing and reading using SPI bus. Files contained in the package are:

- main.py
- sd_defs.py
- sd_raw_drv.py
- sdio.py

User inputs: edit these GPIO values in the script

SCLK = 9 # SCLK: GPIO number used as SCLK

MOSI = 12 as MOSI	# MOSI: GPIO number used
MISO = 11	# MISO: GPIO number used as MISO
SS = 8	# SS: GPIO number used as SS

Applicability: GSM/GPRS Telit products supporting Python1.5.2+ (e.g. GE864, GE865, GL865, GE910-QUAD V3, etc.)

5.2.26. watchd_test.py

Script description: the script is a test for the watchdog function. When the "i" loop counter reaches 2000, the module goes to sleep for 70 seconds. After 60 sec. of the 70 sec. the module restarts due to watchdog timeout

User inputs: none

Applicability: GSM/GPRS Telit products supporting Python1.5.2+ (e.g. GE864, GE865, GL865, GE910-QUAD V3, etc.)

5.2.27. threadstst.py

Script description: the script shows the usage of threads allocating dynamically two concurrent threads

User inputs: none

Applicability: xE910 supporting Python

6. RUN THE APPLICATION

You can run the application on the module or test the application locally on your PC.

6.1. Connect the Module

The connection to the module varies based on the type of module.

The port configurations of the modules contained in Table 1 are described in document [11]. The port configuration of the modules contained in Table 2 are described in document [12].

You must configure the ports properly or you will not be able to connect to the module.

Connect the module to the PC, and then click **AutoConnect COM Port** (). The PY Console starts searching for available COM Ports.

If the module belongs to the modules listed in Table 1, PY Console:

1. Sets #PORTCFG=3,3
2. Reboots the module connected.
3. Displays the result as shown in the figure below.

See document [11] to know the available port configurations managed with the AT#PORTCFG command.

MODULES
HE910 Series
UE910 Series
UL865 Series
CE910 Series
DE910 Series

Table 1: HE/UE/UL Series and #PORTCFG configuration

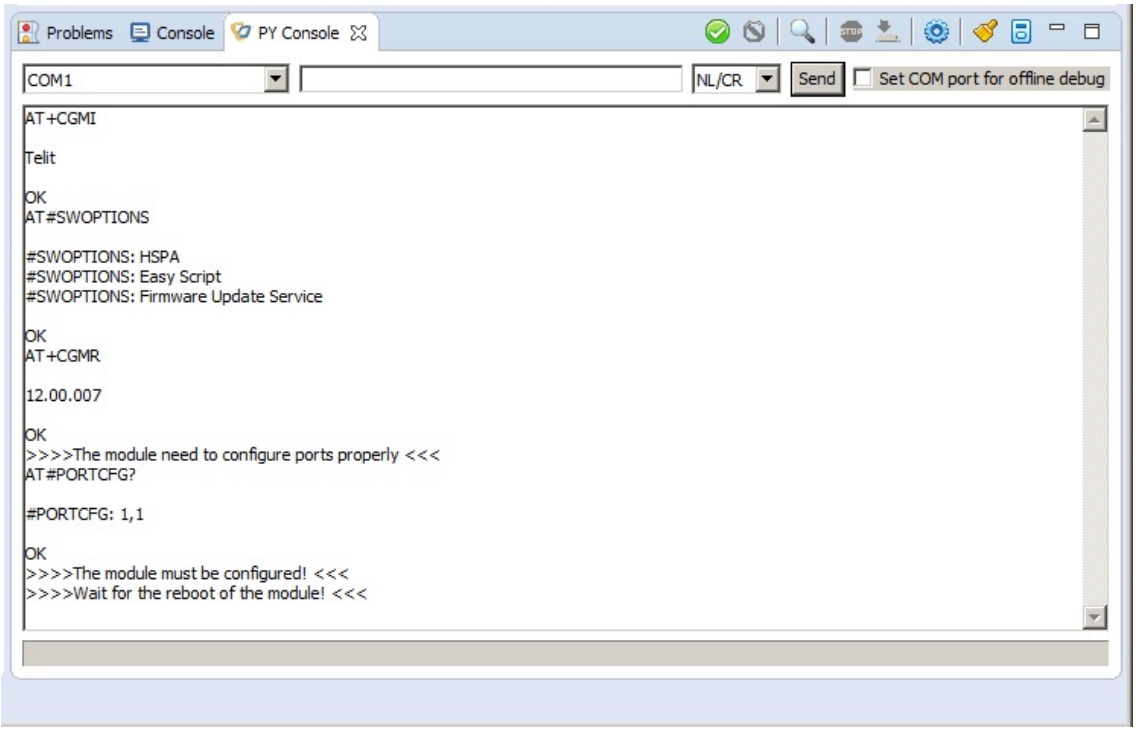


Fig. 1: PY Console after Module Checking

The default configuration of the module for #PORTCFG and #STARTMODESCR is as follows:

AT#PORTCFG?

#PORTCFG: 1,1

OK

AT#STARTMODESCR?

#STARTMODESCR: 0,10

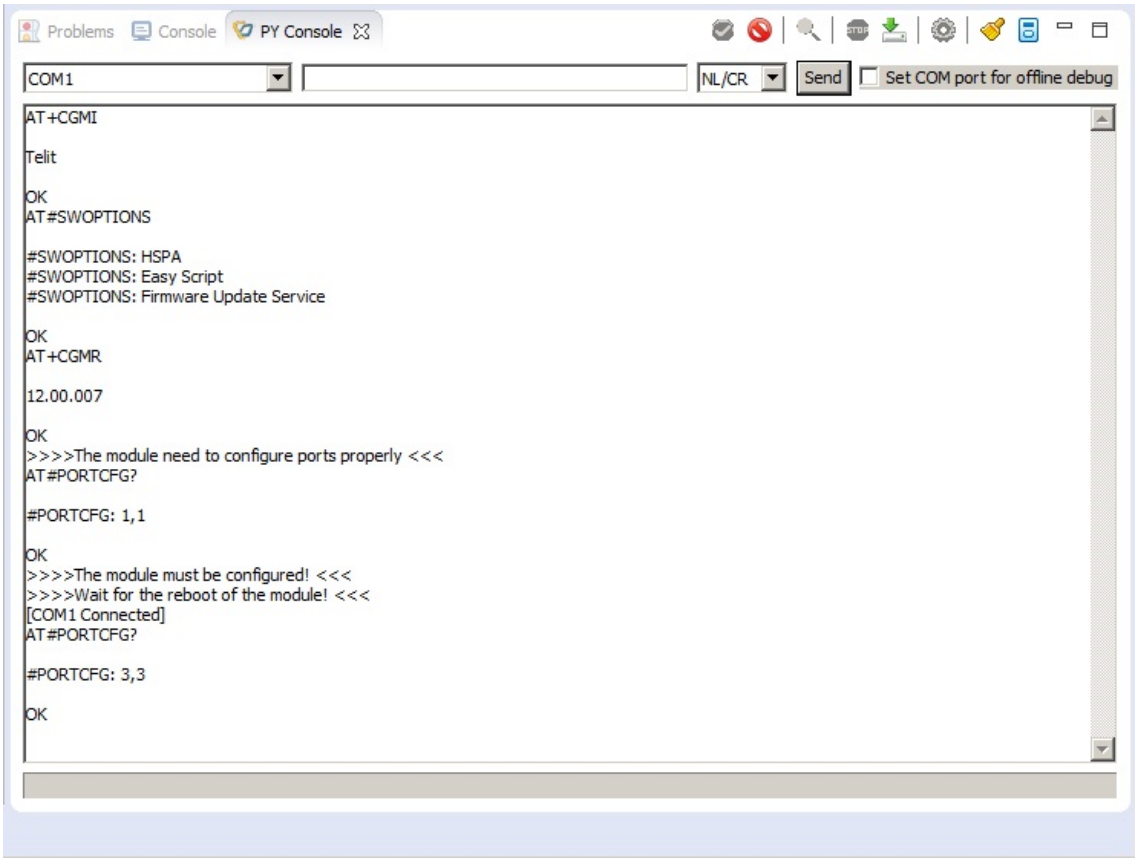
OK

For more information about AT commands see document [6].

To connect to the module:

1. Select COM1 port.
2. Click **Connect COM Port** (). The [COM1 Connect] message is displayed in the PY Console.
3. In the command area, type the following command to check that PORTCFG configuration is #PORTCFG: 3,3:

   AT=PORTCFG?

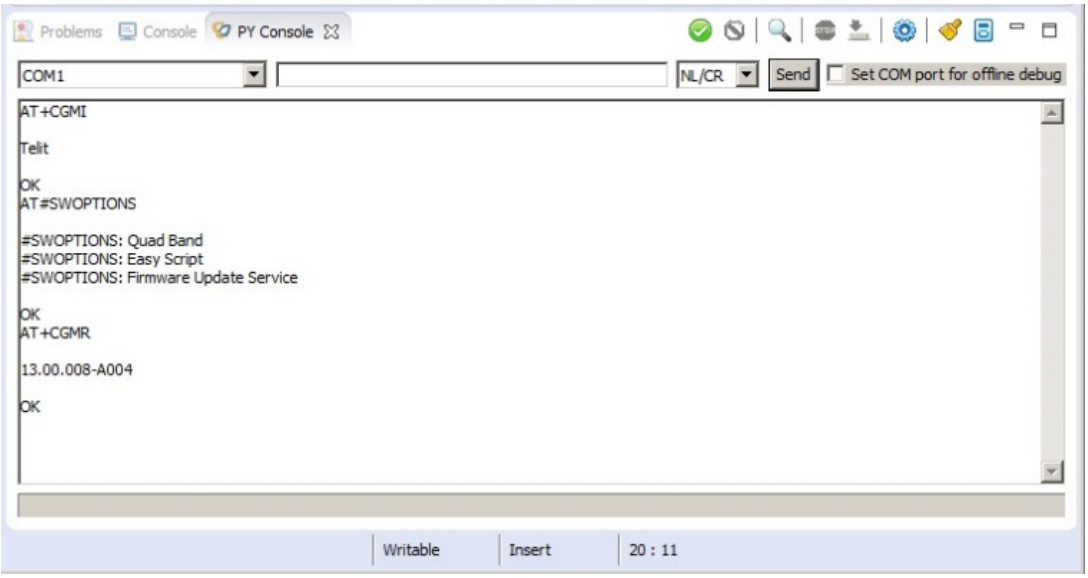


If the module belongs to the modules listed in Table 2, the PY Console does not modify the #PORTCFG configuration and does not reboot the module.

See document [12] to know the available port configurations managed with the AT#PORTCFG command.

MODULES
GE910 Series

Table 2: GE910 Series and #PORTCFG configuration

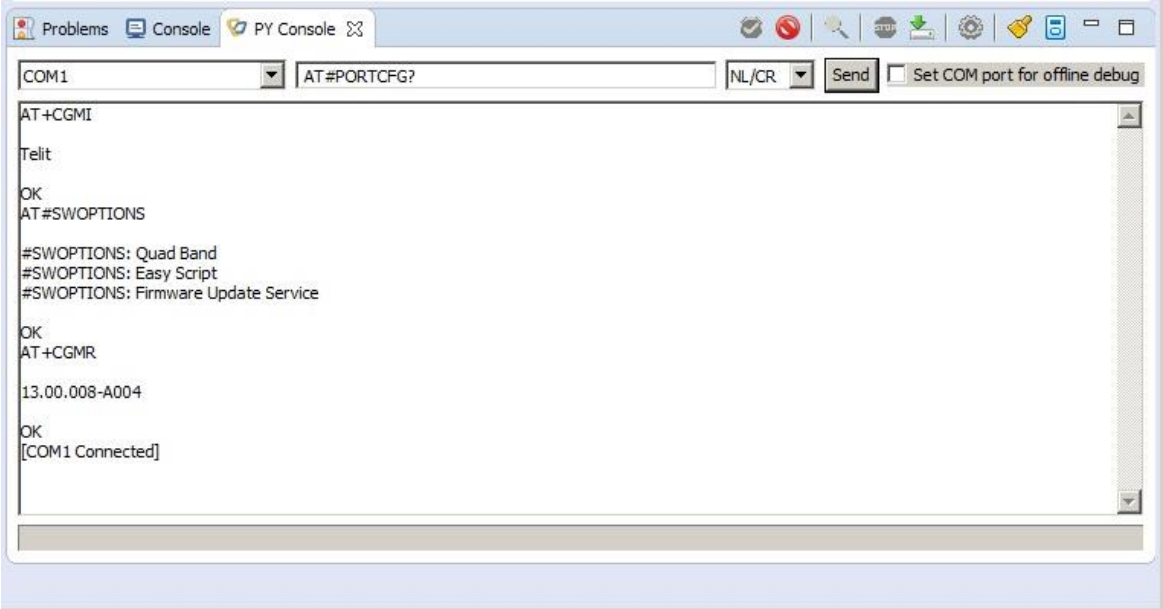


To connect to the module:

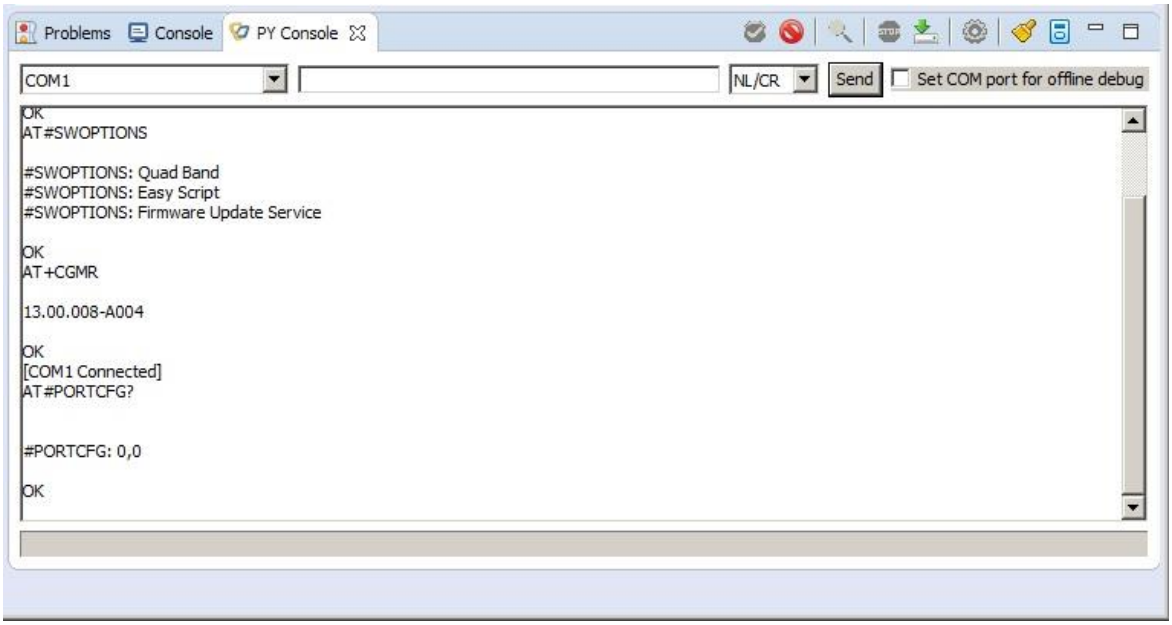
1. Select COM1 port.
2. Click **Connect COM Port** (). The [COM1 Connect] message is displayed in the PY Console.

3. In the command area, type the following command to check that the PORTCFG configuration is #PORTCFG: 0,0:

   AT=PORTCFG?



As shown in the following figure, the port configuration is #PORTCFG:0,0:



The default configuration of the module for #PORTCFG and #STARTMODESCR commands is as follows:

AT#PORTCFG?

#PORTCFG: 0,0

OK

AT#STARTMODESCR?

#STARTMODESCR: 0,10

OK

6.2. Run Python Scripts on the Module

After you connect AppZone Python to the module, you can run the Python scripts on the module.

This section describes how to run the application on the modules. The examples use a module defined in Table 1. The same concepts are applicable to the modules listed in Table 2

To upload the My_First_Py.py script in the src directory of the module, drag and drop it as show in Fig. 2.

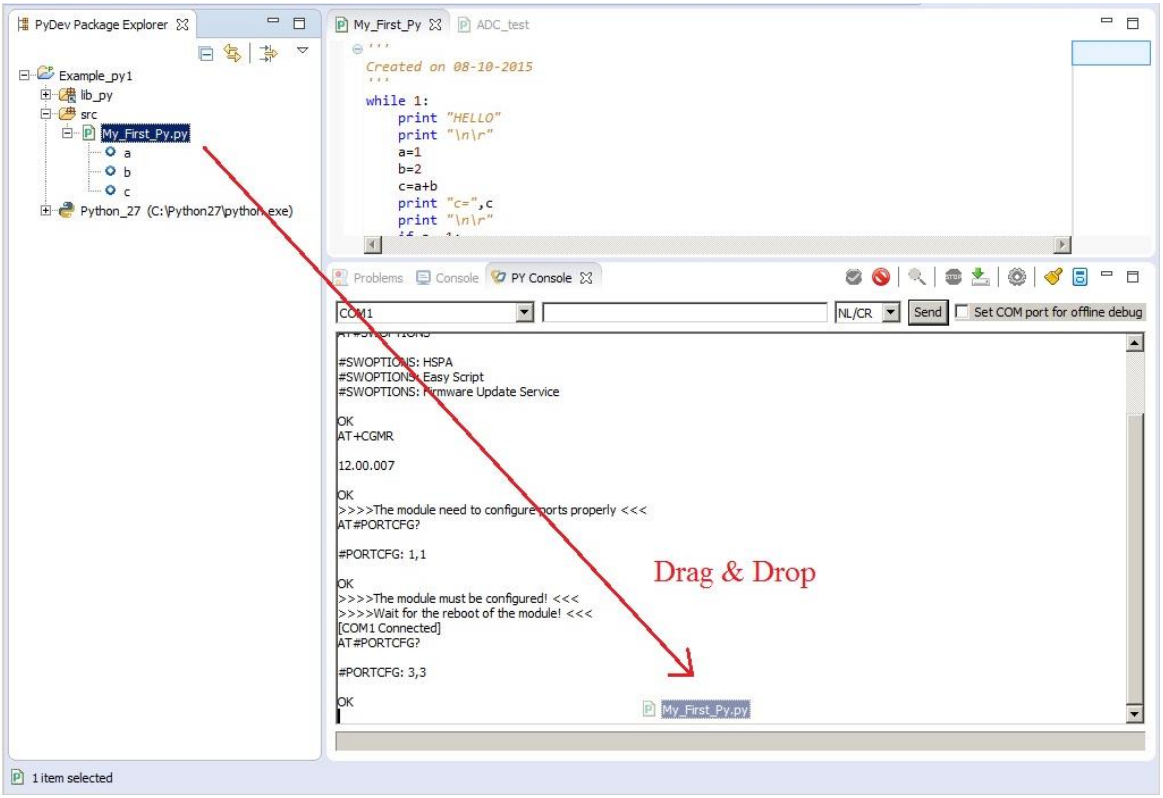
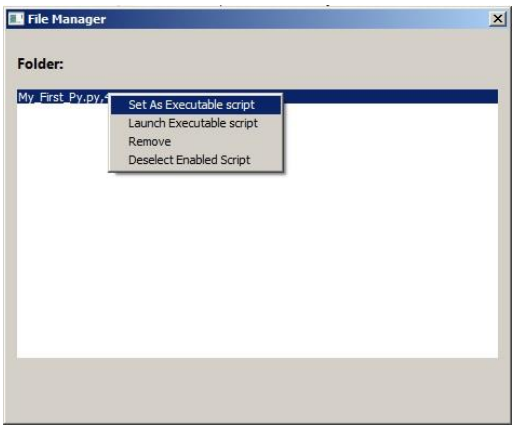


Fig. 2: Python, Drag & Drop

To run the script:

1. Click **File Manager** ( ).
2. Right-click the script.

The PY Console displays the menu shown in the next figure. The items of the menu are self-explanatory, see the equivalent AT command (AT#ESCRIPT="<script_name>", AT#EXECSCR ...) described in the documents [6]/[7].



My_First_Py script

```
while 1:
    print "HELLO"
    print "\n\r"
    a=1
    b=2
    c=a+b
    print "c=",c
    print "\n\r"
    if a==1:
        print "a=",a
        print "\n\r"
    else:
        print "b=\n\r",b
```

Execute the following steps:

1. Connect a terminal to the USIF1 port as shown in Fig. 3.
2. Set the uploaded script as an executable script, and launch it. See the screenshot above. *Print* instruction sends the messages on USIF1 port. The terminal displays the messages shown in the right screenshot.
3. You can send AT commands to AT0 parser by means of the PY Console.

```
c= 3

a= 1

HELLO

c= 3

a= 1

HELLO

c= 3

a= 1
```

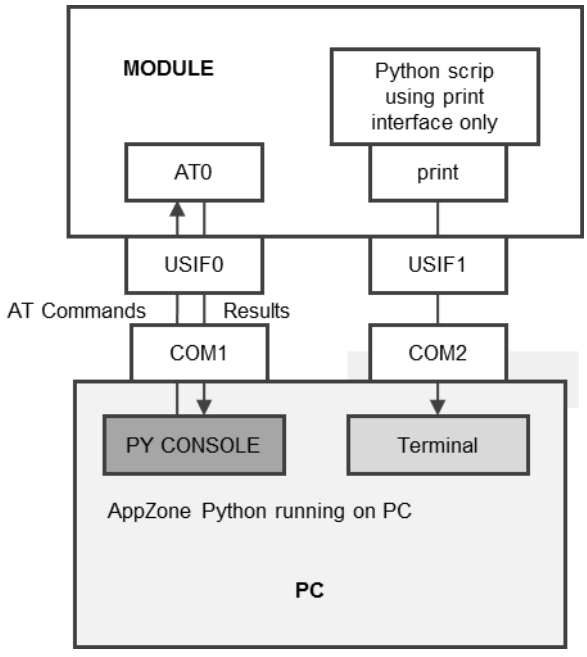


Fig. 3: Python script (print interface only) running on Module

#STARTMODESCR is 0,10 (factory setting), and #PORTCFG=3,3.

6.3. Run Python Scripts Locally

You can also run Python scripts locally on your computer for test purposes. When testing scripts locally on your computer, you must make sure to use the interfaces that the scripts uses correctly. Documents [11]/[12] describe in detail interfaces, such as MDM, SER, and print in accordance with the #PORTCFG configuration used. The following sections provide some examples using a different set of interfaces.

6.3.1. Print Interface Example

In this example, the script uses the *print* interface only. The script sends the messages to the AppZone Phy Console. The module is not needed and it can be powered off. Fig. 4 summarizes this behavior.

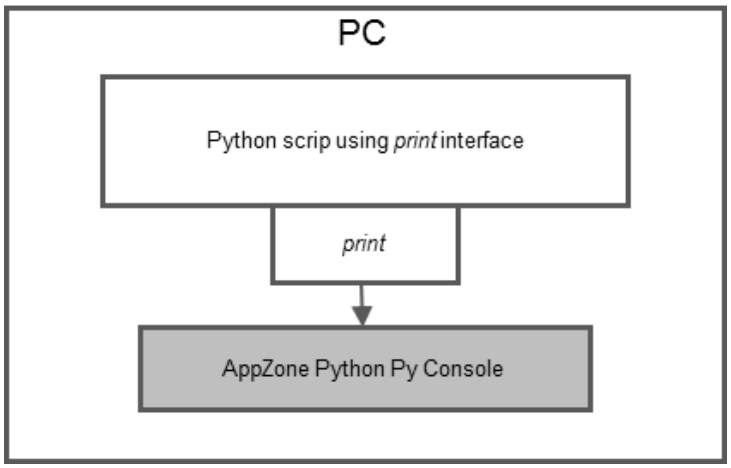
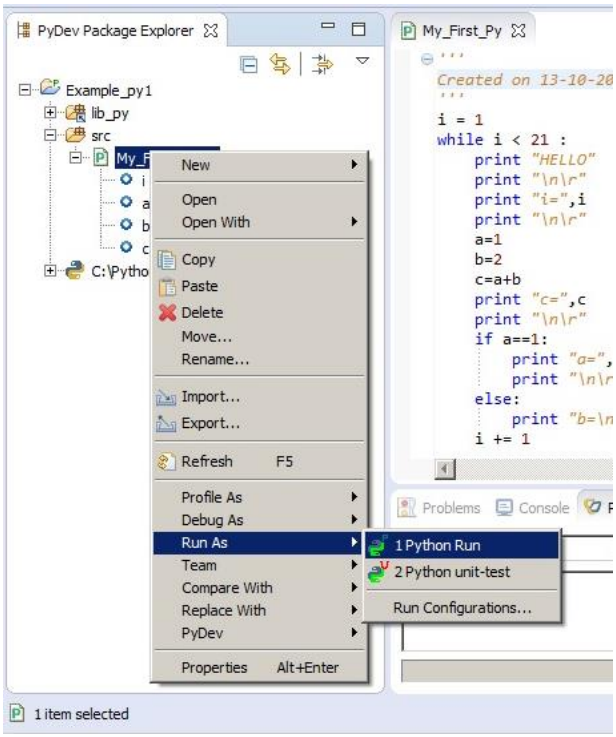


Fig. 4: Python script (*print* interface only) running on local computer

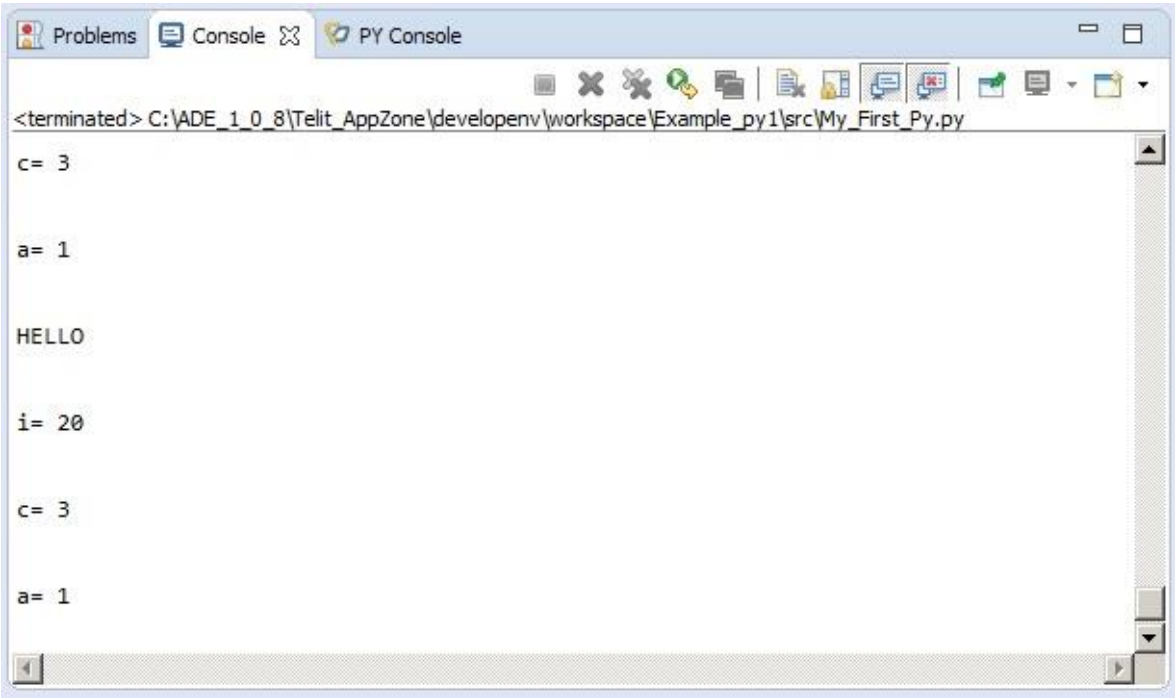
The Python script uses the files stored in the lib_py directory of the project.

To start the script:

1. Right-click My_First_Py script, and then select **Run As > Python Run** or **Debug As > Python Run**.



My_First_Py script, running on the computer, sends messages on the Console. See the following figure.



You can debug the script using the debug facilities provided by Eclipse multi-language development environment (IDE).

6.3.2. Debug Scripts

To debug local python applications that send data to the serial port, you must use a virtual COM port pair to see the debug prints. A virtual COM port pair consists of the following two virtual COM ports that are connected internally:

- Port connected to Eclipse
- Port opened from a COM port terminal

In Linux:

To create the COM port pair on Linux, enter the following command:

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
```

The console displays the names of the new COM ports that were created.

To view the debug prints, use a minicom terminal.

In Windows: You can use free tools to create virtual COM ports (such as http://com0com.sourceforge.net/).

Use a COM port terminal (such as, https://ttssh2.osdn.jp/index.html.en) to read the output.

Note: These links for downloadable software are third party software that is not affiliated to Telit and are included here as possible suggestions only to assist the user.

6.4. Example: Run Script Locally and on the Module

6.4.1. Run Script Locally

In this example, the script uses MDM, *print*, and *SER* interfaces:

```
import SER
import MDM
import time

i = 1

while i < 21 :
    print "HELLO"
    print "\n\r"
    SER.send("HELLO SER\r")
    MDM.send("AT+CGMR\r", 10)
    time.sleep(1)
    r=MDM.read()
    print r
    a=1
    b=2
    c=a+b
    print "c=",c
    print "\n\r"
    print "i=",i
    print "\n\r"
    if a==1:
        print "a=",a
        print "\n\r"
    else:
        print "b=\n\r",b

    i += 1
```

Fig. 5: My_First_Py script with *print*, MDM, SER interfaces

In accordance with MDM interface functionality, the module must be powered on, and connected to the computer as shown in Fig. 6.

- MDM interface must be connected to an AT parser, accordingly Python script must be connected to a module through an available COM port chosen by the user.
- *print* interface is automatically connected to the AppZone Python Console.
- SER interface can be connected to a User Device through a COM port chosen by the user.

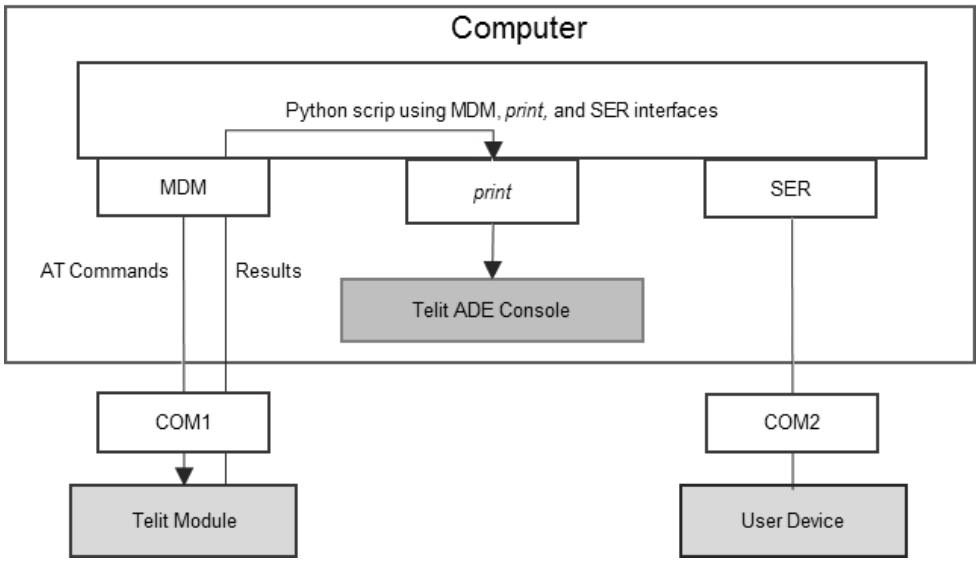
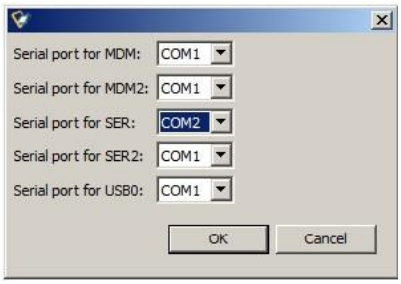


Fig. 6: Python script (MDM, *print*, SER interfaces) running on PC

The My_First_Py script uses the files stored in the lib_py directory of the project.

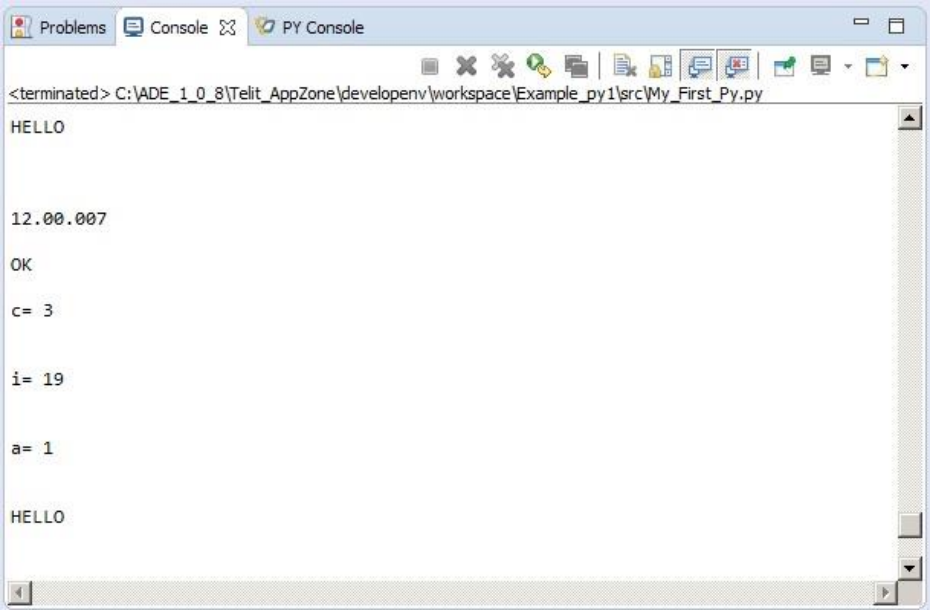
To configure AppZone Python to run the example in Fig. 6:

1. Click **Connect COM Port** ( ) to search the available COM ports. In this PC configuration they are COM1, and COM2.
2. Click **Set Ports Simulation** ( ) and select COM1 for MDM interface and COM2 for SER interface.



3. Right-click the script name and then select **Run As > Python Run** or **Debug As > Python Run**.

   See the outputs of the script on the AppZone Python Console. The script uses the *print* interface



SER interface sends the message "HELLO SER" to the User Device by means of the COM2 port; refer to the script code in Fig. 5, and the basic architecture in Fig. 6.

6.4.1.1 Run Script on the Module

You can run the Python script shown in Fig. 5 on the module. The script runs on the module and uses MDM, *print, and SER* interfaces as when it was running on PC.

- MDM interface is directly connected to AT0 parser.
- *print* interface is connected to USIF1 port.
- SER interface is connected to USIF0 port.

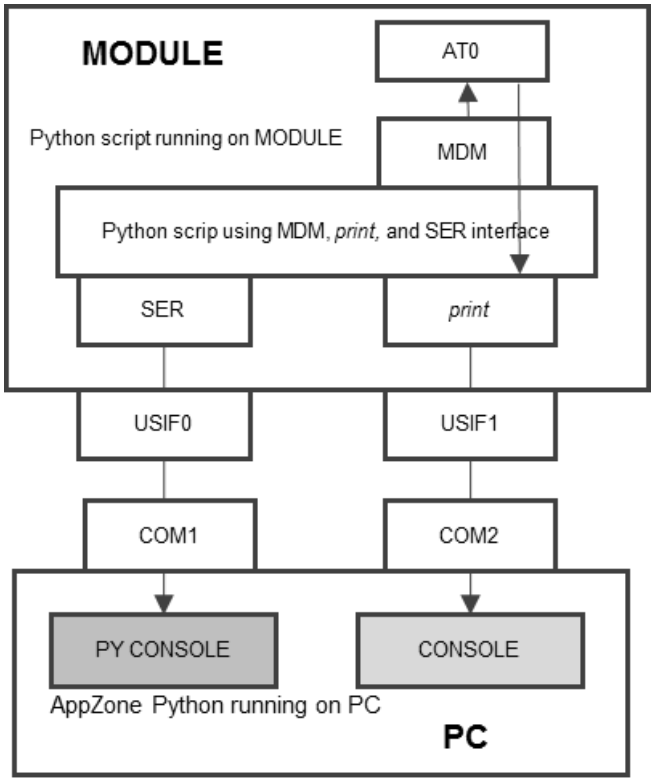
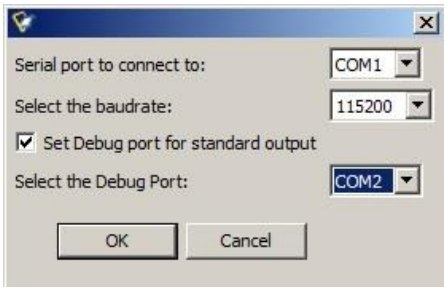


Fig. 7: Python script (MDM, *print*, SER interfaces) runs on MODULE

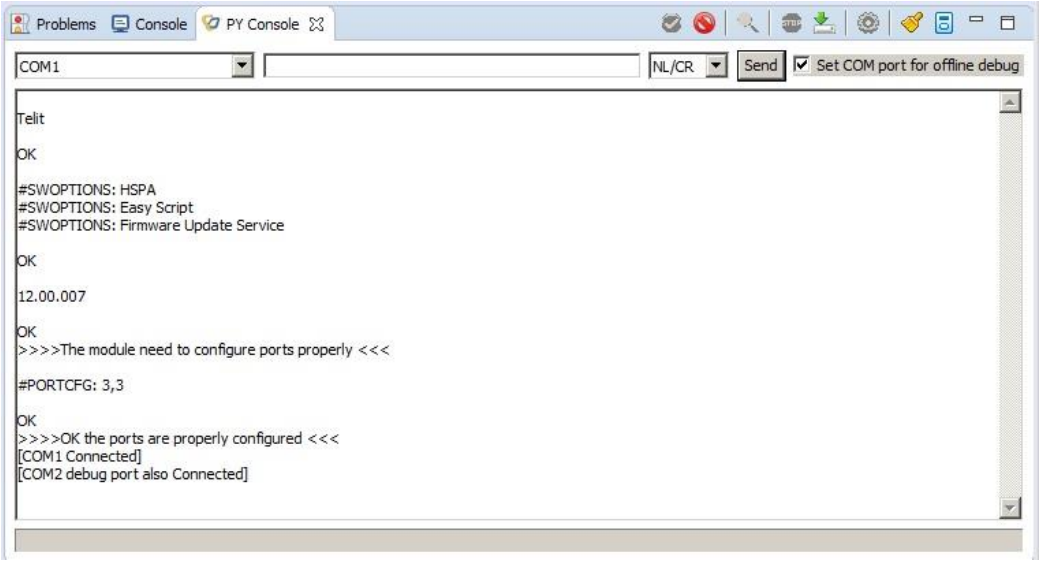
To configure AppZone Python in accordance with the Fig. 7:

1. Start AppZone Python. Assume that the My_First_Py script is already uploaded.
2. Click **AutoConnect COM Port** ( ) to search the available COM ports. In this PC configuration they are COM1, and COM2.
3. Select the Set COM port for offline debug checkbox.
4. Click **Setting** ( ), and then select the **Set Debug port for standard output** checkbox.
5. Select COM2 as debug port. Standard output is AppZone Python Console



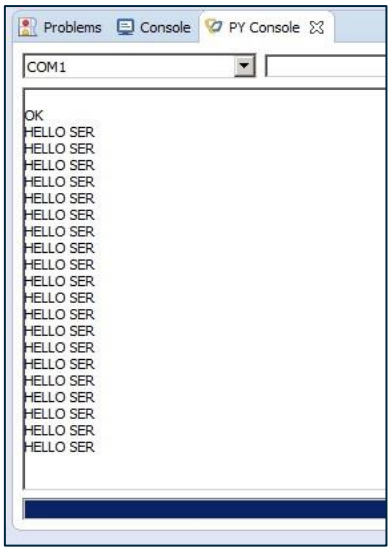
6. Click **Connect COM port** ( ) to connect COM1 port to the PY Console.

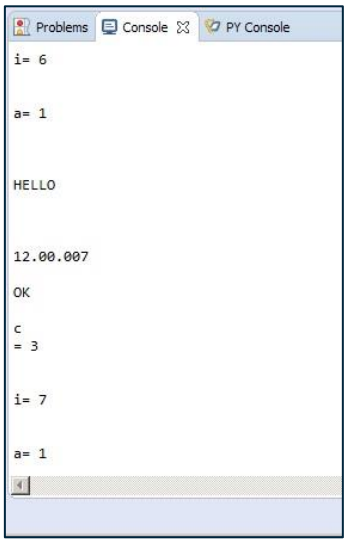
   The following messages are displayed on PY Console, see next screenshot.



7. Click **File Manage**r ( ) and select the script.
8. Right-click the script, and then select **Set as Executable Script**.
9. Right-click the script, and then select **Launch Executable script**.

See the outputs of the script on the PY Console (COM1) and on the AppZone Python Console (COM2).





7. MODULES & SW VER. TABLES

SOFTWARE VER. TABLE

	SW Versions	Comment
HE910 Series		
HE910	**12.00.xx7**	HE910 is the type name of the products marketed as HE910-G & HE910-DG
UE910 Series		
UE910-EUR / UE910-EUD	**12.00.xx7**	
UE910-NAR / UE910-NAD	**12.00.xx7**	
UL865 Series		
UL865-EUR / UL865-EUD	**12.00.xx7**	
UL865-NAR / UL865-NAD	**12.00.xx7**	
UE866 Series		
UE866-N3G	**12.00.xx7**	
GE910 Series		
GE910-QUAD	**13.00.xx8**	
GE910-GNSS	**13.00.xx8**	

The table below summarizes the Services provided by the modules, and shows their coexistence. The available Service depends on the software version installed on the modules. Embedded/External GPS are beyond the scope of this guide.

SERVICES COEXISTENCE TABLE

	Services			
	Embedded GPS	External GPS	AppZone Python	AppZone C
			AZ Python and AZ C are mutually exclusive	
HE910 Series				
HE910	✓		✓	✓*
UE910 Series				
UE910-EUR / UE910-EUD		✓	✓	✓*
UE910-NAR / UE910-NAD		✓	✓	✓*
UL865 Series				
UL865-EUR / UL865-EUD		✓	✓	✓*
UL865-NAR / UL865-NAD		✓	✓	✓*
UE866 Series				
UE866-N3G		✓	✓	✓*
GE910 Series				
GE910-QUAD		✓	✓	✓**
GE910-GNSS	✓		✓	✓**
LE910 Series				
LE910v2		✓		✓

(*): AppZone C available on specific part numbers. Available by default starting from release 12.00.xx8.

(**): AppZone C available on specific part numbers. Available by default starting from release 13.00.xx9.

DOCUMENT HISTORY

Revision	Date	Changes
8	2016-01	Separated from AppZone C User Guide and updated entire document to support integrated IDE


Design Review
120°
180°
Process
Validation
240°
RF/EMC Pre-Certification
Module Selection
60°
300°
Approvals
Support
360°
Value Added Services & Connectivity
Enterprise Integration &
Application Enablement
m2mair