

HE910 Easy Script in Python

80378ST10106A Rev.0 – 2012-02-27



APPLICABILITY TABLE

PRODUCT
HE910-G
HE910-DG
HE910-D
HE910-EUD
HE910-EUR
HE910-EUG
HE910-NAD
HE910-NAR
HE910-NAG

From SW Versions:

12.00.xx2



SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE

Notice

While reasonable efforts have been made to assure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from use of the information obtained herein. The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies or omissions. Telit reserves the right to make changes to any products described herein and reserves the right to revise this document and to make changes from time to time in content hereof with no obligation to notify any person of revisions or changes. Telit does not assume any liability arising out of the application or use of any product, software, or circuit described herein; neither does it convey license under its patent rights or the rights of others.

It is possible that this publication may contain references to, or information about Telit products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Telit intends to announce such Telit products, programming, or services in your country.

Copyrights

This instruction manual and the Telit products described in this instruction manual may be, include or describe copyrighted Telit material, such as computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and its licensors certain exclusive rights for copyrighted material, including the exclusive right to copy, reproduce in any form, distribute and make derivative works of the copyrighted material. Accordingly, any copyrighted material of Telit and its licensors contained herein or in the Telit products described in this instruction manual may not be copied, reproduced, distributed, merged or modified in any manner without the express written permission of Telit. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit, as arises by operation of law in the sale of a product.

Computer Software Copyrights

The Telit and 3rd Party supplied Software (SW) products described in this instruction manual may include copyrighted Telit and other 3rd Party supplied computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and other 3rd Party supplied SW certain exclusive rights for copyrighted computer programs, including the exclusive right to copy or reproduce in any form the copyrighted computer program. Accordingly, any copyrighted Telit or other 3rd Party supplied SW computer programs contained in the Telit products described in this instruction manual may not be copied (reverse engineered) or reproduced in any manner without the express written permission of Telit or the 3rd Party SW supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit or other 3rd Party supplied SW, except for the normal non-exclusive, royalty free license to use that arises by operation of law in the sale of a product.



Usage and Disclosure Restrictions

License Agreements

The software described in this document is the property of Telit and its licensors. It is furnished by express license agreement only and may be used only in accordance with the terms of such an agreement.

Copyrighted Materials

Software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, without prior written permission of Telit

High Risk Materials

Components, units, or third-party products used in the product described herein are NOT fault-tolerant and are NOT designed, manufactured, or intended for use as on-line control equipment in the following hazardous environments requiring fail-safe controls: the operation of Nuclear Facilities, Aircraft Navigation or Aircraft Communication Systems, Air Traffic Control, Life Support, or Weapons Systems (High Risk Activities"). Telit and its supplier(s) specifically disclaim any expressed or implied warranty of fitness for such High Risk Activities.

Trademarks

TELIT and the Stylized T Logo are registered in Trademark Office. All other product or service names are the property of their respective owners.

Copyright © Telit Communications S.p.A.



Contents

1. Introduction	10
1.1. Scope	10
1.2. Audience.....	10
1.3. Contact Information, Support.....	10
1.4. Document Organization	11
1.5. Text Conventions	11
1.6. Related Documents	11
2. Easy Script Extension - Python interpreter	12
2.1. Overview	12
2.2. Python 2.7.2 Copyright Notice.....	14
2.3. Python.....	14
2.4. Python implementation description.....	14
2.5. Python supported features	16
3. Python Script Operations	17
3.1. Executing the Python script.....	17
3.1.1. Write the Python script	17
3.1.2. Compile the Python script.....	17
3.1.3. Download the Python script.....	17
3.1.4. Enable the Python script	20
3.1.5. Run the Python script.....	21
3.1.6. Read the Python script	21
3.1.7. List saved Python scripts.....	22
3.1.8. Delete the Python script	22
3.2. Debug Python script	23
4. Python Build-in Custom Modules.....	24
4.1. MDM built-in module	24
4.1.1. MDM.send(string, timeout)	24
4.1.2. MDM.read()	25
4.1.3. MDM.sendbyte(byte, timeout).....	25
4.1.4. MDM.readbyte()	26
4.1.5. MDM.sendavail().....	26
4.1.6. MDM.getDCD().....	26
4.1.7. MDM.getCTS().....	27



4.1.8.	MDM.getDSR()	27
4.1.9.	MDM.getRI()	27
4.1.10.	MDM.setRTS(RTS_value)	27
4.1.11.	MDM.setDTR(DTR_value)	28
4.2.	MDM2 built-in module	29
4.2.1.	MDM2.send(string, timeout)	29
4.2.2.	MDM2.read()	30
4.2.3.	MDM2.sendbyte(byte, timeout)	30
4.2.4.	MDM2.readbyte()	31
4.2.5.	MDM2.sendavail()	31
4.2.6.	MDM2.getDCD()	31
4.2.7.	MDM2.getCTS()	31
4.2.8.	MDM2.getDSR()	32
4.2.9.	MDM2.getRI()	32
4.2.10.	MDM2.setRTS(RTS_value)	32
4.2.11.	MDM2.setDTR(DTR_value)	32
4.3.	SER built-in module	34
4.3.1.	SER.send(string)	34
4.3.2.	SER.read()	34
4.3.3.	SER.sendbyte(byte)	35
4.3.4.	SER.readbyte()	35
4.3.5.	SER.sendavail()	35
4.3.6.	SER.set_speed(speed, <char format>)	36
4.3.7.	SER.setDCD(DCD_value)	36
4.3.8.	SER.setCTS(CTS_value)	36
4.3.9.	SER.setDSR(DSR_value)	37
4.3.10.	SER.setRI(RI_value)	37
4.3.11.	SER.getRTS()	37
4.3.12.	SER.getDTR()	38
4.4.	GPIO built-in module	39
4.4.1.	GPIO.setIOvalue(GPIOnumber, value)	39
4.4.2.	GPIO.getIOvalue(GPIOnumber)	39
4.4.3.	GPIO.setIOdir(GPIOnumber, value, direction)	40
4.4.4.	GPIO.getIOdir(GPIOnumber)	40
4.4.5.	GPIO.getADC(adcNumber)	40
4.4.6.	GPIO.setDAC(enable, value)	41
4.4.7.	GPIO.setVAUX(vauxNumber, enable)	41
4.4.8.	GPIO.getAXE()	41
4.4.9.	GPIO.setSLED(status, onDuration, offDuration)	42



4.4.10.	GPIO.getCBC().....	42
4.5.	GPS built-in module.....	43
4.5.1.	GPS.powerOnOff(newStatus).....	43
4.5.2.	GPS.getPowerOnOff()	43
4.5.3.	GPS.getActualPosition()	43
4.5.4.	GPS.getLastGGA()	44
4.5.5.	GPS.getLastGLL()	44
4.5.6.	GPS.getLastGSA().....	44
4.5.7.	GPS.getLastGSV().....	45
4.5.8.	GPS.getLastRMC()	45
4.5.9.	GPS.getLastVTG().....	45
5.	Python standard functions.....	46
5.1.	Technical characteristics	46
5.1.1.	General.....	46
5.2.	Python supported features	47
5.2.1.	Operators, statements, functions	47
5.2.2.	Built-in Functions	48
5.2.3.	Built-in Constants	49
5.2.4.	Truth Value Testing	50
5.2.5.	Boolean Operations.....	50
5.2.6.	Comparisons	50
5.2.7.	Numeric Types: Integer, Long Integer and Floating Point	50
5.2.8.	Numeric Types: Integer and Long Integer	51
5.2.9.	Numeric Types: Floating Point	51
5.2.10.	Numeric Types: Complex	51
5.2.11.	Iterator Types	51
5.2.12.	Generator Types.....	51
5.2.13.	Sequence Types: String, List, Tuple, Bytearray, Buffer and Xrange.....	51
5.2.14.	Sequence Types: Unicode.....	52
5.2.15.	Sequence Types: String.....	52
5.2.16.	Mutable Sequence Types: List and Bytearray	53
5.2.17.	Set Types: Set and Frozenset	54
5.2.18.	Mapping Types: Dictionary	55
5.2.19.	File Objects	55
5.2.20.	Memoryview Objects.....	56
5.2.21.	Module Objects	56
5.2.22.	Classes and Class Instances	56
5.2.23.	Function Objects	57
5.2.24.	Method Objects	57



5.2.25.	Code objects.....	57
5.2.26.	Type Objects	57
5.2.27.	Null Object.....	57
5.2.28.	Ellipsis Object.....	57
5.2.29.	NotImplemented Object.....	57
5.2.30.	Internal Types: Frame Objects.....	57
5.2.31.	Internal Types: Traceback Objects.....	57
5.2.32.	Slice Objects	57
5.2.33.	Built-in Exceptions	58
5.2.34.	Built-in Modules: marshal	58
5.2.35.	Built-in Modules: imp.....	59
5.2.36.	Built-in Modules: _ast.....	59
5.2.37.	Built-in Modules: __main__	59
5.2.38.	Built-in Modules: __builtin__.....	59
5.2.39.	Built-in Modules: sys	60
5.2.40.	Built-in Modules: exceptions	61
5.2.41.	Built-in Modules: gc	61
5.2.42.	Built-in Modules: _warnings	62
5.2.43.	Built-in Modules: _md5	62
5.2.44.	Built-in Modules: binascii.....	62
5.2.45.	Built-in Modules: _sre	62
5.2.46.	Built-in Modules: _weakref	62
5.2.47.	Built-in Modules: _symtable.....	62
5.2.48.	Built-in Modules: _functools	62
5.2.49.	Built-in Modules: _socket.....	63
5.2.50.	Built-in Modules: time	65
5.2.51.	Built-in Modules: posix	66
5.2.52.	Built-in Modules: thread.....	66
5.2.53.	Built-in Modules: signal.....	67
5.2.54.	Built-in Modules: errno.....	67
5.2.55.	Built-in Modules: cStringIO	67
5.2.56.	Library Modules.....	68

6. Python non standard functions69

6.1.	_socket non standard functions	69
6.1.1.	Non standard socket option flag: SO_CONTEXTID.....	69
6.2.	signal non standard functions.....	70
6.2.1.	Non standard signal: SIGMDM	70
6.2.2.	Non standard signal: SIGMDM2	70



7. Python Notes	71
7.1. Memory Limits.....	71
7.2. Other Limits	71
8. Python Script Emulation on PC.....	72
8.1. Executing the Python script on PC.....	72
8.1.1. Install Python	72
8.1.2. Install optional serial package.....	72
8.1.3. Copy Python modules	72
8.1.4. Run the Python script.....	73
9. Document History.....	74



1. Introduction

1.1. Scope

Aim of this document is to give an overview of the Easy Script Extension feature, which lets the developer to drive the modem *internally*, writing the controlling application directly in a high level language such as Python.

1.2. Audience

This document is intended for Telit customers developing functionalities on their applications.

1.3. Contact Information, Support

For general contact, technical support, to report documentation errors and to order manuals, contact Telit Technical Support Center (TTSC) at:

TS-EMEA@telit.com
TS-NORTHAMERICA@telit.com
TS-LATINAMERICA@telit.com
TS-APAC@telit.com

Alternatively, use:

<http://www.telit.com/en/products/technical-support-center/contact.php>

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

<http://www.telit.com>

To register for product news and announcements or for product questions contact Telit Technical Support Center (TTSC).

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.



1.4. Document Organization

This document contains the following chapters:

[Chapter 1: “Introduction”](#) provides a scope for this document, target audience, contact and support information, and text conventions.

[Chapter 2: “Easy Script Extension – Python interpreter”](#) gives a broad overview about the extension.

[Chapter 3: “Python script operations”](#) deals with the execution of the scripts operatively.

[Chapter 4: “Python built-in custom modules”](#) explains in detail the single custom built-in modules.

[Chapter 5: “Python standard functions”](#) provides a description of Python language supported features.

[Chapter 6: “Python non standard functions”](#) provides a description of non standard functions added to Python language.

[Chapter 7: “Python notes”](#) deals with some Python limits that should be considered while developing scripts.

1.5. Text Conventions



Danger – This information MUST be followed or catastrophic equipment failure or bodily injury may occur.



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. Related Documents

- HE910 AT Commands Reference Guide, 80378ST10091A
- HE Family Ports Arrangements User Guide, 1vv0300971

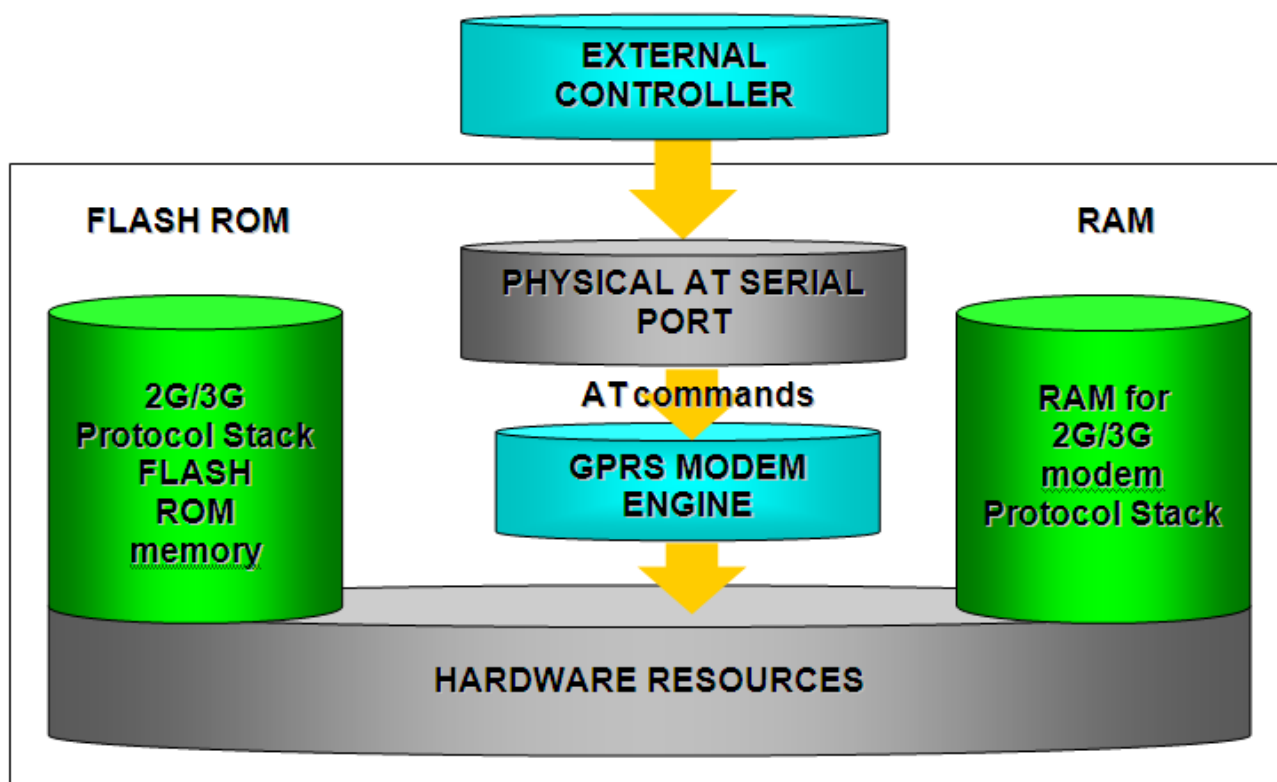


2. Easy Script Extension - Python interpreter

2.1. Overview

The Easy Script Extension is a feature that allows driving the modem *internally*, writing the controlling application directly in the Python high level language. A typical application usually consists of a microcontroller managing data transfer and several I/O pins on the module through the AT command interface.

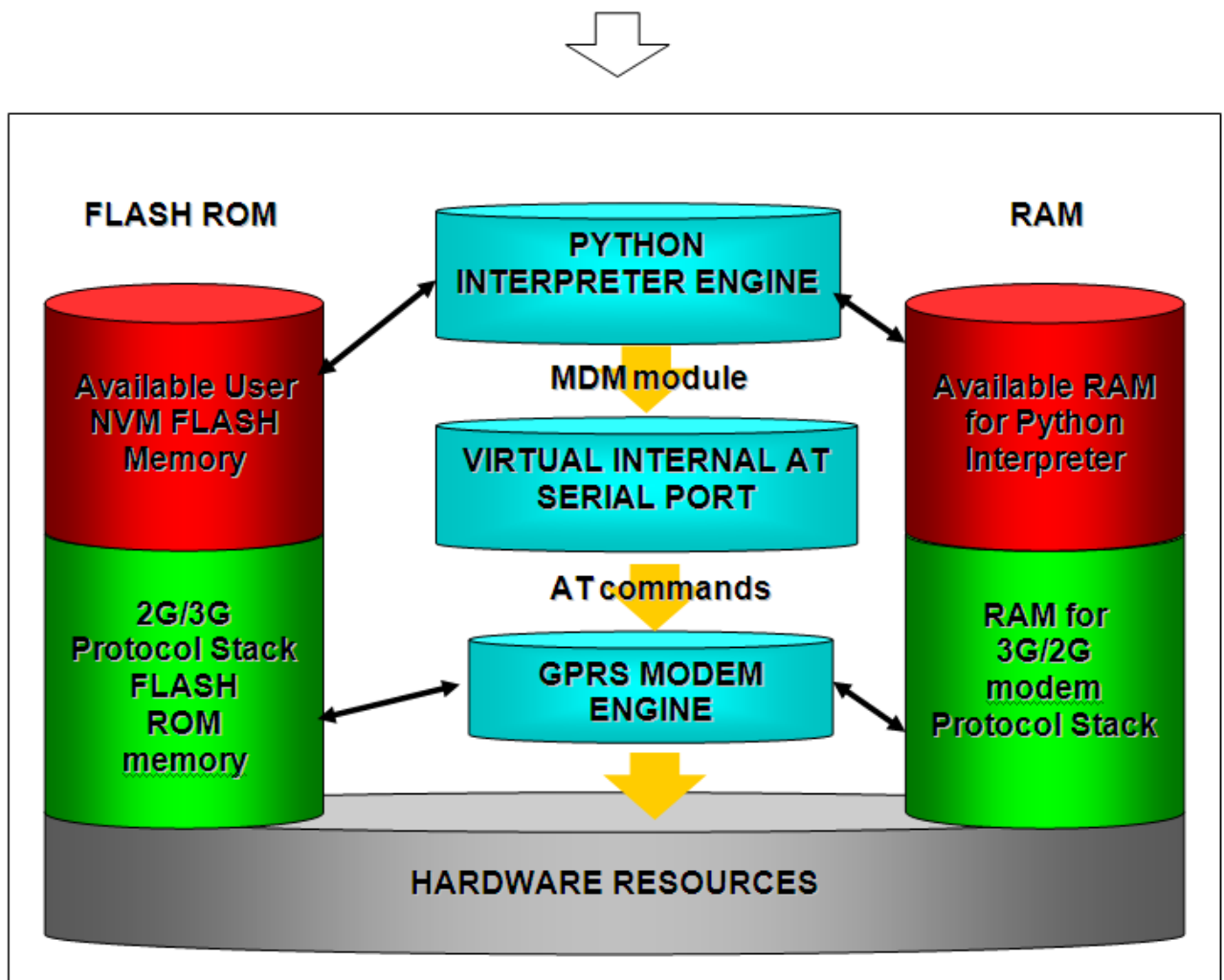
A schematic of such a configuration can be the following:



The Easy Script Extension functionality lets the developer to get rid of the external controller and further simplify the programmed sequence of operations. The equipped Python version features the following:

- Python script interpreter engine version 2.7.2
- 2 MB of Non Volatile Memory space for user scripts and data files
- 2 MB RAM available for the Python engine

The following depicts a schematic of this approach:



2.2. Python 2.7.2 Copyright Notice

Copyright (c) 2001-2011 Python Software Foundation.
All Rights Reserved.
Copyright (c) 2000 BeOpen.com.
All Rights Reserved.
Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.
Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.

Please refer to <http://www.python.org/doc/copyright/>

2.3. Python

Python is a dynamic object-oriented multipurpose high level programming language.
Python interpreter implemented version is 2.7.2.

Refer to

<http://www.python.org/>

and

<http://www.python.org/download/releases/2.7.2/>

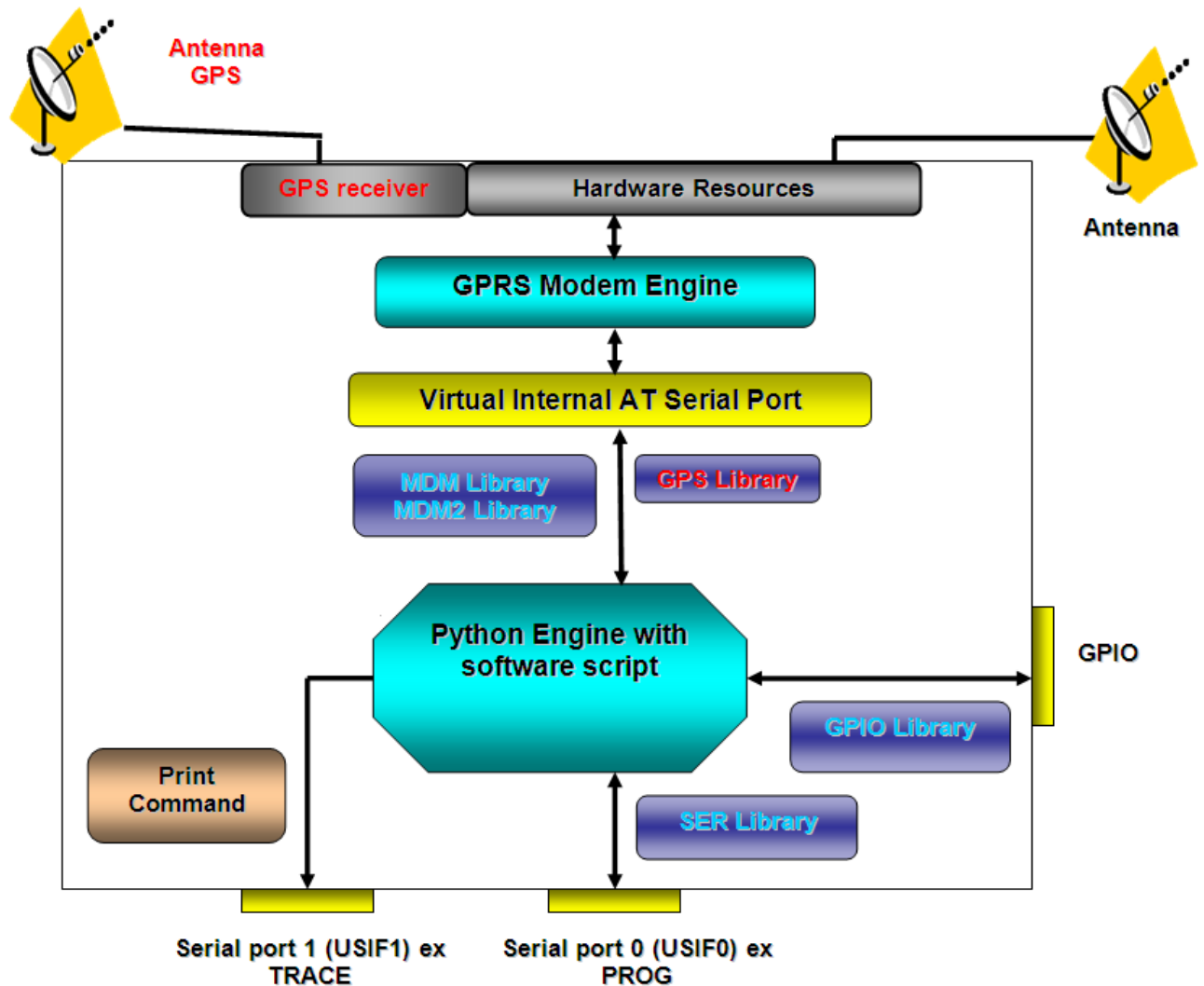
for any information about Python and Python interpreter version 2.7.2.

2.4. Python implementation description

Python scripts are text files stored in the **Telit module NVM** (Non Volatile Memory). There's a file system inside the module that allows to write and read text and binary files with different names on one single level (no subdirectories are supported).



The Python script is executed in a task with the lowest priority on the **Telit module**, so its execution won't interfere with GSM/GPRS/UMTS normal operations. Furthermore, this allows serial ports, protocol stack etc. to run independently from the Python script. The Python script interacts with the **Telit module** functionalities through several built-in interfaces, as depicted below:



NOTE:

Antenna GPS, GPS receiver and GPS Library are available exclusively for the GPS modules.

- **The MDM interface** is the most important one. It allows the Python script to send AT commands, receive responses and unsolicited indications, send data to the network and receive data from the network during connections. It is quite similar to the regular serial port interface on the **Telit module**. The only difference being that this interface is an internal software bridge between Python and module internal AT



command handling engine, and not a physical serial port. All AT commands working on the **Telit module** are working with this software interface as well. Some of them have no meaning for this interface, such as those regarding serial port settings, while others, such as the concept of hardware flow control, keeps its meaning but it's managed internally.

- **The MDM2 interface** is the second interface between Python and the module internal AT command handling. Its purpose is to send AT commands from the Python script to the module and receive AT responses from the module to the Python script when the regular MDM built-in module is already in use.
- **The SER interface** lets the Python script to read from and write to the physical serial port USIF0, usually the default port to send AT commands to the module (e.g.: to read information from an external device). When Python is running, this serial port is free to be used by the Python script since it is not used as the AT command interface; the AT parser, in fact, is mapped into the internal virtual serial port. No flow control is available from Python on this port.
- **The GPIO interface** lets the Python script to handle general purpose input output faster than through AT commands, skipping the command parser and controlling directly the pins.
- **The GPS interface** is the interface between Python and the module's internal GPS controller. Its purpose is to handle the GPS controller without the use of dedicated AT commands through the MDM built-in module.
- **The Python print statement**, for debugging purposes, is directly forwarded to second serial port USIF1.

2.5. Python supported features

Python implemented version is a smaller part than the original: core Python interpreter is mostly supported but only a few Python modules are supported.

The core Python interpreter version is 2.7.2. All Python statements and almost all Python built-in types and functions are available for development. Refer to chapter PYTHON STANDARD FUNCTIONS for more details.



3. Python Script Operations

3.1. Executing the Python script

The steps required to have a script executing by the Python engine on the **Telit module** are:

- write the Python script, in case splitting it in more than one file;
- optionally compile the Python script;
- download the, optionally compiled, Python script into the module NVM;
- enable the Python script;
- run the Python script.

3.1.1. Write the Python script

A Python script is a simple text file with .py extension, it can be written with any text editor. In case of large application it is useful to split it in more than one file.

The following is the "Hello Word" short Python script example that sends the simplest AT command to the AT command parser, immediately reads response and then ends.

```
import MDM
print 'Hello World!'
result = MDM.send('AT\r', 0)
print result
c = MDM.read()
print c
```

3.1.2. Compile the Python script

It is an optional operation.

Compiling the Python script on PC before downloading to module saves time at Python script execution start.

The following procedure allows to compile .py Python files into .pyc Python compiled files:

- install Python version 2.7.2 on your PC (as an example in directory C:\Python27)
- use compileall.py library Python script on your PC to compile all .py files in your working directory (as an example in directory C:\pytemp)

```
cd C:\Python27
python -v -S .\Lib\compileall.py -l -f C:\pytemp
```

3.1.3. Download the Python script

Use the following AT command:

AT#WSCRIPT="<script_name>",<size>[,<know-how>]

where

<script_name>: file name

<size>: file size (number of bytes)



<know-how>: (optional) know how protection, 1 = on, 0 = off (default)

The script, the compiled script, any text or binary file, can be downloaded to the module using the AT#WSCRIPT command. In order to guarantee your company know-how, you have the option to hide the script text so that the AT#RSCRIPT command does not return the text of the script and keeps it "confidential", you can see the name of the script only using the AT#LSCRIPT command.

In order to download the, optionally compiled, Python script you have to choose a name for your script on the module, taking care of the following:

- the extension for scripts is .py;
- the extension for compiled scripts is .pyc;
- any or no extension is permitted for generic text or binary file;
- the maximum file name length allowed is 16 characters;
- file names are case sensitive.

Then you have to find out the exact size in bytes of the script or compiled script, or generic text or binary file. For example, right clicking on the file and selecting "size" in "properties" (attention: this is different from selecting "size on the disc").

It is important for large files, compared to module serial port buffer size of 4096 bytes, to activate hardware flow control on your terminal emulator.

It is possible to overwrite an existing file, there is no need to delete old one first.

When using standard Windows terminal emulator *Hyper Terminal* consider that in some unexpected conditions cuts or changes part of downloaded file.

In *Hyper Terminal* application select "Hardware" flow control in serial settings.

In ASCII Setup set "Send line ends with line feeds" and "Append line feeds to incoming line ends".

Type for example

```
AT#WSCRIPT="a.py",110
```

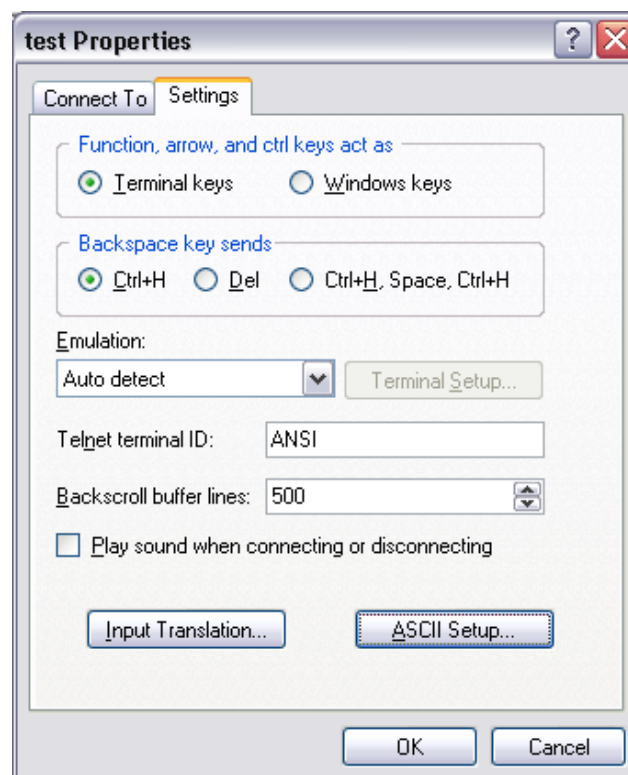
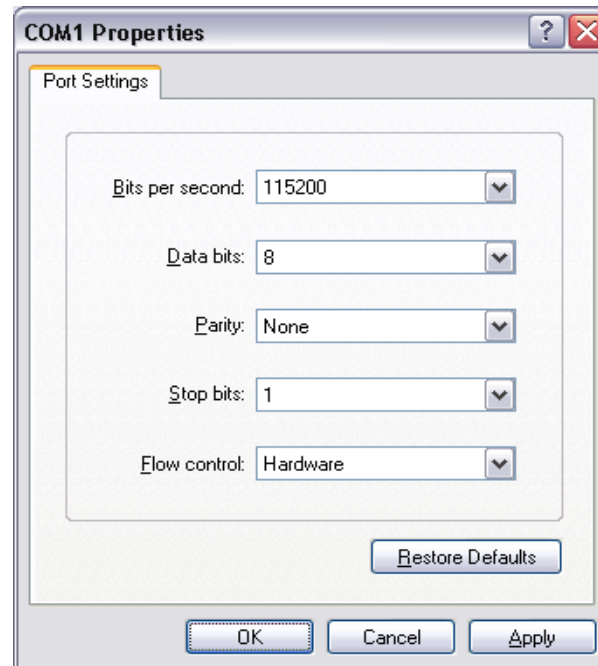
wait for the prompt

```
>>>
```

and use "Send Text file" selecting the proper file.

Wait for the result: OK or ERROR.







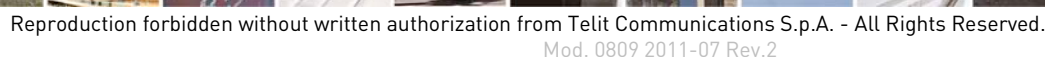
AT#ESCRIP="<script_name>"

<script name>: script name to be executed

When selecting the script to enable between the ones downloaded to the module:
 AT#LSCRIPT - can help checking the names of the scripts;
 AT#ESCRIP? - can help checking the name of the script that is enabled at the moment.



There is no error return value for non existing script name in the module memory typed in command AT#ESCRPT. For this reason it's recommended to double check the name of the script that you want to execute. On the other hand this characteristic permits additional possibilities like enabling the Python script before downloading it on the module or not having to enable the same script name every time the script has been changed, deleted and replaced with another script but with the same name.



Type for example

`AT#ESCRIP="a.py"`

Wait for the result: OK or ERROR.

3.1.5. Run the Python script

In default configuration the Python script downloaded to module and enabled is executed at every module power on if the DTR line is sensed LOW (2.8V at the module DTR pin - RS232 signals are inverted) on USIF0 at start-up (this means that no AT command interface is connected to the modem serial port), and if the script name enabled matches with one of the script names of the scripts downloaded.

For example the Python script correctly downloaded and enabled is executed when the module is powered on and the serial cable was previously disconnected from USIF0.

In order to block Python script execution and control the module through the AT command interface on the serial port (for example to update locally a new script) the module shall be powered on with the DTR line HIGH (0V at the module DTR pin). In this condition the Python engine is not started and the script is not executed.

The real execution of the .py Python script is delayed from the power on due to the time needed by Python to parse the script. The larger is the script, the longer is this delay. The execution of .pyc compiled Python script is faster because there is no parsing of the script, just reading the file from NVM.

Another possibility is to run the correctly downloaded and enabled Python script from terminal emulator using the following AT command:

`AT#EXECSCR`

Another possibility is to select a second way of executing the Python script at every module power on using the following AT command:

`AT#STARTMODESCR=<script_start_mode>[,<script_start_to>]`

where

`<script_start_mode>`: mode (default 0)

`<script_start_to>`: timeout (default 10)

If the mode is set to 1 than the Python script downloaded to module and enabled is executed at every module power on if the user does not send any AT command on the serial port for the time interval specified in `<script_start_to>` parameter (default 10s).

3.1.6. Read the Python script

Use the following AT command:

`AT#RSCRIPT="<script_name>"`

where

`<script_name>`: file name

Using the command `AT#RSCRIPT` read a saved script, compiled script, generic text or binary file. If know-how protection is activated than `AT#RSCRIPT` will return only OK: no Python



script source code will be returned. In this way no one will be able to read your Python script from the module serial port.

It is important for large files, compared to PC serial port buffer size, to activate hardware flow control on your terminal emulator.

Type for example

```
AT#RSCRIPT="a.py
```

Wait for the prompt

```
<<<
```

Receive file data and wait for the result: OK or ERROR.

3.1.7. List saved Python scripts

Use the following AT command:

```
AT#LSCRIPT
```

This command shows the list of the file names currently saved, their size and the number of free bytes in memory.

3.1.8. Delete the Python script

Use the following AT command:

```
AT#DSCRIPT="<script_name>"
```

where

```
<script_name>: file name
```

Using the AT#DSCRIPT command delete from the module memory a saved script, compiled script, generic text or binary file.

Type for example

```
AT#DSCRIPT="a.py"
```

Wait for the result: OK or ERROR.



3.2. Debug Python script

The debug of the running Python script can be done on the second serial port USIF1 of the module.

Use the following AT command:

```
AT#PORTCFG=3
```

to configure ports properly.

Reboot module to make #PORTCFG configuration working.

In #PORTCFG: 3 configuration Python standard output and standard error, including print statements, are redirected to USIF1 at 115200.

In this configuration AT2 parser instance on USIF1 is not available.

In this configuration Python scripts can be debugged with or without USB inserted.

Connect to the second module serial port USIF1 at 115200.

Collect Python standard output and standard error:

- Python information messages (for example the version);
- Python error information;
- Results of all Python “print” statements.



4. Python Build-in Custom Modules

Several built-in custom modules have been included in the Python core, specifically developed keeping in mind the hardware environment of the module.

The built-in modules included are:

| | |
|-------------|---|
| MDM | interface between Python and the module AT command handling |
| MDM2 | second interface between Python and the module AT command handling |
| SER | interface between Python and the module serial port USIF0 direct handling |
| GPIO | interface between Python and the module internal general purpose input output direct handling |
| GPS | interface between Python and the module internal GPS controller |

4.1. MDM built-in module

The MDM built-in module is the interface between Python and the module AT commands parser engine.

You need to use the MDM built-in module if you want to send AT commands and data from the Python script to the network and receive responses and data from the network during connections.

In the default configuration, echo (ATE0) is disabled and the response format of result codes is set to verbose (ATV1).

If you want to use this module you need to *import* it first:

```
import MDM
```

then you can use MDM built-in module methods as in the following example:

```
a = MDM.send('AT', 0)
b = MDM.sendbyte(0x0d, 0)
c = MDM.read()
```

which sends 'AT' and reads 'OK' response.

More details about MDM built-in module methods can be found in the following paragraphs.

4.1.1. MDM.send(string, timeout)

This command sends a string to the AT command interface.

The first input parameter *string* is a Python string to send to the AT command interface.

The second input parameter *timeout* is a Python integer, measured in 1/10s, and is important in online mode when flow control is activated. In fact it represents the maximum time to wait for the string to be sent to the AT command interface when buffer is full and flow control blocks further data. The timeout range is (0 ÷ 32767).



This method returns immediately after the string has been sent to the AT interface or after the timeout period if the whole string could not be sent to the AT interface. The return value is a Python integer which is -1 if the timeout period has expired, 1 otherwise.

Example:

```
a = MDM.send('AT', 5)
```

sends the string 'AT' to AT command handling, waiting up to 0.5 s, assigning the return value to a.



NOTE:

The buffer available for the MDM.send command is 4096 bytes.

4.1.2. MDM.read()

This command receives a string from the AT command interface.
It has no input parameter.

The return value is a Python string which contains the data received and stored in buffer at the moment of command execution. The value might be empty if no data is received.

Example:

```
a = MDM.read()
```

Receives a string from AT command handling, assigning the return value to a.



NOTE:

The buffer available for MDM.read command is 4096 bytes.



NOTE:

It is up to Python script to keep empty MDM.read buffer.

4.1.3. MDM.sendbyte(byte, timeout)

This command sends one byte to the AT command interface.

The first input parameter *byte* can be zero or any Python byte to send to the AT command interface.

The second input parameter *timeout* is a Python integer, measured in 1/10s, and is important in online mode when flow control is activated. In fact it represents the maximum time to wait for the byte to be sent to the AT command interface when buffer is full and flow control blocks further data. The timeout range is (0 ÷ 32767).



This method returns immediately after the byte has been sent to the AT interface or after the timeout period if the byte could not be sent to the AT interface. The return value is a Python integer which is -1 if the timeout expired, 1 otherwise.

Example:

```
b = MDM.sendbyte(0x0d, 0)
```

Sends the byte 0x0d (carriage return <CR>) to the AT commands handling, without waiting and assigning the return value to b.

4.1.4. **MDM.readbyte()**

This command receives a byte from the AT command interface.

It has no input parameter.

The return value is a Python integer which is the byte value received and stored in buffer at the moment of command execution or is -1 if no data is received. The return value can also be zero.

Example:

```
b = MDM.readbyte()
```

receives a byte from AT command handling, assigning the return value to b.

4.1.5. **MDM.sendavail()**

This command queries the number of bytes available to send to MDM buffer.

It has no input parameter.

The return value is a Python integer which is the number of bytes available to send to MDM buffer.

Example:

```
n = MDM.sendavail()
```

queries the number of bytes available to send, assigning the return value to n.

4.1.6. **MDM.getDCD()**

This command gets Carrier Detect (DCD) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is either 0 if DCD is OFF or 1 if DCD is ON.

Example:

```
cd = MDM.getDCD()
```



gets DCD from AT command handling, assigning the return value to cd.

4.1.7. **MDM.getCTS()**

This command gets Clear to Send (CTS) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is either 0 if CTS is set to OFF or 1 if CTS is set to ON.

Example:

```
cts = MDM.getCTS()
```

gets CTS from AT command handling, assigning the return value to cts.

4.1.8. **MDM.getDSR()**

This command gets Data Set Ready (DSR) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is either 0 if DSR is OFF or 1 if DSR is ON.

Example:

```
dsr = MDM.getDSR()
```

gets DSR from AT command handling, assigning the return value to dsr.

4.1.9. **MDM.getRI()**

This command gets Ring Indicator (RI) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is either 0 if RI is set to OFF or 1 if RI is set to ON.

Example:

```
ri = MDM.getRI()
```

gets RI from AT command handling, assigning the return value to ri.

4.1.10. **MDM.setRTS(RTS_value)**

This command sets Request to Send (RTS) in the AT command interface.

The input parameter *RTS_value* is a Python integer which is either 0 if setting RTS to OFF or 1 if setting RTS to ON.

No return value.

Example:



```
MDM.setRTS(1)
```

sets RTS to ON in AT command handling.

4.1.11. **MDM.setDTR(DTR_value)**

This command sets Data Terminal Ready (DTR) in the AT command interface.

The input parameter *DTR_value* is a Python integer which is either 0 if setting DTR to OFF or 1 if setting DTR to ON.

No return value.

Example:

```
MDM.setDTR(0)
```

sets DTR to OFF in AT command handling.



4.2. MDM2 built-in module

MDM2 built-in module is the second interface between Python and the module internal AT command handling. It is used to send AT commands from Python script to module and receive AT responses from module to Python script when the classic MDM built-in module is already in use.

Though MDM2 built-in module is independent from activation of CMUX protocol, it works on the second instance of AT parser in the same way the second CMUX port does. So the rules on AT commands that apply on the first and second CMUX ports (AT parser instances) apply on MDM and MDM2 as well.

See "AT Commands Reference Guide" and "CMUX User Guide" for details on availability of AT commands on all instances and for the rules on parallel execution of AT commands on two instances.

In the default configuration, echo (ATE0) is disabled and the response format of result codes is set to verbose (ATV1).

If you want to use MDM2 built-in module you need to import it first:

```
import MDM2
```

than you can use MDM2 built-in module methods like in the following example:

```
a = MDM2.send('AT', 0)
b = MDM2.sendbyte(0x0d, 0)
c = MDM2.read()
```

which sends 'AT' and reads 'OK' response.

More details about MDM2 built-in module methods can be found in the following paragraphs.

4.2.1. MDM2.send(string, timeout)

This command sends a string to the AT command interface.

The first input parameter *string* is a Python string to send to the AT command interface.

The second input parameter *timeout* is a Python integer, measured in 1/10s, and is important in online mode when flow control is activated. In fact it represents the maximum time to wait for the string to be sent to the AT command interface when buffer is full and flow control blocks further data. The timeout range is (0 ÷ 32767).

This method returns immediately after the string has been sent to the AT interface or after the timeout period if the whole string could not be sent to the AT interface. The return value is a Python integer which is -1 if the timeout period has expired, 1 otherwise.

Example:

```
a = MDM2.send('AT', 5)
```

sends string 'AT' to AT command handling, possibly waiting for 0.5 s, assigning the return value to a.





NOTE:

The buffer available for MDM2.send command is 4096 bytes.

4.2.2. MDM2.read()

This command receives a string from the AT command interface.

It has no input parameter.

The return value is a Python string which contains the data received and stored in buffer at the moment of command execution. The value might be empty if no data is received.

Example:

```
a = MDM2.read()
```

receives a string from AT command handling, assigning the return value to a.



NOTE:

The buffer available for MDM2.read command is 4096 bytes.



NOTE:

It is up to Python script to keep empty MDM2.read buffer.

4.2.3. MDM2.sendbyte(byte, timeout)

This command sends a byte to the AT command interface.

The first input parameter *byte* can be zero or any Python byte to send to the AT command interface.

The second input parameter *timeout* is a Python integer, measured in 1/10s, and is important in online mode when flow control is activated. In fact it represents the maximum time to wait for the byte to be sent to the AT command interface when buffer is full and flow control blocks further data. The timeout range is (0 ÷ 32767).

This method returns immediately after the byte has been sent to the AT interface or after the timeout period if the byte could not be sent to the AT interface. The return value is a Python integer which is -1 if the timeout expired, 1 otherwise.

Example:

```
b = MDM2.sendbyte(0x0d, 0)
```

sends byte 0x0d, that is <CR>, to AT command handling, without waiting, assigning the return value to b.



4.2.4. **MDM2.readbyte()**

This command receives a byte from the AT command interface.

It has no input parameter.

The return value is a Python integer which is the byte value received and stored in buffer at the moment of command execution or is -1 if no data is received. The return value can also be zero.

Example:

```
b = MDM2.readbyte()
```

receives a byte from AT command handling, assigning the return value to b.

4.2.5. **MDM2.sendavail()**

This command queries the number of bytes available to send to MDM2 buffer.

It has no input parameter.

The return value is a Python integer which is the number of bytes available to send to MDM2 buffer.

Example:

```
n = MDM2.sendavail()
```

queries the number of bytes available to send, assigning the return value to n.

4.2.6. **MDM2.getDCD()**

This command gets Carrier Detect (DCD) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is 0 if DCD is set to OFF or 1 if DCD is set to ON.

Example:

```
cd = MDM2.getDCD()
```

gets DCD from AT command handling, assigning the return value to cd.

4.2.7. **MDM2.getCTS()**

This command gets Clear to Send (CTS) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is either 0 if CTS is set to OFF or 1 if CTS is set to ON.

Example:



```
cts = MDM2.getCTS()
```

gets CTS from AT command handling, assigning the return value to cts.

4.2.8. **MDM2.getDSR()**

This command gets Data Set Ready (DSR) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is either 0 if DSR is set to OFF or 1 if DSR is set to ON.

Example:

```
dsr = MDM2.getDSR()
```

gets DSR from AT command handling, assigning the return value to dsr.

4.2.9. **MDM2.getRI()**

This command gets Ring Indicator (RI) from the AT command interface.

It has no input parameter.

The return value is a Python integer which is 0 if RI is set to OFF or 1 if RI is set to ON.

Example:

```
ri = MDM2.getRI()
```

gets RI from AT command handling, assigning the return value to ri.

4.2.10. **MDM2.setRTS(RTS_value)**

This command sets Request to Send (RTS) in the AT command interface.

The input parameter *RTS_value* is a Python integer which is 0 if setting RTS to set to OFF or 1 if setting RTS to set to ON.

No return value.

Example:

```
MDM2.setRTS(1)
```

sets RTS to ON in AT command handling.

4.2.11. **MDM2.setDTR(DTR_value)**

This command sets Data Terminal Ready (DTR) in the AT command interface.

The input parameter *DTR_value* is a Python integer which is 0 if setting DTR to set to OFF or 1 if setting DTR to set to ON.



No return value.

Example:

```
MDM2.setDTR(0)
```

sets DTR to OFF in AT command handling.



4.3. SER built-in module

The SER built-in module is an interface between the Python core and the device serial port over the RXD/TXD pins direct handling. You need to use the SER built-in module if you want to send data from the Python script to the serial port and to receive data from the serial port USIF0 to the Python script. This serial port handling module can be used, for example, to interface the module with an external device (such as a GPS) and read/send its data (e.g. NMEA). The SER built-in module allows to control physical lines used as GPIO.

If you want to use SER built-in module you need to import it:

```
import SER
```

then you can use its methods, like in the following example:

```
a = SER.set_speed('9600')
b = SER.send('test')
c = SER.sendbyte(0x0d)
d = SER.read()
```

which sends 'test' followed by <CR> and receives data.

More details about SER built-in module methods can be found in the following paragraphs.

4.3.1. SER.send(string)

This command sends a string to the serial port TXD/RXD.
The input parameter *string* is a Python string to send to the serial port USIF0.
The return value is a Python integer which is -1 if an error occurred otherwise is 1.

Example:

```
a = SER.send('test')
```

sends the string 'test' to the serial port USIF0 handling, assigning the return value to a.



NOTE:

the buffer available for the SER.send command is 4096 bytes.

4.3.2. SER.read()

This command receives a string from the serial port TXD/RXD.
It has no input parameter.
The return value is a Python string which contains the data received and stored in buffer at the moment of command execution. The value might be empty if no data is received.



Example:

```
a = SER.read()
```

receives a string from the serial port USIF0 handling, assigning the return value to a.



NOTE:

the buffer available for the SER.read command is 4096 bytes.



NOTE:

It is up to Python script to keep empty SER.read buffer.

4.3.3. **SER.sendbyte(byte)**

This command sends a byte to the serial port TXD/RXD.

The input parameter *byte* can be zero or any Python byte to send to the serial port

The return value is a Python integer which is -1 if an error occurred otherwise is 1.

Example:

```
b = SER.sendbyte(0x0d)
```

sends the byte 0x0d, that corresponds to <CR>, to the serial port USIF0 handling, assigning the return value to b.

4.3.4. **SER.readbyte()**

This command receives a byte from the serial port TXD/RXD.

It has no input parameter.

The return value is a Python integer which is the byte value received and stored in buffer at the moment of command execution or is -1 if no data is received. The return value can also be zero.

Example:

```
b = SER.readbyte()
```

receives a byte from serial port USIF0 handling, assigning the return value to b.

4.3.5. **SER.sendavail()**

This command queries the number of bytes available to send to SER buffer.

It has no input parameter.

The return value is a Python integer which is the number of bytes available to send to SER buffer.



Example:

```
n = SER.sendavail()
```

queries the number of bytes available to send, assigning the return value to n.

4.3.6. **SER.set_speed(speed, <char format>)**

This command sets the serial port TXD/RXD speed. The default serial port TXD/RXD speed is 115200.

The first input parameter *speed* is a Python string which is the value of the serial port speed. It can assume the same values as the +IPR command.

The second optional parameter <char format> is a Python string that represents the character format to be used:

the first character is the number of bits per char (7 or 8), then the parity setting (N - none, E- even, O- odd) and in the end the number of stop bits (1 or 2). The default value is "8N1".

The return value is a Python integer which is -1 if an error occurred otherwise is 1.

Example:

```
b = SER.set_speed('9600')
```

sets the serial port USIF0 speed to 9600, assigning the return value to b.



NOTE:

sending the +IPR command to the device does not affect the physical serial port, you must use this function to set the speed of the port when using the Python engine.

4.3.7. **SER.setDCD(DCD_value)**

This command sets Carrier Detect (DCD) on the serial port USIF0.

The input parameter *DCD_value* is a Python integer which is either 0 if DCD is set to OFF or 1 if DCD is set to ON.

No return value.

Example:

```
SER.setDCD(1)
```

sets DCD to ON in USIF0.

4.3.8. **SER.setCTS(CTS_value)**

This command sets Clear to Send (CTS) on the serial port USIF0.



The input parameter *CTS_value* is a Python integer which is either 0 if CTS is set to OFF or 1 if CTS is set to ON.

No return value.

Example:

```
SER.setCTS(1)
```

sets CTS to ON in USIF0.

4.3.9. **SER.setDSR(DSR_value)**

This command sets Data Set Ready (DSR) on the serial port USIF0.

The input parameter *DSR_value* is a Python integer which is either 0 if DSR is set to OFF or 1 if DSR is set to ON.

No return value.

Example:

```
SER.setDSR(1)
```

sets DSR to ON in USIF0.

4.3.10. **SER.setRI(RI_value)**

This command sets Ring Indicator (RI) on the serial port USIF0.

The input parameter *RI_value* is a Python integer which is either 0 if RI is set to OFF or 1 if RI is set to ON.

No return value.

Example:

```
SER.setRI(1)
```

sets RI to ON in USIF0.

4.3.11. **SER.getRTS()**

This command gets Request to Send (RTS) from the serial port USIF0.

It has no input parameter.

The return value is a Python integer which is either 0 if RTS is set to OFF or 1 if RTS is set to ON.

Example:

```
rts = SER.getRTS()
```

gets RTS from USIF0, assigning the return value to rts.



4.3.12. **SER.getDTR()**

This command gets Data Terminal Ready (DTR) from the serial port USIF0.

It has no input parameter.

The return value is a Python integer which is either 0 if DTR is set to OFF or 1 if DTR is set to ON.

Example:

```
dtr = SER.getDTR()
```

gets DTR from USIF0, assigning the return value to dtr.



4.4. GPIO built-in module

The GPIO built-in module is an interface between the Python core and the module internal general purpose input output direct handling. The GPIO built-in module is used to set GPIO values and to read GPIO values from the Python script. You can control the GPIO pins also by sending internal 'AT#GPIO' commands using the MDM module, but using the GPIO module is faster because no command parsing is involved, therefore its use is recommended.



NOTE:

The Python core does not verify if the pins are already used for other purposes by other functions, it's the customer responsibility to ensure that no conflict over pins occurs.

If you want to use the GPIO built-in module you need to import it first:

```
import GPIO
```

then you can use its methods as in the following example:

```
a = GPIO.getIOvalue(5)
b = GPIO.setIOvalue(4, 1)
```

this reads the GPIO 5 value and sets GPIO 4 to the output with value 1.

More details about GPIO built-in module methods are in the following paragraphs.

4.4.1. GPIO.setIOvalue(GPIOnumber, value)

This method sets the output value of a GPIO pin.

The first input parameter *GPIOnumber* is a Python integer which is the number of the GPIO. The second input parameter *value* is a Python integer which is the output value. It can be 0 or 1.

The return value is a Python integer which is -1 if an error occurred otherwise is 1.

Example:

```
b = GPIO.setIOvalue(4, 1)
```

sets GPIO 4 to output with value 1, assigning the return value to b.

4.4.2. GPIO.getIOvalue(GPIOnumber)

This method gets the input value of a GPIO.

The input parameter *GPIOnumber* is a Python integer which is the number of the GPIO. The return value is a Python integer which is -1 if an error occurred otherwise it is the input value. It can be either 0 or 1.

Example:



gets the GPIO 5 input value, assigning the return value to b.

This method sets the direction of a GPIO.

The first input parameter *GPIOnumber* is a Python integer which is the number of the GPIO. The second input parameter *value* is a Python integer which is the output value. It can be either 0 or 1. It is only used if the *direction* value is 1, it has no meaning if the *direction* value is 0.

The third input parameter *direction* is a Python integer which is the direction value. It can be either 0 for input or 1 for output or 2 for alternate function or 3 for tristate pull down ??????????????. Refer to AT#GPIO command in "AT Commands Reference Guide" for further notes.

The return value is a Python integer which is -1 if an error occurred otherwise is 1.

Example:

sets GPIO 4 to input with the *value* parameter having no meaning, and assigning the return value to c.



NOTE:

when the *direction* value is not 1, although the parameter *value* has no meaning, it is necessary to assign it one of the two possible values: 0 or 1

This method gets the direction of a GPIO.

This method gets the direction of a GPIO.

The input parameter *GPIOnumber* is a Python integer which is the number of the GPIO.

The return value is a Python integer which is -1 if an error occurred otherwise is direction value. It is 0 for input or 1 for output or 2 for alternate function or 3 for tristate pull down.

Example:

gets GPIO 7 direction, assigning the return value to d.

This method gets ADC value. It is equivalent to the AT#ADC command.

The input parameter *adcNumber* is a Python integer which represents the ADC number that will be read and converted in voltage.



The return value is a Python integer which is -1 if an error occurred otherwise the converted voltage is returned in mV.

Example:

```
mV = GPIO.getADC(2)
```

gets ADC number 2 input voltage, assigning the return value in mV.

4.4.6. **GPIO.setDAC(enable, value)**

This method sets the DAC value. It is equivalent to the AT#DAC command.

The first input parameter *enable* is a Python integer and can assume values 0 or 1. If it is set to 1 enables DAC output otherwise if it is set to 0 disabled DAC output.

The second input parameter *value* is a Python integer and represents the scale factor of output voltage and can assume values in the range 0-1023.

The return value is a Python integer that has value -1 if an error occurred otherwise it has value 1.

Example:

```
res = GPIO.setDAC(1, 512)
```

sets DAC output voltage at half the range, assigning the return value to res.

4.4.7. **GPIO.setVAUX(vauxNumber, enable)**

This method enables or disables the VAUX. It is equivalent to the AT#VAUX command.

The first input parameter *vauxNumber* is a Python integer that represents VUAX number that will be enabled or disabled.

The second input parameter *enable* is a Python integer that can assume value 1 in order to enable VAUX output or 0 if VAUX output should be disabled.

The return value is a Python integer that has value -1 if an error occurred otherwise it has the value 1.

Example:

```
res = GPIO.setVAUX(1, 1)
```

enables VAUX number 1 output, assigning the return value to res.

4.4.8. **GPIO.getAXE()**

This method gets the hands free status value. It is equivalent to the AT#AXE command.

It has no input parameter.

The return value is a Python integer that is either 0 if a hand free is not connected or 1 if a hand free is connected.



Example:

```
hf = GPIO.getAXE()
```

gets the AXE value, assigning the return value to hf.

4.4.9. **GPIO.setSLED(status, onDuration, offDuration)**

This method sets the status led configuration values. It is equivalent to the AT#SLED command.

The first input parameter *status* is a Python integer that represents the configuration of status led and can assume the following values:

- 0 - ALWAYS OFF
- 1 - ALWAYS ON
- 2 - AUTO
- 3 - BLINKING

The second input parameter *onDuration* is a Python integer which is the period of ON configuration of status led measured in 1/10s.

The third input parameter *offDuration* is a Python integer which is the period of OFF configuration of status led measured in 1/10s.

The return value is a Python integer which is -1 if an error occurred otherwise it is 1.

Example:

```
res = GPIO.setSLED(3, 10, 90)
```

sets status led configuration to blinking with 1s in ON period and 9s in OFF period, assigning the return value to res.

4.4.10. **GPIO.getCBC()**

This method gets the charger status and battery voltage. It is equivalent to the AT#CBC command.

It has no input parameters.

The return value is a Python tuple formatted in the following way:

```
(chargerStatus, batteryVoltage).
```

First element of tuple is a Python integer which is charger status:

- 0 - charger not connected
- 1 - charger connected and charging
- 2 - charger connected and charging process completed

Second element of tuple is a Python integer which is battery voltage in mV.

Example:

```
cbc = GPIO.getCBC()
```

gets charger status and battery voltage values, assigning the return value to cbc tuple.



4.5. GPS built-in module

GPS built-in module is the interface between Python and module internal GPS controller. It is used in order to handle GPS controller without dedicated AT commands through MDM built-in module.

If you want to use GPS built-in module you need to import it first:

```
import GPS
```

After this you can start using GPS built-in module methods like in the following example:

```
position = GPS.getActualPosition()
```

gets a string with position information formatted in the same way as AT\$GPSACP response. More details about GPS built-in module methods can be found in the following paragraphs.

4.5.1. GPS.powerOnOff(newStatus)

This method powers ON/OFF GPS controller. It is equivalent to the AT\$GPSP command. The input parameter *newStatus* is a Python integer and can have the following values:

0 - to power OFF GPS controller

1 - to power ON GPS controller.

There is no return value.

Example:

```
GPS.powerOnOff(0)
```

GPS controller is powered OFF.

4.5.2. GPS.getPowerOnOff()

This method gets GPS controller current power ON/OFF status.

It has no input parameter.

The return value is a Python integer which is 0 if GPS controller is powered off or 1 if GPS controller is powered on.

Example:

```
status = GPS.getPowerOnOff()
```

gets GPS controller current power ON/OFF status, assigning the return value to status.

4.5.3. GPS.getActualPosition()

This method gets GPS last position information. It is equivalent to the AT\$GPSACP command.



It has no input parameter.

The return value is a Python string which is the last position information formatted in the same way as for AT\$GPSACP command response.

Example:

```
lastPosition = GPS.getActualPosition()
```

gets GPS last position information, assigning the return value to lastPosition.

4.5.4. **GPS.getLastGGA()**

This method gets GPS last GGA NMEA sentence stored.

It has no input parameter.

The return value is a Python string which is the last GGA NMEA sentence formatted according to NMEA specification.

Example:

```
gga = GPS.getLastGGA()
```

gets last GGA NMEA sentence, assigning the return value to gga.

4.5.5. **GPS.getLastGLL()**

This method gets GPS last GLL NMEA sentence stored.

It has no input parameter.

The return value is a Python string which is the last GLL NMEA sentence formatted according to NMEA specification.

Example:

```
gll = GPS.getLastGLL()
```

gets last GLL NMEA sentence, assigning the return value to gll.

4.5.6. **GPS.getLastGSA()**

This command gets GPS last GSA NMEA sentence stored.

It has no input parameter.

The return value is a Python string which is the last GSA NMEA sentence formatted according to NMEA specification.

Example:

```
gsa = GPS.getLastGSA()
```

gets last GSA NMEA sentence, assigning the return value to gsa.



4.5.7. **GPS.getLastGSV()**

This command gets GPS last GSV NMEA sentence stored.

It has no input parameter.

The return value is a Python string which is the concatenation of the last GSV NMEA sentences formatted according to NMEA specification.

Example:

```
gsv = GPS.getLastGSV()
```

gets last GSV NMEA sentence, assigning the return value to gsv.

4.5.8. **GPS.getLastRMC()**

This command gets GPS last RMC NMEA sentence stored.

It has no input parameter.

The return value is a Python string which is the last RMC NMEA sentence formatted according to NMEA specification.

Example:

```
rms = GPS.getLastRMC()
```

gets last RMC NMEA sentence, assigning the return value to rms.

4.5.9. **GPS.getLastVTG()**

This command gets GPS last VTG NMEA sentence stored.

It has no input parameter.

The return value is a Python string which is the last VTG NMEA sentence formatted according to NMEA specification.

Example:

```
vtg = GPS.getLastVTG()
```

gets last VTG NMEA sentence, assigning the return value to vtg.



5. Python standard functions

In this paragraph you can find detailed description of Python language supported features in [Telit module](#). Note that all the functions listed below are available in the Python version 2.7.2.

5.1. Technical characteristics

5.1.1. General

All Python statements and almost all Python built-in types and functions are supported. See in the table below the features not supported:

| |
|-----------|
| complex |
| unicode |
| docstring |
| packages |

Available standard built-in modules are:

| |
|-------------|
| marshal |
| imp |
| _ast |
| __main__ |
| __builtin__ |
| sys |
| exceptions |
| gc |
| _warnings |
| _md5 |
| binascii |
| _sre |
| _weakref |
| _symtable |
| _functools |
| _socket |
| time |
| posix |
| thread |
| signal |
| errno |
| cStringIO |

All others are not supported.

A small collection of standard Python modules written in Python (not built-in) is available. These .py files are mostly identical to the ones available for PC with minor changes.



5.2. Python supported features

Refer to the documents available online such as: Python 2.7.2 Tutorial, Python 2.7.2 Reference Manual or Python 2.7.2 Library Reference for details about all the features listed in the paragraphs below.

5.2.1. Operators, statements, functions

List of supported operators, statements, functions:

- comments #
- line joining \
- operators +, -, *, /, **, %
- operators <<, >>, &, |, ^, ~
- parentheses
- assignment
- comparison operators <, >, ==, <=, >=, !=, <>
- comparison operators in, not in
- print statement
- if, elif, else statement
- indentation
- and, or, not keywords
- for in statement
- while statement
- range() function
- break and continue statements
- pass statement
- functions (without docstrings) (def)
- return statement
- lambda forms
- objects
- object methods
- del statement
- modules
- import statement
- from statement
- exceptions
- try except finally statements
- raise statement
- classes (class)
- class instances
- global statement
- is, is not tests
- exec statement
- iterators



- generators
- yield statement
- with statement

5.2.2. Built-in Functions

The following built-in functions are supported:

| |
|----------------------------|
| abs |
| all |
| any |
| basestring |
| bin |
| bool |
| bytearray |
| callable |
| chr |
| classmethod |
| cmp |
| compile |
| complex (raises exception) |
| delattr |
| dict |
| dir |
| divmod |
| enumerate |
| eval |
| execfile |
| filter |
| float |
| format |
| frozenset |
| getattr |
| globals |
| hasattr |
| hash |
| help |
| hex |
| id |
| input |
| int |
| isinstance |
| issubclass |
| iter |
| len |
| list |
| locals |
| long |

| |
|------------|
| map |
| max |
| memoryview |
| min |
| next |
| object |



| |
|---------------|
| oct |
| open |
| ord |
| pow |
| print |
| property |
| range |
| raw_input |
| reduce |
| reload |
| repr |
| reversed |
| round |
| set |
| setattr |
| slice |
| sorted |
| staticmethod |
| str |
| sum |
| super |
| tuple |
| type |
| vars |
| xrange |
| zip |
| <u>import</u> |
| apply |
| buffer |
| coerce |
| intern |

5.2.3. Built-in Constants

The following built-in constants are supported:

| |
|----------------|
| False |
| True |
| None |
| NotImplemented |
| Ellipsis |
| <u>debug</u> |



5.2.4. Truth Value Testing

Truth value testing is supported.

5.2.5. Boolean Operations

The following Boolean operations are supported:

| |
|---------|
| x or y |
| x and y |
| not x |

5.2.6. Comparisons

The following comparisons are supported:

| |
|--------|
| < |
| <= |
| > |
| >= |
| == |
| != |
| is |
| is not |

5.2.7. Numeric Types: Integer, Long Integer and Floating Point

The following operations are supported with the integer, long integer and floating point type:

| |
|---------------|
| x + y |
| x - y |
| x * y |
| x / y |
| x // y |
| x % y |
| -x |
| +x |
| abs(x) |
| int(x) |
| long(x) |
| float(x) |
| divmod(x, y) |
| pow(x, y) |
| x ** y |
| round(x[, n]) |



5.2.8. Numeric Types: Integer and Long Integer

The following bit-string operations and methods are supported with the integer and long integer type:

| |
|---------------------------|
| <code>x y</code> |
| <code>x ^ y</code> |
| <code>x & y</code> |
| <code>x << y</code> |
| <code>x >> y</code> |
| <code>~x</code> |
| <code>bit_length()</code> |

5.2.9. Numeric Types: Floating Point

The following methods are supported with the floating point type:

| |
|---------------------------------|
| <code>as_integer_ratio()</code> |
| <code>is_integer()</code> |
| <code>hex()</code> |
| <code>fromhex(s)</code> |

5.2.10. Numeric Types: Complex

Complex numbers are not supported.

5.2.11. Iterator Types

The following methods are supported with the iterator type:

| |
|-------------------------|
| <code>__iter__()</code> |
| <code>next()</code> |

5.2.12. Generator Types

Generator types are supported.

5.2.13. Sequence Types: String, List, Tuple, Bytearray, Buffer and Xrange

The following operations are supported with the string, list, tuple, bytearray, buffer and xrange types:



| |
|---------------------------|
| <code>x in s</code> |
| <code>x not in s</code> |
| <code>s + t</code> |
| <code>s * n, n * s</code> |
| <code>s[i]</code> |
| <code>s[i:j]</code> |
| <code>s[i:j:k]</code> |
| <code>len(s)</code> |
| <code>min(s)</code> |
| <code>max(s)</code> |
| <code>s.index(i)</code> |
| <code>s.count(i)</code> |

Xrange type supports only indexing, iteration and len().

5.2.14. Sequence Types: Unicode

Unicode is not supported.

5.2.15. Sequence Types: String

The following methods are supported with the string types:

| |
|-------------------------|
| <code>capitalize</code> |
| <code>center</code> |
| <code>count</code> |
| <code>decode</code> |
| <code>encode</code> |
| <code>endswith</code> |
| <code>expandtabs</code> |
| <code>find</code> |
| <code>format</code> |
| <code>index</code> |
| <code>isalnum</code> |
| <code>isalpha</code> |
| <code>isdigit</code> |
| <code>islower</code> |
| <code>isspace</code> |
| <code>istitle</code> |
| <code>isupper</code> |
| <code>join</code> |
| <code>ljust</code> |

| |
|------------------------|
| <code>lower</code> |
| <code>lstrip</code> |
| <code>partition</code> |
| <code>replace</code> |
| <code>rfind</code> |
| <code>rindex</code> |



| |
|------------|
| rjust |
| rpartition |
| rsplit |
| rstrip |
| split |
| splitlines |
| startswith |
| strip |
| swapcase |
| title |
| translate |
| upper |
| zfill |

5.2.16. Mutable Sequence Types: List and bytearray

The following additional operations are supported with the lists and bytearray types:

| |
|--------------------------------|
| s[i] = x |
| s[i:j] = t |
| del s[i:j] |
| s[i:j:k] = t |
| del s[i:j:k] |
| s.append(x) |
| s.extend(x) |
| s.count(x) |
| s.index(x[, i[, j]]) |
| s.insert(i, x) |
| s.pop([i]) |
| s.remove(x) |
| s.reverse() |
| s.sort([cmp[, key[, reverse]]) |



5.2.17. Set Types: Set and Frozenset

The following methods are supported with the set and frozenset types:

| |
|----------------------|
| len(s) |
| x in s |
| x not in s |
| isdisjoint() |
| issubset() |
| set <= other() |
| set < other() |
| issuperset |
| set >= other() |
| set > other() |
| union |
| set other ...() |
| intersection |
| set & other & ...() |
| difference |
| set - other - ...() |
| symmetric_difference |
| set ^ other() |
| copy() |

The following methods are supported with the set type:

| |
|-----------------------------|
| update |
| set = other ...() |
| intersection_update |
| set &= other & ...() |
| difference_update |
| set -= other ...() |
| symmetric_difference_update |
| set ^= other() |
| add |
| remove |
| discard |
| pop |
| clear |



5.2.18. Mapping Types: Dictionary

The following operations are supported with the dictionaries:

| |
|---------------------------|
| len(d) |
| d[key] |
| d[key] = value |
| del d[key] |
| key in d |
| key not in d |
| iter(d) |
| clear() |
| copy() |
| fromkeys(seq,[value]) |
| get(key,[default]) |
| has_key(key) |
| items() |
| iteritems() |
| iterkeys() |
| itervalues() |
| keys() |
| pop(key,[default]) |
| popitem() |
| setdefault(key,[default]) |
| update([other]) |
| values() |
| viewitems() |
| viewkeys() |
| viewvalues() |

5.2.19. File Objects

The following methods are supported with the file objects:

| |
|------------------------|
| close() |
| flush() |
| fileno() |
| isatty() |
| next() |
| read([size]) |
| readline([size]) |
| readlines([sizehint]) |
| xreadlines() |
| seek(offset[, whence]) |
| tell() |
| write(str) |
| writelines(seq) |

The following attributes are supported with the file objects:



| |
|-----------|
| closed |
| encoding |
| errors |
| mode |
| name |
| newlines |
| softspace |

NOTE:

Root directory for Python scripts and in general for text and binary files is:
/sys
and cannot be changed.
Files path name in Python scripts shall refer to this root directory.
Path separator is /.



Example:

```
f = open('/sys/example.txt', 'rb')
```

5.2.20. Memoryview Objects

The following methods are supported with the memoryview objects:

| |
|-----------|
| tobytes() |
| tolist() |

The following attributes are supported with the memoryview objects:

| |
|----------|
| format |
| itemsize |
| shape |
| ndim |
| strides |
| readonly |

5.2.21. Module Objects

Module objects are supported.
The following attributes are supported:

| |
|-------------|
| <u>dict</u> |
| name |

5.2.22. Classes and Class Instances

Classes and class instances are supported.



5.2.23. Function Objects

Function objects and function call are supported.

5.2.24. Method Objects

Method objects are supported.

5.2.25. Code objects

Code objects are supported.

5.2.26. Type Objects

Type objects are supported.

5.2.27. Null Object

Null object is supported.

5.2.28. Ellipsis Object

Ellipsis object is supported.

5.2.29. NotImplemented Object

NotImplemented object is supported.

5.2.30. Internal Types: Frame Objects

Frame objects are supported.

5.2.31. Internal Types: Traceback Objects

Traceback objects are supported.

5.2.32. Slice Objects

Slice objects are supported.



5.2.33. Built-in Exceptions

The following built-in exceptions are supported:

| |
|---------------------|
| BaseException |
| Exception |
| StandardError |
| ArithmeticError |
| BufferError |
| LookupError |
| EnvironmentError |
| AssertionError |
| AttributeError |
| EOFError |
| FloatingPointError |
| GeneratorExit |
| IOError |
| ImportError |
| IndexError |
| KeyError |
| KeyboardInterrupt |
| MemoryError |
| NameError |
| NotImplementedError |
| OSError |
| OverflowError |
| ReferenceError |
| RuntimeError |
| StopIteration |
| SyntaxError |
| IndentationError |
| TabError |
| SystemError |
| SystemExit |
| TypeError |
| UnboundLocalError |
| UnicodeError |
| ValueError |
| ZeroDivisionError |

5.2.34. Built-in Modules: marshal

Built-in marshal module is supported with the following functions:

| |
|-------|
| dump |
| load |
| dumps |



| |
|-------|
| loads |
|-------|

The following constant is supported:

| |
|---------|
| version |
|---------|

5.2.35. Built-in Modules: **imp**

Built-in imp module is supported with the following functions:

| |
|--------------|
| find_module |
| get_magic |
| get_suffixes |
| load_module |
| new_module |
| lock_held |
| acquire_lock |
| release_lock |

The following constants are supported:

| |
|-------------|
| PY_SOURCE |
| PY_COMPILED |
| C_BUILTIN |
| PY_FROZEN |

5.2.36. Built-in Modules: **_ast**

Built-in _ast module is supported.

5.2.37. Built-in Modules: **__main__**

Built-in __main__ module is supported.

5.2.38. Built-in Modules: **__builtin__**

Built-in __builtin__ module is supported.



5.2.39. Built-in Modules: sys

Built-in sys module is supported with the following functions:

| |
|-------------------|
| _clear_type_cache |
| _current_frames |
| displayhook |
| exc_info |
| exc_clear |
| excepthook |
| exit |
| getrefcount |
| getrecursionlimit |
| getsizeof |
| _getframe |
| setcheckinterval |
| getcheckinterval |
| setprofile |
| getprofile |
| setrecursionlimit |
| settrace |
| gettrace |
| call_tracing |

The following variables are supported:

| |
|----------------------|
| stdin |
| stdout |
| stderr |
| __stdin__ |
| __stdout__ |
| __stderr__ |
| __displayhook__ |
| __excepthook__ |
| version |
| hexversion |
| subversion |
| _mercurial |
| dont_write_bytecode |
| api_version |
| copyright |
| platform |
| executable |
| prefix |
| exec_prefix |
| maxsize |
| maxint |
| py3kwarning |
| float_info |
| long_info |
| builtin_module_names |
| byteorder |
| warnoptions |
| version_info |
| flags |

| |
|------------------|
| float_repr_style |
|------------------|



| |
|---------------------|
| argv |
| exitfunc |
| last_type |
| last_value |
| last_traceback |
| modules |
| meta_path |
| path |
| path_hooks |
| path_importer_cache |
| tracebacklimit |

5.2.40. Built-in Modules: exceptions

Built-in exceptions module is supported.
See Built-in Exceptions paragraph.

5.2.41. Built-in Modules: gc

Built-in gc module is supported with the following functions:

| |
|---------------|
| enable |
| disable |
| isenabled |
| collect |
| set_debug |
| get_debug |
| get_objects |
| set_threshold |
| get_count |
| get_threshold |
| get_referrers |
| get_referents |
| is_tracked |

The following variable is supported:

| |
|---------|
| garbage |
|---------|

The following constants are supported:

| |
|---------------------|
| DEBUG_STATS |
| DEBUG_COLLECTABLE |
| DEBUG_UNCOLLECTABLE |
| DEBUG_INSTANCES |
| DEBUG_OBJECTS |
| DEBUG_SAVEALL |
| DEBUG_LEAK |



5.2.42. Built-in Modules: `_warnings`

Built-in `_warnings` module is supported.

5.2.43. Built-in Modules: `_md5`

Built-in `_md5` module is supported:

5.2.44. Built-in Modules: `binascii`

Built-in `binascii` module is supported with the following functions:

| |
|----------------------------|
| <code>a2b_uu</code> |
| <code>b2a_uu</code> |
| <code>a2b_base64</code> |
| <code>b2a_base64</code> |
| <code>a2b_qp</code> |
| <code>b2a_qp</code> |
| <code>a2b_hqx</code> |
| <code>rledecode_hqx</code> |
| <code>rlecode_hqx</code> |
| <code>b2a_hqx</code> |
| <code>crc_hqx</code> |
| <code>crc32</code> |
| <code>b2a_hex</code> |
| <code>hexlify</code> |
| <code>a2b_hex</code> |
| <code>unhexlify</code> |

The following exceptions are supported:

| |
|-------------------------|
| <code>Error</code> |
| <code>Incomplete</code> |

5.2.45. Built-in Modules: `_sre`

Built-in `_warnings` module is supported.

5.2.46. Built-in Modules: `_weakref`

Built-in `_weakref` module is supported.

5.2.47. Built-in Modules: `_symtable`

Built-in `_symtable` module is supported.

5.2.48. Built-in Modules: `_functools`

Built-in `_functools` module is supported.



5.2.49. Built-in Modules: `_socket`

Built-in `_socket` module is supported with the following functions:

| |
|--------------------------------|
| <code>socket</code> |
| <code>gethostbyname</code> |
| <code>gethostbyname_ex</code> |
| <code>gethostbyaddr</code> |
| <code>ntohs</code> |
| <code>ntohl</code> |
| <code>htons</code> |
| <code>htonl</code> |
| <code>inet_aton</code> |
| <code>inet_ntoa</code> |
| <code>getaddrinfo</code> |
| <code>getnameinfo</code> |
| <code>getdefaulttimeout</code> |
| <code>setdefaulttimeout</code> |

The following exceptions are supported:

| |
|-----------------------|
| <code>error</code> |
| <code>herror</code> |
| <code>gaierror</code> |
| <code>timeout</code> |

The following constants are supported:

| |
|-------------------------------|
| <code>has_ipv6</code> |
| <code>AF_INET</code> |
| <code>AF_UNSPEC</code> |
| <code>INADDR_ANY</code> |
| <code>INADDR_BROADCAST</code> |
| <code>IPPROTO_IP</code> |
| <code>IPPROTO_TCP</code> |
| <code>IPPROTO_UDP</code> |
| <code>IP_HDRINCL</code> |
| <code>IP_TOS</code> |
| <code>IP_TTL</code> |
| <code>MSG_DONTWAIT</code> |
| <code>SHUT_RD</code> |
| <code>SHUT_RDWR</code> |
| <code>SHUT_WR</code> |
| <code>SOCK_STREAM</code> |
| <code>SOCK_DGRAM</code> |
| <code>SOCK_RAW</code> |
| <code>SOL_SOCKET</code> |

| |
|----------------------------|
| <code>SO_ACCEPTCONN</code> |
| <code>SO_BROADCAST</code> |
| <code>SO_ERR</code> |
| <code>SO_KEEPALIVE</code> |
| <code>SO_LINGER</code> |
| <code>SO_RCVBUF</code> |
| <code>SO_RCVTIMEO</code> |



| |
|-----------------|
| SO_REUSEADDR |
| SO_SNDBUF |
| SO_TYPE |
| TCP_MAXSEG |
| TCP_NODELAY |
| AI_ADDRCONFIG |
| AI_ALL |
| AI_CANONNAME |
| AI_DEFAULT |
| AI_MASK |
| AI_NUMERICHOST |
| AI_PASSIVE |
| AI_V4MAPPED |
| AI_V4MAPPED_CFG |
| EAI_ADDRFAMILY |
| EAI_AGAIN |
| EAI_BADFLAGS |
| EAI_BADHINTS |
| EAI_FAIL |
| EAI_FAMILY |
| EAI_MEMORY |
| EAI_NODATA |
| EAI_NONAME |
| EAI_PROTOCOL |
| EAI_SERVICE |
| EAI_SOCKTYPE |
| EAI_SYSTEM |
| NI_DGRAM |
| NI_MAXHOST |
| NI_MAXSERV |
| NI_NAMEREQD |
| NI_NOFQDN |
| NI_NUMERICHOST |
| NI_NUMERICSERV |

The following non standard constant is supported:

| |
|--------------|
| SO_CONTEXTID |
|--------------|

Socket objects support the following methods:

| |
|--------|
| accept |
|--------|



| |
|---------------|
| bind |
| close |
| connect |
| connect_ex |
| fileno |
| getpeername |
| getsockname |
| getsockopt |
| listen |
| recv |
| recv_into |
| recvfrom |
| recvfrom_into |
| send |
| sendall |
| sendto |
| setblocking |
| settimeout |
| gettimeout |
| setsockopt |
| shutdown |

Socket objects support the following attributes:

| |
|---------|
| family |
| type |
| proto |
| timeout |

5.2.50. Built-in Modules: time

Built-in time module is supported with the following functions:

| |
|-------|
| time |
| clock |
| sleep |



5.2.51. Built-in Modules: posix

Built-in posix module is supported with the following functions:

| |
|------------|
| stat |
| unlink |
| remove |
| rename |
| open |
| close |
| closerange |
| lseek |
| read |
| write |
| fstat |
| isatty |
| strerror |

The following variable is supported:

| |
|---------|
| environ |
|---------|

The following exception is supported:

| |
|-------|
| error |
|-------|

The following constants are supported:

| |
|------|
| F_OK |
| R_OK |
| W_OK |
| X_OK |

5.2.52. Built-in Modules: thread

Built-in thread module is supported with the following functions:

| |
|--------------------|
| start_new_thread |
| exit |
| allocate_lock |
| get_ident |
| stack_size |
| (raises exception) |

Lock objects support the following methods:

| |
|---------|
| acquire |
| release |
| locked |

The following exception is supported:

| |
|-------|
| error |
|-------|



5.2.53. Built-in Modules: signal

Built-in signal module is supported with the following functions:

| |
|-----------|
| signal |
| getsignal |

The following constants are supported:

| |
|---------|
| SIG_DFL |
| SIG_IGN |

The following non standard constants are supported:

| |
|---------|
| SIGMDM |
| SIGMDM2 |

5.2.54. Built-in Modules: errno

Built-in errno module is supported with the following variable:

| |
|-----------|
| errorcode |
|-----------|

5.2.55. Built-in Modules: cStringIO

Built-in cStringIO module is supported with the following function:

| |
|----------|
| StringIO |
|----------|

The following objects are supported:

| |
|------------|
| InputType |
| OutputType |



5.2.56. Library Modules

A small collection of standard Python modules written in Python (not built-in) is available. These .py files are mostly identical to the ones available for PC with minor changes.

- os.py
- posixpath.py
- stat.py
- genericpath.py
- socket.py
- functools.py
- types.py
- threading.py
- hashlib.py



6. Python non standard functions

6.1. _socket non standard functions

6.1.1. Non standard socket option flag: SO_CONTEXTID

The following non standard constant is supported:

SO_CONTEXTID

It is the socket option flag used to link a socket object to a context identifier after GPRS Context Activation procedure.



NOTE:

Context Activation procedure can be obtained using AT+CGDCONT and AT#SGACT commands on MDM or MDM2 interface.

All socket objects must be linked to a context identifier.

In the following example

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_CONTEXTID, 1)
```

a socket object is created and then linked to context identifier number one.

NOTE:



There is a firewall always active on module.

Without firewall proper configuration socket methods might not work as expected.

Firewall can be configured using AT#FRWL command on MDM or MDM2 interface.

Sockets used by standard functions (e.g. gethostbyname) are silently linked to context identifier number one.



6.2. signal non standard functions

6.2.1. Non standard signal: SIGMDM

The following non standard constant is supported:

| |
|--------|
| SIGMDM |
|--------|

It is the signal number linked to the event of presence of data in MDM.read buffer.

Example:

```
signal.signal(signal.SIGMDM, MDMReadHandler)
```

6.2.2. Non standard signal: SIGMDM2

The following non standard constant is supported:

| |
|---------|
| SIGMDM2 |
|---------|

It is the signal number linked to the event of presence of data in MDM2.read buffer.

Example:

```
signal.signal(signal.SIGMDM2, MDM2ReadHandler)
```



7. Python Notes

7.1. Memory Limits

In order to prevent *memory error*, in phase of execution of the script, we advise you to consider the following limits:

- allocated memory for each variable;
- number of the variables.

The memory available on modules includes:

- 2 MB of Non Volatile Memory for the user scripts and data files;
- 2 MB RAM available for Python engine usage.

Some limits of the available NVM that affect file saving procedures and need to be considered are listed below:

| | |
|--|---------------|
| max number of files
open contemporary | 16 |
| max length of file
name | 16 characters |

It is highly recommended not to use the module as a data logger since all flash memories have limited number of writing and deleting cycles.

7.2. Other Limits

Some other Python limits that should be considered while developing your Python script in order to find an appropriate solution are listed below:

- Python scripts should not interfere with GSM/GPRS standard operations, for this reason Python scripts run at lower priority;
- GPIO polling frequency from Python scripts might be slower than expected.



8. Python Script Emulation on PC

8.1. Executing the Python script on PC

The steps required to have a script executing by the Python engine on PC in a similar way as on the **Telit module** are:

- install Python on PC;
- install optional serial package on PC;
- copy on PC Python modules that emulates custom built-in modules (MDM, MDM2, SER, GPIO, GPS);
- run the Python script.

8.1.1. Install Python

Download Python 2.7.2 installation from

<http://www.python.org/>

<http://www.python.org/download/releases/2.7.2/>

and install it.

8.1.2. Install optional serial package

Download PythonWin installation related to Python 2.7 from

<http://sourceforge.net/projects/pywin32/files/pywin32/>

and install it.

Download pyserial package installation from

<http://sourceforge.net/projects/pyserial/files/pyserial/>

and install it.

8.1.3. Copy Python modules

A collection of Python modules written in Python (not built-in) that emulates custom built-in modules is available.

MDM.py

MDM2.py

SER.py

GPIO.py

GPS.py

Copy these files on PC.

These modules make the difference between running the script on PC and on module.



8.1.4. Run the Python script

Run the Python script on PC.

Main differences between executing the Python script on PC compared to module are:

- script speed execution;
- different behaviour between emulating modules and custom built-in modules.



9. Document History

| Revision | Date | Changes |
|----------|------------|-------------|
| 0 | 2012-02-27 | First issue |

