

Title: GM862-PCS Python: User Manual

VERIFICATION AND APPROVALS

Function Responsabile ():_____

Project Manager ():_____

Authors (Giorgio De Pauli, Michela Vinci):_____

REVISION LOG

Issues	Modified Sections	Date	P.d.M nr.
1		30.09.04	

INDEX

1	INTRODUCTION	4
1.1	Scope of document	4
1.2	Abbreviation symbols	4
2	APPLICABLE DOCUMENTS.....	4
2.1	Internal Document.....	4
2.2	External documents	4
3	TECHNICAL CHARACTERISTICS.....	5
3.1	General.....	5
4	PYTHON SCRIPT IN GM862-PCS.....	7
4.1	Python supported features in GM862-PCS.....	7
4.2	Executing a Python script.....	7
4.2.1	Write Python script.....	7
4.2.2	Download Python script.....	7
4.2.3	Enable Python script.....	8
4.2.4	Execute Python script	8
4.2.5	Debug Python script.....	8
5	NEW SUPPORTED AT COMMAND SET	9
5.1	AT commands description.....	9
6	BUILT-IN MODULES.....	12
6.1	MDM built-in module	12
6.1.1	MDM.send(string, timeout)	12
6.1.2	MDM.receive(timeout).....	12
6.1.3	MDM.read()	13
6.1.4	MDM.sendbyte(byte, timeout).....	13
6.1.5	MDM.receivebyte(timeout).....	13
6.1.6	MDM.readbyte()	13
6.1.7	MDM.getDCD()	14
6.1.8	MDM.getCTS()	14
6.1.9	MDM.getDSR()	14
6.1.10	MDM.getRI()	14
6.1.11	MDM.setRTS().....	14
6.1.12	MDM.setDTR()	14
6.2	SER built-in module.....	16
6.2.1	SER.send(string).....	16
6.2.2	SER.receive(timeout).....	16
6.2.3	SER.read()	16
6.2.4	SER.sendbyte(byte).....	16
6.2.5	SER.receivebyte(timeout)	17
6.2.6	SER.readbyte()	17
6.2.7	SER.SetSpeed(speed)	17
6.3	GPIO built-in module	18
6.3.1	GPIO.setIOWrite(GPIOnumber, value)	18
6.3.2	GPIO.getIOWrite(GPIOnumber).....	18
6.3.3	GPIO.setIODir(GPIOnumber, value, direction)	18
6.3.4	GPIO.getIODir(GPIOnumber)	18

6.4	MOD built-in module	20
6.4.1	MOD.secCounter()	20
6.4.2	MOD.sleep(sleeptime)	20
6.4.3	MOD.reactivatePython()	20
7	PC SCRIPT DEBUG	21
7.1	PC Script Debug: Windows 2000 with Python 1.5.2+	21
7.1.1	Install Telit Python 1.5.2+ Package	21
7.1.2	Debug your script	21
7.2	PC Script Debug: Windows 2000 with Python 1.5.2	22
7.2.1	Install Python 1.5.2	22
7.2.2	Install win32all	22
7.2.3	Install pyserial-1.21	22
7.2.4	Install MDM.py, SER.py, GPIO.py, MOD.py	22
7.2.5	Debug your script	23
7.3	PC Script Debug: Windows 2000 with Python 2.3.4	24
7.3.1	Install Python 2.3.4	24
7.3.2	Install win32all	24
7.3.3	Install pyserial-2.0	24
7.3.4	Install MDM.py, SER.py, GPIO.py, MOD.py	24
7.3.5	Debug your script	24
7.4	PC Script Debug: Linux with Python 1.5.2, Python 1.5.2+	25
7.4.1	Install Python 1.5.2 or Python 1.5.2+	25
7.4.2	Install pyserial-1.21	25
7.4.3	Install MDM.py, SER.py, GPIO.py, MOD.py	25
7.4.4	Debug your script	26
8	PYTHON SCRIPTS TIPS	27
8.1	Python script supported features	27
8.2	Python script structure	27
8.3	Timers in Python script	28
8.4	Python script debug	29
8.5	Speed up Python script start	29
8.6	Warning about exceptions module during Python script start	29
8.7	File objects in Python script	30
8.8	Protect Python script know how	30
9	GM862-PCS PYTHON SCRIPTS LIMITS	31
9.1	Python limits	31
9.2	AT commands limits	31
9.3	Python script length limit	32

1 INTRODUCTION

The Data Terminal Module GM862-PCS Python has the possibility of executing a Python script.

1.1 Scope of document

Scope of this document is to introduce the Data Terminal Module GM862-PCS Python, to describe how to execute a Python script in the Data Terminal Module GM862-PCS Python and to give the details about the new supported AT command set and the new Python built-in modules. In the end there are a description on debugging Python scripts on PC, some tips about writing Python scripts for the Data Terminal Module GM862-PCS Python and a summary of known limits.

1.2 Abbreviation symbols

None

2 APPLICABLE DOCUMENTS

2.1 Internal Document

- a) Telit GM862-PCS/-GPRS/-GSM Product Description
- b) Telit GM862-PCS/-GPRS/-GSM Software User Guide

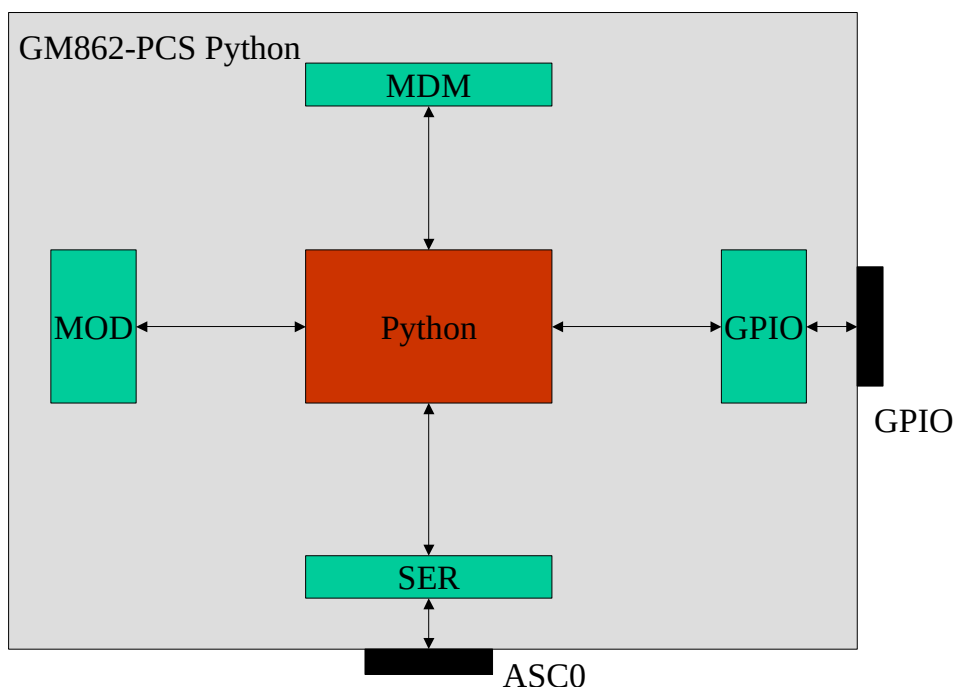
2.2 External documents

- a) Python 1.5.2 Tutorial
- b) Python 1.5.2 Library Reference
- c) Python 1.5.2 Reference Manual

3 TECHNICAL CHARACTERISTICS

3.1 General

You can run one Python script in the Data Terminal Module GM862-PCS Python. The Python script is stored in NVM inside the Data Terminal Module GM862-PCS Python. The Python script is executed in a task inside the Data Terminal Module GM862-PCS Python allowing all other tasks to interrupt it. This allows serial ports, protocol stack etc. to run independently from the Python script. The Python script interacts with the Data Terminal Module GM862-PCS Python functionalities through four interfaces.



MDM interface is the most important one. It allows Python script to send AT commands, receive AT responses and AT unsolicited indications, send data to the network and receive data from the network.

It is quite the same as the usual serial port interface in the Data Terminal Module GM862-PCS. The difference is that this interface is not a real serial port but just an internal software bridge between Python and mobile internal AT command handling.

All AT commands working in the Data Terminal Module GM862-PCS are working in this software interface as well. Some of them have no meaning on this interface, such as those regarding serial port speed.

The usual concept of flow control keeps its meaning over this interface.



SER interface allows Python script to read from and write to serial port ASC0. This serial port is free to be used by Python script because it is not used as AT command interface as it is usually. No flow control is available from Python on this port.

GPIO interface allows Python script to handle general purpose input output faster then through AT commands.

MOD interface is a collection of useful functions.

4 Python Script in GM862-PCS

4.1 Python supported features in GM862-PCS

Python version is 1.5.2+ (string methods added to 1.5.2).

You can use all Python statements and almost all Python built-in types and functions.

The following are not supported:

complex;

float;

long;

docstring.

Available modules are marshal, imp, __main__, __builtin__, sys and md5. All the others are not supported.

And you can use the following built-in modules:

MDM: interface between Python and mobile internal AT command handling;

SER: interface between Python and mobile internal serial port ASC0 direct handling;

GPIO: interface between Python and mobile internal general purpose input output direct handling;

MOD: interface between Python and mobile miscellaneous functions.

4.2 Executing a Python script

4.2.1 Write Python script

Write a Python script, remembering supported features described in 4.1, and save it on PC in a file.

The following is a short Python script that sends the simplest AT command to AT command interface and waits for response.

```
import MDM
a = MDM.send('AT', 0)
print a
b = MDM.sendbyte(0x0d, 0)
print b
c = MDM.receive(10)
print c
```

4.2.2 Download Python script

Download the Python script you wrote and saved to the module using the following AT command:

AT#WSCRIPT.

First you have to choose a name for your script in the module:

it must have extension .py;

the maximum allowed length is 16 characters;

script name is case sensitive.

Then you have to find out the size in bytes of the script you saved on PC (for example right clicking on the file and selecting "properties")

Now connect to the module serial port ASC0 (modem port) at 115200 8-N-1 with hardware flow control active.

Type (script name and size are examples):

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

wait for the prompt

```
>>>
```

and use "Send Text file" (ASCII Setup: Send line ends with line feeds) in HyperTerminal or "Send File" (RAW ASCII) in Procomm Plus application.

Wait for download result: OK or ERROR.

4.2.3 Enable Python script

Enable the Python script you downloaded to module using the following AT command:

```
AT#ESCRIP.T
```

First choose the script you want to enable between the ones you downloaded:

```
AT#LSCRIPT? can help you;
```

remember that script names are case sensitive.

Connect to the module serial port ASC0 (modem port).

Type (script name is an example):

```
AT#ESCRIP.T="a.py"
```

Wait for enable result: OK or ERROR.

4.2.4 Execute Python script

The Python script you downloaded to module and enabled is executed at every module power on

if no AT command interface is connected to ASC0 (modem port)

(the test is done on DTR line)

and

if the script name you enabled matches one of the script names of the scripts you downloaded.

The Python script is executed with `-v -S -OO` options.

The real execution of the Python script is delayed from the power on due to the time needed by Python to parse the script. The longer is the script, the longer is this delay.

4.2.5 Debug Python script

The debug of the executing Python script can be done on serial port ASC1 (trace port).

Connect to the module serial port ASC1 (trace port) at 9600 8-N-1 with no flow control active.

Now you can see all Python outputs to stdout and stderr:

Python information messages (for example version);

Python error informations;

results of all Python "print" statements.

5 New Supported AT Command Set

5.1 AT commands description

- **Select active script #ESCRIP**

Command	Possible responses
#ESCRIP=< script-name >	
#ESCRIP?	#ESCRIP: < script-name >

Description

Set command selects the name of the script to be executed at further start-up.
Read command returns the currently active script name.

Values

< script-name >: file name

NOTE. The script name should be passed between quotes.

- **Write script text #WSCRIPT**

Command	Possible responses
#WSCRIPT=<script-name>, <dim>[, <know-how>]	

Description

Set command insert a script text and save it with the name <script-name>

The script text should be sent using "Send Text file" in HyperTerminal application selecting in ASCII Setup the option "Send line ends with line feeds".

Or the script text should be sent using "Send File" in Procomm Plus application, selecting "Raw Ascii" or "Ascii" as Transfer Protocol. If you use "Ascii" transfer protocol, be sure the options "Expand tabs" and "Expand black lines" are not selected.

It is important to set properly the port settings. In particular:

Baud rate: 115200 bps

Flow control: hardware.

If know how protection is activated than any following attempt to read the script text through AT#RSCRIPT will return only OK: no Python script source code will be returned.

Values

< script-name >: file name

< dim >: file size (number of bytes)

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

NOTE. The script name should be passed between quotes.

- **Read script text #RSCRIPT**

Command	Possible responses
#RSCRIPT=< script-name >	

Description

Set command read a saved script text.

The script text read can be saved using "Capture Text" in HyperTerminal or "Capture File" in Procomm Plus application. Port settings should be:

Baud rate: 115200 bps

Flow control: hardware.

If know how protection is activated than AT#RSCRIPT will return only OK: no Python script source code will be returned.

Values

< script-name >: file name

NOTE. The script name should be passed between quotes.

- **List saved scripts AT#LSCRIPT**

Command	Possible responses
#LSCRIPT?	#LSCRIPT: (list of < script-name >s), < n >

Description

Read command shows the list of the scripts names currently saved and number of free bytes in memory.

- **Delete saved script AT#DSCRIPT**

Command	Possible responses
#DSCRIPT = < script-name >	

Description

Set command causes the deletion of a script from memory.

Values

< script-name >: file name

NOTE. The script name should be passed between quotes.



Restart AT#REBOOT

Command	Possible responses
#REBOOT	

Description

Execution command causes the module to restart, and the active script to be executed.

6 Built-in Modules

6.1 MDM built-in module

MDM built-in module is the interface between Python and mobile internal AT command handling.

You need to use MDM built-in module if you want to send AT commands from Python script to mobile and to receive AT responses from mobile to Python script.

Default start configuration is echo disabled (ATE0) and long form (verbose) return codes (ATV1),

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

```
import MDM
```

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

```
a = MDM.send('AT', 0)
b = MDM.sendbyte(0x0d, 0)
c = MDM.receive(10)
```

which sends 'AT' and receives 'OK'.

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

6.1.1 MDM.send(string, timeout)

Sends a string to AT command interface.

First input parameter string is a Python string which is the string to send to AT command interface.

Second input parameter timeout is a Python integer which is the value in 1/10 s to wait for the string to be sent to AT command interface before timeout expires. Waiting time is caused by flow control.

Return value is a Python integer which is -1 if timeout expired otherwise is 1.

Example:

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

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

6.1.2 MDM.receive(timeout)

Receives a string from AT command interface waiting for it until timeout is expired. Request to Send (RTS) is set to ON.

Input parameter timeout is a Python integer which is the value in 1/10 s to wait for a string from AT command interface before timeout expires.

Return value is a Python string which is an empty string if timeout expired without any data received otherwise is the string containing data received.

Example:

```
a = MDM.receive(15)
```

receives a string from AT command handling, possibly waiting for it for 1.5 s, assigning return value to a.

6.1.3 MDM.read()

Receives a string from AT command interface without waiting for it. Request to Send (RTS) is set to ON.

No input parameter.

Return value is a Python string which is an empty string if no data received otherwise is the string containing data received.

Example:

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

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

6.1.4 MDM.sendbyte(byte, timeout)

Sends a byte to AT command interface.

First input parameter byte is a Python byte which is any byte value to send to AT command interface. It can be zero.

Second input parameter timeout is a Python integer which is the value in 1/10 s to wait for the byte to be sent to AT command interface before timeout expires. Waiting time is caused by flow control.

Return value is a Python integer which is -1 if timeout expired otherwise is 1.

Example:

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

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

6.1.5 MDM.receivebyte(timeout)

Receives a byte from AT command interface waiting for it until timeout is expired. Request to Send (RTS) is set to ON.

Input parameter timeout is a Python integer which is the value in 1/10 s to wait for a byte from AT command interface before timeout expires.

Return value is a Python integer which is -1 if timeout expired without any data received otherwise is the byte value received. It can be zero.

Example:

```
b = MDM.receivebyte(20)
```

receives a byte from AT command handling, possibly waiting for it for 2.0 s, assigning return value to b.

6.1.6 MDM.readbyte()

Receives a byte from AT command interface without waiting for it. Request to Send (RTS) is set to ON.

No input parameter.

Return value is a Python integer which is -1 if no data received otherwise is the byte value received. It can be zero.

Example:

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

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

6.1.7 MDM.getDCD()

Gets Carrier Detect (DCD) from AT command interface.

No input parameter.

Return value is a Python integer which is 0 if DCD is OFF or 1 if DCD is ON.

Example:

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

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

6.1.8 MDM.getCTS()

Gets Clear to Send (CTS) from AT command interface.

No input parameter.

Return value is a Python integer which is 0 if CTS is OFF or 1 if CTS is ON.

Example:

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

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

6.1.9 MDM.getDSR()

Gets Data Set Ready (DSR) from AT command interface.

No input parameter.

Return value is a Python integer which is 0 if DSR is OFF or 1 if DSR is ON.

Example:

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

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

6.1.10 MDM.getRI()

Gets Ring Indicator (RI) from AT command interface.

No input parameter.

Return value is a Python integer which is 0 if RI is OFF or 1 if RI is ON.

Example:

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

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

6.1.11 MDM.setRTS()

Sets Request to Send (RTS) in AT command interface.

Input parameter is a Python integer which is 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.

6.1.12 MDM.setDTR()

Sets Data Terminal Ready (DTR) in AT command interface.

Input parameter is a Python integer which is 0 if setting DTR to OFF or 1 if setting DTR to ON.



DAI TELECOM

D a i T e l e c o m G r o u p

TECHNICAL SPECIFICATION

Code:	NppppST1xxxxA
Proj. nr:	pppp
Rel:	0
Date:	22.07.04

No return value.

Example:

MDM.setDTR(0)

sets DTR to OFF in AT command handling.

Documentazione riservata. Questo documento non può essere riprodotto, anche parzialmente, né reso noto a terzi senza preventiva autorizzazione scritta di DAI Telecom S.p.A.
Tutti i diritti sono tutelati a norma di legge.

Restricted document. No part of this document can be reproduced or copied in any format, in part or totally, nor its contents revealed in any manner to others without the express written permission of DAI Telecom S.p.A.
All rights reserved.

6.2 SER built-in module

SER built-in module is the interface between Python and mobile internal serial port ASC0 direct handling.

You need to use SER built-in module if you want to send data from Python script to serial port ASC0 and to receive data from serial port ASC0 to Python script.

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

```
import SER
```

then you can use SER built-in module methods like in the following example:

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

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

```
c = SER.receive(10)
```

which sends 'test' followed by CR and receives data waiting for one second.

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

6.2.1 SER.send(string)

Sends a string to serial port ASC0.

Input parameter string is a Python string which is the string to send to serial port ASC0.

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

Example:

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

sends string 'test' to serial port ASC0 handling, assigning return value to a.

6.2.2 SER.receive(timeout)

Receives a string from serial port ASC0 waiting for it until timeout is expired.

Input parameter timeout is a Python integer which is the value in 1/10 s to wait for a string from serial port ASC0 before timeout expires.

Return value is a Python string which is an empty string if timeout expired without any data received otherwise is the string containing data received.

Example:

```
a = SER.receive(15)
```

receives a string from serial port ASC0 handling, waiting for it for 1.5 s, assigning return value to a.

6.2.3 SER.read()

Receives a string from serial port ASC0 without waiting for it.

No input parameter.

Return value is a Python string which is an empty string if no data received otherwise is the string containing data received.

Example:

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

receives a string from serial port ASC0 handling, assigning return value to a.

6.2.4 SER.sendbyte(byte)

Sends a byte to serial port ASC0.

Input parameter byte is a Python byte which is any byte value to send to serial port ASC0. It can be zero.

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

Example:

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

sends byte 0x0d, that is CR, to serial port ASC0 handling, assigning return value to b.

6.2.5 SER.receivebyte(timeout)

Receives a byte from serial port ASC0 waiting for it until timeout is expired.

Input parameter timeout is a Python integer which is the value in 1/10 s to wait for a byte from serial port ASC0 before timeout expires.

Return value is a Python integer which is -1 if timeout expired without any data received otherwise is the byte value received. It can be zero.

Example:

```
b = SER.receivebyte(20)
```

receives a byte from serial port ASC0 handling, waiting for it for 2.0 s, assigning return value to b.

6.2.6 SER.readbyte()

Receives a byte from serial port ASC0 without waiting for it.

No input parameter.

Return value is a Python integer which is -1 if no data received otherwise is the byte value received. It can be zero.

Example:

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

receives a byte from serial port ASC0 handling, assigning return value to b.

6.2.7 SER.SetSpeed(speed)

Sets serial port ASC0 speed. Default serial port ASC0 speed is 9600.

Input parameter speed is a Python string which is the value of the serial port speed. It can be '9600', '19200' or '115200'.

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

Example:

```
b = SER.SetSpeed('115200')
```

sets serial port ASC0 speed to 115200, assigning return value to b.

6.3 GPIO built-in module

GPIO built-in module is the interface between Python and mobile internal general purpose input output direct handling.

You need to use GPIO built-in module if you want to set GPIO values from Python script and to read GPIO values from Python script.

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

```
import GPIO
```

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

```
a = GPIO.getIOvalue(5)
```

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

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

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

6.3.1 GPIO.setIOvalue(GPIOnumber, value)

Sets output value of a GPIO.

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

Second input parameter value is a Python integer which is the output value. It can be 0 or 1.

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 return value to b.

6.3.2 GPIO.getIOvalue(GPIOnumber)

Gets input or output value of a GPIO.

Input parameter GPIOnumber is a Python integer which is the number of the GPIO.

Return value is a Python integer which is -1 if an error occurred otherwise is input or output value. It is 0 or 1.

Example:

```
a = GPIO.getIOvalue(5)
```

gets GPIO 5 input or output value, assigning return value to b.

6.3.3 GPIO.setIOdir(GPIOnumber, value, direction)

Sets direction of a GPIO.

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

Second input parameter value is a Python integer which is the output value. It can be 0 or 1. It is only used if direction value is 1.

Third input parameter value is a Python integer which is the direction value. It can be 0 for input or 1 for output.

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

Example:

```
c = GPIO.setIOdir(4, 0, 0)
```

sets GPIO 4 to input with value having no meaning, assigning return value to c.

6.3.4 GPIO.getIOdir(GPIOnumber)

Gets direction of a GPIO.



Input parameter GPIOnumber is a Python integer which is the number of the GPIO.
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.

Example:

```
d = GPIO.getIOWdir(7)
```

gets GPIO 7 direction, assigning return value to d.

6.4 MOD built-in module

MOD built-in module is the interface between Python and mobile miscellaneous functions. You need to use MOD built-in module if you want to generate timers in Python script, to reactivate Python from Python script, etc.

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

```
import MOD
```

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

```
MOD.reactivatePython()
```

which reactivates Python after next exiting from Python script.

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

6.4.1 MOD.secCounter()

Returns seconds elapsed since 1 January 1970.

This method is useful for timers generation in Python script.

No input parameter.

Return value is a Python integer which is the value of seconds elapsed since 1 January 1970.

Example:

```
a = MOD.secCounter()
```

returns seconds elapsed since 1 January 1970.

6.4.2 MOD.sleep(sleeptime)

Blocks Python script execution for a given time.

Input parameter timesleep is a Python integer which is the time in 1/10 s to block script execution.

No return value.

Example:

```
MOD.sleep(15)
```

blocks Python script for 1.5 s.

6.4.3 MOD.reactivatePython()

Reactivates Python script after exiting from actual Python script.

This method is useful for Python script restart.

The effect of this method is to restart the complete procedure of selecting the Python script to be executed and of executing it.

If you want this method to have the expected effect you need to exit actual Python script as soon as possible after calling it (for example braking while or for loops).

No input parameter.

No return value.

Example:

```
MOD.reactivatePython()
```

reactivates Python after next exiting from Python script.

7 PC Script Debug

You have the possibility to debug the script you wrote running it on PC and connecting PC through serial cable to GM862 ASC0 (modem port).

The following configurations have been tested:

- 1) Windows 2000 with Python 1.5.2+;
- 2) Windows 2000 with Python 1.5.2;
- 3) Windows 2000 with Python 2.3.4;
- 4) Linux with Python 1.5.2, Python 1.5.2+.

7.1 PC Script Debug: Windows 2000 with Python 1.5.2+

Python 1.5.2+ is the version that runs in Data Terminal Module GM862-PCS Python.

Remember that supported features in Data Terminal Module GM862-PCS Python are those described in 4.1.

7.1.1 Install Telit Python 1.5.2+ Package

Install Telit Python 1.5.2+ Package on your PC running installation utility TelitPy1.5.2+.exe.

This package contains:

- Python 1.5.2+;
- win32all for Python 1.5.2;
- pyserial 1.21;
- Telit MDM.py, SER.py, GPIO.py, MOD.py interface files simulation on PC.

This package is provided by Dai Telecom.

7.1.2 Debug your script

Connect PC COM port you want to use as AT commands interface to GM862 ASC0 (modem port) and debug your script running it on PC.

7.2 PC Script Debug: Windows 2000 with Python 1.5.2

Python 1.5.2 is very close to the version that runs in Data Terminal Module GM862-PCS Python.

Remember that supported features in Data Terminal Module GM862-PCS Python are those described in 4.1.

7.2.1 Install Python 1.5.2

Install Python 1.5.2 on your PC running installation utility py152.exe.

You can look for it in <http://www.python.org/> and then download it.

7.2.2 Install win32all

Install win32all for Python 1.5.2 on your PC running installation utility win32all-143.exe.

You can look for it in <http://www.python.org/> and then download it.

7.2.3 Install pyserial-1.21

Install pyserial-1.21.

This needs some tricks because pyserial-1.21 version has been tested from authors on Python 2.0. You can look for it in <http://pyserial.sourceforge.net/> and download it.

Unzip pyserial-1.21.zip.

Edit file \pyserial-1.21\serial\serialutil.py and change line 29 from

```
line += c #not very efficient but lines are usually not that long
```

to

```
line = line + c #not very efficient but lines are usually not that long
```

and save file.

Copy all directory \serial in C:\Program Files\Python\Lib or similar.

Otherwise just copy all already modified directory \serial in C:\Program Files\Python\Lib or similar. Already modified directory \serial is provided by Dai Telecom.

7.2.4 Install MDM.py, SER.py, GPIO.py, MOD.py

Install MDM.py copying this file in C:\Program Files\Python\Lib or similar.

Edit this file and modify lines 33 and 47 according to the COM port you want to use as AT commands interface.

This Python module simulates on PC built-in module MDM that is available in Data Terminal Module GM862-PCS Python.

Install SER.py copying this file in C:\Program Files\Python\Lib or similar.

Edit this file and modify line 33 according to the COM port you want to use as free serial port.

This Python module simulates on PC built-in module SER that is available in Data Terminal Module GM862-PCS Python.

Install GPIO.py copying this file in C:\Program Files\Python\Lib or similar.

This Python module simulates on PC built-in module GPIO that is available in Data Terminal Module GM862-PCS Python.

Install MOD.py copying this file in C:\Program Files\Python\Lib or similar.

This Python module partially simulates on PC built-in module MOD that is available in Data Terminal Module GM862-PCS Python.

Files MDM.py, SER.py, GPIO.py, MOD.py are provided by Dai Telecom.



DAI TELECOM

D a i T e l e c o m G r o u p

TECHNICAL SPECIFICATION

Code: NppppST1xxxxA
Proj. nr: pppp
Rel: 0
Date: 22.07.04

7.2.5 Debug your script

Connect PC COM port you want to use as AT commands interface to GM862-PCS ASC0 (modem port) and debug your script running it on PC.

Documentazione riservata. Questo documento non può essere riprodotto, anche parzialmente, né reso noto a terzi senza preventiva autorizzazione scritta di DAI Telecom S.p.A.
Tutti i diritti sono tutelati a norma di legge.

Restricted document. No part of this document can be reproduced or copied in any format, in part or totally, nor its contents revealed in any manner to others without the express written permission of DAI Telecom S.p.A.
All rights reserved.

7.3 PC Script Debug: Windows 2000 with Python 2.3.4

Remember that Python 2.3.4 is not the version that runs in Data Terminal Module GM862-PCS Python and that supported features in Data Terminal Module GM862-PCS Python are those described in 4.1.

7.3.1 Install Python 2.3.4

Install Python 2.3.4 on your PC running installation utility Python-2.3.4.exe.
You can look for it in <http://www.python.org/> and then download it.

7.3.2 Install win32all

Install win32all for Python 2.3.4 on your PC running installation utility pywin32-201.win32-py2.3.
You can look for it in <http://www.python.org/> and then download it.

7.3.3 Install pyserial-2.0

Install pyserial-2.0 on your PC running installation utility pyserial-2.0.win32.exe.
This needs no tricks. You can look for it in <http://pyserial.sourceforge.net/> and download it.

7.3.4 Install MDM.py, SER.py, GPIO.py, MOD.py

Install MDM.py copying this file in C:\Python23\Lib or similar.
Edit this file and modify lines 33 and 47 according to the COM port you want to use as AT commands interface.
This Python module simulates on PC built-in module MDM that is available in Data Terminal Module GM862-PCS Python.
Install SER.py copying this file in C:\Program Files\Python\Lib or similar.
Edit this file and modify line 33 according to the COM port you want to use as free serial port.
This Python module simulates on PC built-in module SER that is available in Data Terminal Module GM862-PCS Python.
Install GPIO.py copying this file in C:\Program Files\Python\Lib or similar.
This Python module simulates on PC built-in module GPIO that is available in Data Terminal Module GM862-PCS Python.
Install MOD.py copying this file in C:\Program Files\Python\Lib or similar.
This Python module partially simulates on PC built-in module MOD that is available in Data Terminal Module GM862-PCS Python.
Files MDM.py, SER.py, GPIO.py, MOD.py are provided by Dai Telecom.

7.3.5 Debug your script

Connect PC COM port you want to use as AT commands interface to GM862 ASC0 (modem port) and debug your script running it on PC.

7.4 PC Script Debug: Linux with Python 1.5.2, Python 1.5.2+

Python 1.5.2 is very close to the version that runs in Data Terminal Module GM862-PCS Python.

Python 1.5.2+ is the version that runs in Data Terminal Module GM862-PCS Python.

Remember that supported features in Data Terminal Module GM862-PCS Python are those described in 4.1.

7.4.1 Install Python 1.5.2 or Python 1.5.2+

Install Python 1.5.2 or Python 1.5.2+ on your PC downloading source code and running through usual configure, make, make install procedure according to file README.

You can look for source code in <http://www.python.org/> and then download it.

You need to compile file \Modules\termios.c because pyserial imports termios module. To get file \Modules\termios.c compiled edit \Modules\ Setup.in and remove comment in line

```
#termios termios.c # Steen Lumholt's termios module  
changing it to  
termios termios.c # Steen Lumholt's termios module  
and save file.
```

7.4.2 Install pyserial-1.21

Install pyserial-1.21.

This needs some tricks because pyserial-1.21 version has been tested from authors on Python 2.0. You can look for it in <http://pyserial.sourceforge.net/> and download it.

Unzip pyserial-1.21.zip.

Edit file \pyserial-1.21\serial\ serialutil.py and change line 29 from

```
line += c #not very efficient but lines are usually not that long
```

to

```
line = line + c #not very efficient but lines are usually not that long
```

and save file.

Copy all directory \serial in \lib\python1.5 or similar.

Otherwise just copy all already modified directory \serial in \lib\python1.5 or similar. Already modified directory \serial is provided by Dai Telecom.

7.4.3 Install MDM.py, SER.py, GPIO.py, MOD.py

Install MDM.py copying this file in \lib\python1.5 or similar.

Edit this file and modify lines 33 and 47 according to the serial port number you want to use as AT commands interface.

This Python module simulates on PC built-in module MDM that is available in Data Terminal Module GM862-PCS Python.

Install SER.py copying this file in \lib\python1.5 or similar.

Edit this file and modify line 33 according to the serial port number you want to use as free serial port.

This Python module simulates on PC built-in module SER that is available in Data Terminal Module GM862-PCS Python.

Install GPIO.py copying this file in \lib\python1.5 or similar.



This Python module simulates on PC built-in module GPIO that is available in Data Terminal Module GM862-PCS Python.

Install MOD.py copying this file in \lib\python1.5 or similar.

This Python module partially simulates on PC built-in module MOD that is available in Data Terminal Module GM862-PCS Python.

Files MDM.py, SER.py, GPIO.py, MOD.py are provided by Dai Telecom.

7.4.4 Debug your script

Connect PC serial port number you want to use as AT commands interface to GM862 ASC0 (modem port) and debug your script running it on PC.

8 Python scripts tips

The following tips come from Dai Telecom experience.

8.1 Python script supported features

Remember that supported features in Data Terminal Module GM862-PCS Python are those described in 4.1.

8.2 Python script structure

The Python script you downloaded to module and enabled is executed at every module power on according to 4.2.4.

This means that if your script exits then you will lose control on Data Terminal Module GM862-PCS Python.

Very likely you need to keep control on Data Terminal Module GM862-PCS Python and you can achieve this result adding to your Python script a loop and breaking it only when module is going to power off. In order to control Data Terminal Module GM862-PCS Python in this loop there will be the proper polling on mobile internal AT command interface, on mobile internal serial port ASC0 interface and on mobile internal GPIO interface.

The suggested structure for the Python script is the following:

initialization

while 1:

 polling on GPIO

 actions related to GPIO

 polling on AT command interface

 actions related to AT command interface

 polling on serial port ASC0 interface

 actions related to serial port ASC0 interface

 if (conditions to break loop):

 break

Of course if you are not using serial port ASC0, for example, you will cut the corresponding part in the Python script suggested structure.



8.3 Timers in Python script

You have the possibility to block Python script execution for a given time using MOD.sleep(). But during this time you are not polling on available interfaces.

You have the possibility to write your own class of timers to be used inside the loop to test if a given time is elapsed. The following is a tested example:

```
import MOD

class timer:

    def __init__(self, seconds):
        self.start(seconds)

    def start(self, seconds):
        self.startTime = MOD.secCounter()
        self.expirationTime = self.startTime + seconds
        if seconds != 0:
            self.running = 1
            self.expired = 0
        else:
            self.running = 0
            self.expired = 0

    def stop(self):
        self.running = 0
        self.expired = 0

    def isexpired(self):
        if self.running == 1:
            timeNow = MOD.secCounter()
            if timeNow > self.expirationTime:
                self.running = 0
                self.expired = 1
            else:
                self.expired = 0
        return self.expired

    def isrunning(self):
        if self.running == 1:
            timeNow = MOD.secCounter()
            if timeNow > self.expirationTime:
                self.running = 0
                self.expired = 1
            else:
                self.expired = 0
        return self.running

    def change(self, seconds):
        self.expirationTime = self.startTime + seconds

    def count(self):
        if self.running == 1:
            timeNow = MOD.secCounter()
            return (timeNow - self.startTime)
        else:
            return -1
```

8.4 Python script debug

Add print statements to debug your Python script.

It is better to remove them at the end of debugging.

Python script debug is easier running it on PC with the serial port number you want to use as AT commands interface connected to GM862 ASC0 (modem port) as described in 7.

8.5 Speed up Python script start

As described in 4.2.4 the real execution of the Python script is delayed from the power on due to the time needed by Python to parse the script.

It is possible to speed up Python script start, taking advantage of the fact that Python compiles imported modules. The suggestion is to split the Python script in a small script and a big one. The small script simply imports the big one and executes a main function of the imported module. The big script has a main function, to be called by the small one, in which is included all the control on Data Terminal Module GM862-PCS Python and of course can have a structure like the one suggested in 8.2.

The following is the suggested small script structure:

```
import bigscriptname  
bigscriptname.main()
```

The following is the suggested big script structure:

```
def main():  
    initialization  
    while 1:  
        polling on GPIO  
        actions related to GPIO  
        polling on AT command interface  
        actions related to AT command interface  
        polling on serial port ASC0 interface  
        actions related to serial port ASC0 interface  
        if (conditions to break loop):  
            break
```

In this way the imported big script will be compiled during first execution and saved to bigscriptname.pyo file taking the usual long time to start, but in next executions the start phase will be very fast.

8.6 Warning about exceptions module during Python script start

Don't worry about this warning:

'import exceptions' failed; traceback:

ImportError: No module named exceptions

Warning! Falling back to string-based exceptions

The reason of this warning is that module exceptions is imported automatically at startup by Python but it not present in module. But it is only a warning.

Anyway if you want to avoid it simply download file exceptions.py to the module using the following AT command:

AT#WSCRIPT="exceptions.py",7168

Module exceptions will be compiled during first execution and saved to exceptions.pyo file taking the usual long compilation time, but in next executions the start phase will be very fast. File exceptions.py is provided by Dai Telecom.

8.7 File objects in Python script

If you want to write a file in the module or read a file from the module from your Python script it is not worth using AT#WSCRIPT and AT#RSCRIPT. It is easier to use file object and file objects methods write or read.

8.8 Protect Python script know how

If you want to protect your Python script know how you have to write your Python script in the module using the <know-how> parameter of AT#WSCRIPT command like in the following example:

AT#WSCRIPT="a.py",110, 1

Any further attempt to read your Python script from the module through AT#RSCRIPT will return only OK: no Python script source code will be returned.

In this way nobody will be able to read your Python script from the module.

The Python script will be still in the Python scripts list. It will be still possible to delete the Python script and to overwrite it.

9 GM862-PCS Python scripts limits

The following paragraphs summarize what you can do and what you can't do with Data Terminal Module GM862-PCS Python.

9.1 Python limits

The following are Python supported features in Data Terminal Module GM862-PCS Python, described in 4.1 as well.

Python version is 1.5.2+ (string methods added to 1.5.2).

You can use all Python statements and almost all Python built-in types and functions.

The following are not supported:

complex;
float;
long;
docstring.

Available modules are marshal, imp, __main__, __builtin__, sys and md5. All the others are not supported.

And you can use the following built-in modules:

MDM: interface between Python and mobile internal AT command handling;

SER: interface between Python and mobile internal serial port ASC0 direct handling;

GPIO: interface between Python and mobile internal general purpose input output direct handling;

MOD: interface between Python and mobile miscellaneous functions.

9.2 AT commands limits

If you use Data Terminal Module GM862-PCS Python controlled from external device through usual serial port there are no limits to supported AT commands. The behavior of Data Terminal Module GM862-PCS Python is the same as standard Data Terminal Module GM862-PCS.

If you use a Python script running inside Data Terminal Module GM862-PCS Python there are some slight limits to AT commands supported, introduced in 3.1, 6.3 and 8.7 as well.

Generally all AT commands supported in the Data Terminal Module GM862-PCS are supported in Python scripts in Data Terminal Module GM862-PCS Python. Refer to the following document for more details: Telit GM862-PCS/-GPRS/-GSM Product Description.

Some AT commands have no meaning in a Python script that is running inside Data Terminal Module GM862-PCS Python:

- AT+IPR: it is an AT command related to serial port speed;
- AT+FPR: it is an AT command related to serial port speed.

Some AT commands are not worth using in a Python script that is running inside Data Terminal Module GM862-PCS Python because there are Python methods easier to use

- AT#GPO, AT#GPI, AT#GPIO: it is easier to use Python GPIO built-in module methods;
- AT#WSOCKET, AT#RSCRIPT: it is easier to use Python file objects and Python file objects methods write or read.

The usual concept of flow control keeps its meaning when exchanging data between a Python script and the network.



9.3 Python script length limit

It is not possible to say the exact maximum length of a Python script running inside Data Terminal Module GM862-PCS Python.

In fact the limit is due to the available RAM. Python uses a lot of RAM, especially during parsing phase and this can cause an early exit from Python script with an error message related to lack of memory.