



# Basic Development Guide for Open AT<sup>®</sup> v3.03

Revision: 001  
Date: November 2005



**wavecom**<sup>®</sup>  
Make it wireless

# **Basic Development Guide**


## **for Open-AT<sup>®</sup> V3.03**

Reference : **WM\_ASW\_OAT\_UGD\_00052**  
Revision : **001**  
Date : **29<sup>th</sup> November 2005**

## Document History

Revision	Date	History	
001	November 29th 2005	Creation	

## Trademarks

 <sup>®</sup>, WAVECOM<sup>®</sup>, WISMO<sup>®</sup>, Open AT<sup>®</sup>, and certain other trademarks and logos appearing on this document, are filed or registered trademarks of Wavecom S.A. in France or in other countries. All other company and/or product names mentioned may be filed or registered trademarks of their respective owners.

## Copyright

This manual is copyrighted by WAVECOM with all rights reserved. No part of this manual may be reproduced in any form without the prior written permission of WAVECOM.

No patent liability is assumed with respect to the use of the information contained herein.

## Overview

This User's Guide describes the Open AT<sup>®</sup> facility and provides guidelines for developing an Embedded Application. It applies to Open AT<sup>®</sup> V3.03 and upper versions until further notice.

# TABLE OF CONTENTS

<b>1</b>	<b>Introduction.....</b>	<b>9</b>
1.1	References.....	9
1.2	Glossary .....	9
1.3	Abbreviations .....	10
<b>2</b>	<b>DESCRIPTION.....</b>	<b>11</b>
2.1	Software Architecture .....	11
2.1.1	Software Organization .....	11
2.1.2	Software Supplied by Wavecom .....	12
2.2	Minimum Embedded Application Code.....	13
2.3	Specificity of AT Commands in the Open-AT Architecture.....	14
2.3.1	AT Command Size .....	14
2.3.2	AT+WDWL Command .....	14
2.3.3	AT+WOPEN Command.....	14
2.4	Notes on Memory Management .....	15
2.5	Known Limitations .....	16
2.5.1	Command Pre-Parsing Limitation .....	16
2.5.2	Missing Unsolicited Messages in Remote Application .....	16
2.6	Minimum Embedded Application Code.....	16
2.7	Security.....	17
2.7.1	Software Security .....	17
2.7.2	If an Embedded Application crash is detected, the Target Monitoring Tool screen displays: "Customer watchdog."Hardware Security	18
<b>3</b>	<b>API.....</b>	<b>19</b>
3.1	Data Types .....	19
3.2	Mandatory Functions .....	19
3.2.1	Required Header .....	19
3.2.2	Task identifiers .....	19
3.2.3	Task table .....	19
3.2.4	Stack Initialization.....	20
3.2.5	The Init Functions.....	21
3.2.6	The Parser Functions.....	22
3.3	AT Command API.....	27
3.3.1	Required Header .....	27
3.3.2	The wm_atSendCommand Function.....	27
3.3.3	The wm_atUnsolicitedSubscription Function .....	30
3.3.4	The wm_atIntermediateSubscription Function .....	32
3.3.5	The wm_atCmdPreParserSubscribe Function .....	34
3.3.6	The wm_atRspPreParserSubscribe Function.....	36

3.3.7	The wm_atSendRspExternalApp Function .....	39
3.3.8	The wm_atSendRspExternalAppExt Function .....	39
3.3.9	The wm_atSendUnsolicitedExternalApp Function .....	40
3.3.10	The wm_atSendIntermediateExternalApp Function.....	41
3.3.11	The wm_atSendIntermediateExternalAppExt Function.....	42
3.4	OS API .....	43
3.4.1	Required Header .....	43
3.4.2	The wm_osStartTimer Function .....	43
3.4.3	The wm_osStopTimer Function .....	44
3.4.4	The wm_osStartTickTimer Function.....	44
3.4.5	The wm_osStopTickTimer Function .....	45
3.4.6	The wm_osDebugTrace Function .....	46
3.4.7	The wm_osDebugFatalError Function .....	47
3.4.8	Important Note on Data Flash Management .....	48
3.4.9	The wm_osWriteFlashData Function .....	48
3.4.10	The wm_osReadFlashData Function .....	49
3.4.11	The wm_osGetLenFlashData Function .....	49
3.4.12	The wm_osDeleteFlashData Function .....	50
3.4.13	The wm_osGetAllowedMemoryFlashData Function.....	50
3.4.14	The wm_osGetFreeMemoryFlashData Function .....	50
3.4.15	The wm_osGetUsedMemoryFlashData Function.....	51
3.4.16	The wm_osDeleteAllFlashData Function .....	51
3.4.17	The wm_osDeleteRangeFlashData Function .....	51
3.4.18	The wm_osGetHeapMemory Function .....	52
3.4.19	The wm_osReleaseHeapMemory Function .....	52
3.4.20	The wm_osSuspend function .....	52
3.4.21	The wm_osGetTask Function.....	53
3.4.22	The wm_osSendMsg Function .....	53
3.4.23	Example: Managing Data Flash Objects .....	54
3.4.24	Example: RAM management .....	54
3.5	Flow Control Manager API .....	55
3.5.1	Required Header .....	56
3.5.2	The wm_fcmFlow_e enum type .....	56
3.5.3	The wm_fcmOpen Function.....	57
3.5.4	The wm_fcmClose Function.....	58
3.5.5	The wm_fcmSubmitData Function.....	59
3.5.6	Receive Data Blocks.....	60
3.5.7	The wm_fcmCreditToRelease Function .....	61
3.5.8	The wm_fcmQuery Function.....	62
3.6	Input Output API .....	63
3.6.1	Required Header .....	63
3.6.2	Serial Link State functions .....	63
3.6.3	GPIO types and functions .....	65
3.7	GPRS API .....	75
3.7.1	GPRS Overview .....	75
3.7.2	The wm_gprsAuthentication function .....	77
3.7.3	The wm_gprsIPCPIInformations function .....	78
3.7.4	The wm_gprsOpen function .....	79
3.7.5	The wm_gprsClose function .....	79
3.8	BUS API .....	80
3.8.1	Required Header .....	80
3.8.2	Returned values definition .....	80

3.8.3	The wm_busOpen Function .....	81
3.8.4	The wm_busClose Function .....	86
3.8.5	The wm_busWrite Function .....	87
3.8.6	The wm_busRead Function .....	88
3.8.7	Error codes values .....	90
3.9	Scratch Memory API .....	90
3.9.1	Required Header .....	90
3.9.2	Returned values definition .....	90
3.10	Lists management API .....	91
3.10.1	Required Header .....	91
3.10.2	Types definition .....	91
3.10.3	The wm_lstCreate Function .....	92
3.10.4	The wm_lstDestroy Function .....	92
3.10.5	The wm_lstClear Function .....	93
3.10.6	The wm_lstGetCount Function .....	93
3.10.7	The wm_lstAddItem Function .....	94
3.10.8	The wm_lstInsertItem Function .....	94
3.10.9	The wm_lstGetItem Function .....	95
3.10.10	The wm_lstDeleteItem Function .....	95
3.10.11	The wm_lstFindItem Function .....	96
3.10.12	The wm_lstFindAllItem Function .....	96
3.10.13	The wm_lstFindNextItem Function .....	97
3.10.14	The wm_lstResetItem Function .....	97
3.11	Sound API .....	98
3.11.1	Required header .....	98
3.11.2	The wm_sndTonePlay Function .....	98
3.11.3	The wm_sndToneStop Function .....	100
3.11.4	The wm_sndDtmfPlay Function .....	101
3.11.5	The wm_sndDtmfStop Function .....	102
3.11.6	The wm_sndMelodyPlay Function .....	103
3.11.7	The wm_sndMelodyStop Function .....	104
3.12	Standard Library .....	105
3.12.1	Required Header .....	105
3.12.2	Standard C function set .....	105
3.12.3	String processing function set .....	106
3.13	Application & Data storage API .....	107
3.13.1	Required Header .....	107
3.13.2	Returned values definition .....	107
3.13.3	The wm_adAllocate Function .....	108
3.13.4	The wm_adRetrieve Function .....	108
3.13.5	The wm_adFindInit Function .....	109
3.13.6	The wm_adFindNext Function .....	109
3.13.7	The wm_adWrite Function .....	110
3.13.8	The wm_adFinalise Function .....	110
3.13.9	The wm_adResume Function .....	111
3.13.10	The wm_adInfo Function .....	111
3.13.11	The wm_adDelete Function .....	112
3.13.12	The wm_adStats Function .....	112
3.13.13	The wm_adSpaceState Function .....	113
3.13.14	The wm_adFormat Function .....	113
3.13.15	The wm_adRecompactInit Function .....	113
3.13.16	The wm_adRecompact Function .....	114

3.13.17	The wm_adInstall Function.....	114
3.14	[Deprecated] WAP API.....	114
3.15	GPS API	115
3.15.1	Required Header .....	115
3.15.2	The wm_gpsGetPosition Function .....	115
3.15.3	The wm_gpsGetSpeed Function .....	116
3.15.4	The wm_gpsGetSatview Function .....	117
<b>4</b>	<b>FUNCTIONING.....</b>	<b>118</b>
4.1	Standalone External Application .....	118
4.2	Embedded Application in Standalone Mode .....	120
4.3	Cooperative Mode .....	123
4.3.1	Command Pre-Parsing Subscription Mechanism: WM_AT_CMD_PRE_EMBEDDED_TREATMENT .....	124
4.3.2	Command Pre-Parsing Subscription Process: WM_AT_CMD_PRE_BROADCAST .....	128
4.3.3	Response Pre-Parsing Subscription Process: WM_AT_RSP_PRE_EMBEDDED_TREATMENT .....	131
4.3.4	Response Pre-Parsing Subscription Process: WM_AT_RSP_PRE_BROADCAST .....	135
4.3.5	Example: Embedded Application Using the Different Functioning Modes .....	138



## LIST OF FIGURES

Figure 1: General software architecture.....	11
Figure 2: Reset hardware security example .....	18
Figure 3: Flow Control Function .....	55
Figure 4: Parallel bus chronogram .....	85
Figure 5: Standalone external application function .....	118
Figure 6: Embedded Application in standalone mode function .....	120
Figure 7: WM_AT_CMD_PRE_EMBEDDED_TREATMENT .....	124
Figure 8: WM_AT_CMD_PRE_BROADCAST .....	128
Figure 9: WM_AT_RSP_PRE_EMBEDDED_TREATMENT .....	131
Figure 10: WM_AT_RSP_PRE_BROADCAST .....	135

# 1 Introduction

## 1.1 References

- I. Tools Manual (ref WM\_ASW\_OAT\_UGD\_00051 revision 001 for Open AT<sup>®</sup> V3.03)
- II. AT Commands Interface Guide (for AT x51: ref WM\_ASW\_OAT\_UGD\_016 revision 006)

## 1.2 Glossary

<b>Application Mandatory API</b>	Mandatory software interfaces to be used by the Embedded Application.
<b>AT commands</b>	Set of standard modem commands.
<b>AT function</b>	Software that processes the AT commands and AT subscriptions.
<b>Embedded API layer</b>	Software developed by Wavecom, containing the Open-AT <sup>®</sup> APIs (Application Mandatory API, AT Command Embedded API, OS API, Standard API, FCM API, IO API, and BUS API).
<b>Embedded Application</b>	User application sources to be compiled and run on a Wavecom product.
<b>Embedded Core software</b>	Software that includes the Embedded Application and the Wavecom library.
<b>Embedded software</b>	User application binary: set of Embedded Application sources + Wavecom library.
<b>External Application</b>	Application external to the Wavecom product that sends AT commands through the serial link.
<b>Target</b>	Open AT <sup>®</sup> compatible product supporting an Embedded Application.
<b>Target Monitoring Tool</b>	Set of utilities used to monitor a Wavecom product.
<b>Receive command pre-parsing</b>	Process for intercepting AT responses.
<b>Send command pre-parsing</b>	Process for intercepting AT commands.
<b>Standard API</b>	Standard set of "C" functions.
<b>Wavecom library</b>	Library delivered by Wavecom to interface Embedded Application sources with Wavecom Core Software functions.
<b>Wavecom Core Software</b>	Set of GSM and open functions supplied to the User.

### 1.3 Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
IR	Infrared
KB	Kilobyte
OS	Operating System
PDU	Protocol Data Unit
RAM	Random-Access Memory
ROM	Read-Only Memory
RTK	Real-Time Kernel
SMA	Small Adapter
SMS	Short Message Services
SDK	Software Development Kit

## 2 DESCRIPTION

### 2.1 Software Architecture

#### 2.1.1 Software Organization

The Open-AT<sup>®</sup> facility is a software mechanism. It relies on the following software architecture:

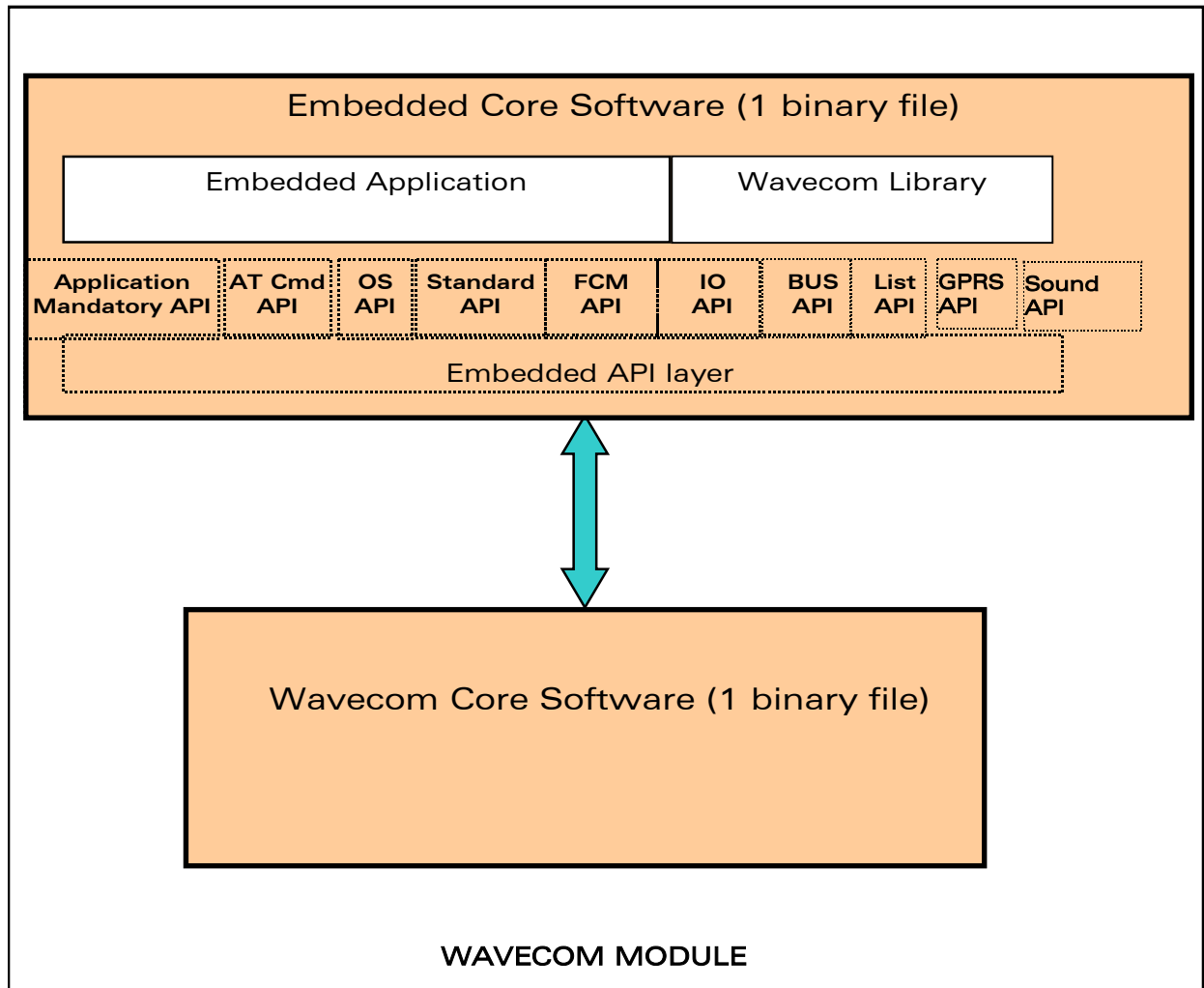


Figure 1: General software architecture

The different software elements on a Wavecom product are described here-below.

The **Embedded Core Software** (binary file) includes the following items:

- ❑ the Embedded Application: application to be developed and downloaded into the Wavecom Target product. The Embedded Application must be linked to the Wavecom library.
- ❑ the Wavecom library: software library provided by Wavecom (included in the Open-AT<sup>®</sup> SDK) and based on the Embedded API layer.
- ❑ the Embedded API Layer (developed by Wavecom), which includes:
  - the Application Mandatory API : mandatory software interfaces to be used by the Embedded Application,
  - the AT Command API : software interfaces providing access to the set of AT functions,
  - the OS API : software interfaces providing access to the Operating System functions,
  - the FCM API : software interfaces providing access to the Flow Control Manager functions (secure access to V24 and Data IO flows),
  - the IO API : software interfaces providing control on the serial link mode, and on the Gpio devices,
  - the GPRS API : software interfaces providing access to the GPRS service (for authentication and IPCP information),
  - the BUS API : software interfaces providing control on bus devices (as SPI or I2C bus),
  - the List API : set of list processing functions.
  - the Sound API: set of sound processing functions
  - the Standard API : standard set of "C" functions, in addition of some string processing functions,

The **Wavecom Core Software** (another binary file), manages the GSM protocol.

### 2.1.2 Software Supplied by Wavecom

The software items supplied are as follows:

- ❑ one software library, `wmopenat.lib`,
- ❑ one set of header files (.h), defining the Open AT<sup>®</sup> API functions,
- ❑ source code samples,
- ❑ a set of tools called Development ToolKit, for designing and testing any application (see document [Ref I]).

## 2.2 Minimum Embedded Application Code

The following code must be included in any Embedded Application:

```
u32 wm_apmCustomStack [ 256 ];
/* the value 256 is an example */
const u16 wm_apmCustomStackSize = sizeof (wm_apmCustomStack);

s32 wm_apmAppliInit (wm_apmInitType_e  InitType)
{
    return OK;
}

s32 wm_apmAppliParser ( wm_apmMsg_t * Message )
{
    return OK;
}
```

`wm_apmCustomStack` and `wm_apmCustomStackSize` are two mandatory variables, used to define the application call stack size (see § 3.2.4).

`wm_apmAppliInit()` is a mandatory function; this is the first function called at the Embedded Application initialization (see § 3.2.3).

`wm_apmAppliParser()` is a mandatory function; it is called each time the Embedded Application receives a message from the Wavecom Core Software (see § 3.2.4).

**Important Remark :** in former Open AT<sup>®</sup> versions, the `wm_apmCustomStack` variable was declared as an u8 table. This is modified since version 2, when `wm_apmCustomStack` became an u32 table, for memory alignment compatibility purpose with ADS compiler.

## **2.3 Specificity of AT Commands in the Open AT<sup>®</sup> Architecture**

See document [AT Commands Interface Guide].

### **2.3.1 AT Command Size**

The maximum size of an AT command string or a Response string that can be sent through the serial link is 512 bytes. Therefore, if the Embedded Application needs to send more data, it must be sent in several increments.

### **2.3.2 AT+WDWL Command**

By default the AT+WDWL command, used to download an application, is not pre-parsed. Therefore, even if the Embedded Application has subscribed to the command pre-parsing mechanism, this command is processed by means of the Wavecom software and it is not sent back to this application. The embedded application may also pre-parse it to prevent external application downloading another application over it.

**Note:**

the AT+WDWL command is described in the document [Ref II].

### **2.3.3 AT+WOPEN Command**

Open AT<sup>®</sup> require some specific AT commands such as AT+WOPEN. The latter is described below.

By default this command is always available for an External Application. It is not pre-parsed and it is processed even if the AT software is busy. The embedded application may also pre-parse it to prevent external application stopping it.

This command deactivates an Embedded Application in order to ensure that a new application can be downloaded. Typically, if an Embedded Application continuously sends AT commands, the Wavecom AT command software is always busy. Therefore, if the AT+WDWL command is sent by an External Application, it is not processed.

AT+WOPEN can take the values

- 0 Stop,
- 1 Start,
- 2 Get the Open AT<sup>®</sup> library versions,
- 3 Erase the objects flash of the Open AT<sup>®</sup> Embedded Application,
- 4 Erase the Open AT<sup>®</sup> Embedded Application).
- 5 Suspends the Open AT<sup>®</sup> application tasks (same effect than the `wm_osSuspend` API). Open AT<sup>®</sup> application running in Remote Task Environment can not be suspended (the command has no effect).

- ❑ Sending the AT+WOPEN=0 command first, by means of an External Application, deactivates the Embedded Application: a new Embedded Application may then be downloaded, or the actual objects flash can be erased or the actual Open AT<sup>®</sup> Embedded Application can be erased.
- ❑ If the Embedded Application is deactivated, it can be restarted using AT+WOPEN=1. The module then reboots and this application is restarted 20 sec after the module boot. In this case, the objects flash or the application can't be erased.

**Remark :**

You can check if the Open AT<sup>®</sup> feature is enabled with the command AT+WOPEN=? . In the case of feature disabled, the answer is +CME ERROR 3. Else it is +WOPEN: (0-4).

**Note:**

Refer to the document [Ref II] for an overview of the complete set of AT commands.

## 2.4 Notes on Memory Management

The Embedded software runs within an RTK task: the user must define the size of the customer application call stack.

The Wavecom Core Software and the Embedded Application manage their own RAM area. Any access from one of these programs to the other's RAM area is prohibited and causes a reboot.

In case an Embedded Application uses more than the maximum allocated RAM in global variables, or uses more than the maximum allocated ROM, then the behavior of the Embedded software becomes erratic.

Global variables, call stack and dynamic memory are all part of the RAM allocated to the Embedded Application.

The available memory sizes are :

For 16 Mbits flash size products ('A' WISMO module series):

- 256 Kbytes of ROM
- 32 Kbytes of RAM
- 5 Kbytes of Flash Object Data
- 0 Kbytes of Application & Data Storage Volume



For 32 Mbits flash size products ('B' WISMO module series):

- 512 Kbytes of ROM
- 128 Kbytes of RAM
- 128 Kbytes of Flash Object Data
- 512 Kbytes of Application & Data Storage Volume

## 2.5 Known Limitations

### 2.5.1 Command Pre-Parsing Limitation

In normal operating mode "command mode", the target serial link manager checks to see whether every command starts with "AT" and ends with a carriage return and end-of-line chars. Therefore, the only commands to be dispatched to the Embedded Application (in case of command pre-parsing subscription) are the ones complying with the here-above description.

Remark:

If you want to receive particular commands which are not AT commands (starting with another thing than "AT"), you can use the "data mode" to send and receive these commands into the Embedded Application (see the Flow Control Manager API).

### 2.5.2 Missing Unsolicited Messages in Remote Application

In Remote Application Execution mode, the application is started a few seconds after the Target. Therefore, some unsolicited events might be lost.

A pre-processor flag like `__REMOTETASKS__` can be used to add some specific code for remote mode.

## 2.6 Minimum Embedded Application Code

The following code must be included in any Embedded Application:

```
const wm_apmTask_t wm_apmTask [ WM_APM_MAX_TASK ] =
{
  { StackSize1, Stack1, InitFct1, ParserFct1 },
  { StackSize2, Stack2, InitFct2, ParserFct2 },
  { StackSize3, Stack3, InitFct3, ParserFct3 }
};
```

`stackX` and `stackSizeX` are variables used to define the application tasks call stack size (see § 3.2.4: "Stack Initialization").

**InitFctX()** are functions which are the first called ones for each Embedded Application task (see § 3.2.5: The Init Functions ).

**ParserFctX()** are functions which are called each time an Embedded Application task receives a message from the Wavecom Core Software (see § 3.2.6: "The Parser Functions").

## 2.7 Security

### 2.7.1 Software Security

Two software safeguards are used in the Open AT<sup>®</sup> platform:

- RAM access protection
- watchdog protection.

After reboot, the "**Init ()**" function of each task will have its parameter set to **WM\_APM\_REBOOT\_FROM\_EXCEPTION**.

After a reboot caused by a software crash, the application is started only 20 seconds after the start of the Wavecom Core software. This allows at least 20 seconds delay to re-download a new application.

In case of normal reboot, the application restarts immediately.

#### 2.7.1.1 RAM Access Protection

A specific RAM area is allocated to the Embedded Application.

The Embedded Application is seen as a Real-Time Task in the Wavecom software, and each time this task runs, the Wavecom RAM protection is activated.

If the Embedded Application tries to access this RAM, then an exception occurs and the software reboots.

In case of illegal RAM access, the Target Monitoring Tool screen displays: "**ARM exception 1 xxx,**" where "xxx" is the address the application was attempting to access.

If the symbol file is correctly configured in the Target Monitoring Tool (see document [Ref I]), then a Back Trace must describe the affected "C" functions in which the crash occurred.

#### 2.7.1.2 Watchdog Protection

The Embedded Application software is protected from reaching a dead-end lock by a 8 seconds watchdog.

In case of a crash, the software reboots.

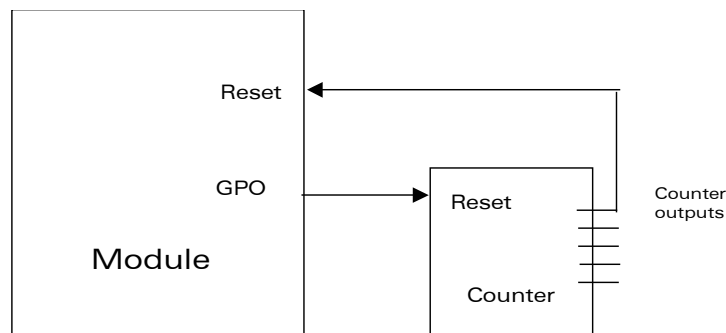
### **2.7.2 If an Embedded Application crash is detected, the Target Monitoring Tool screen displays: "Customer watchdog."Hardware Security**

Protection can also be improved using an external watchdog reset circuitry. With such a hardware watchdog protection, the Wavecom product will always be reset even in case of the software crashes.

To achieve this, one can use a GPO connected to a specific hardware counter that will reset the product if not refreshed.

For example, this specific hardware can be a counter with a specific counter output connected to the reset pin of the module, and the counter reset pin connected to a GPO.

In this way, the software in the module is supposed to reset the counter periodically. If not, the counter will increase until it reaches the specified limit and then resets the module.



**Figure 2: Reset hardware security example**

## 3 API

### 3.1 Data Types

The available data types are described in the `wm_types.h` file. They ensure compatibility with the data types used in the functional prototypes and are used for both Target and Visual C++ generation.

### 3.2 Mandatory Functions

The API described below includes a set of functions the Embedded software must supply and some mandatory variables the Embedded software must set. This API is located in the `wm_apm.h` file.

#### 3.2.1 Required Header

An Open AT<sup>®</sup> application must include the `wm_apm.h` header file. This file includes all other APIs' header files.

#### 3.2.2 Task identifiers

The several embedded application tasks are defined by identifiers, based on the following type :

```
typedef enum
{
    WM_OS_TASK_1,      // Task 1
    WM_OS_TASK_2,      // Task 2
    WM_OS_TASK_3,      // Task 3

    WM_OS_TASK_MAX,    // Maximum number of tasks
    WM_OS_TASK_WAVECOM=0xFF // for messages coming from Wavecom
                           // Core Software
} wm_osTask_e;
```

#### 3.2.3 Task table

The task table is used to define the embedded application tasks parameters, using the following type :

```
typedef struct
{
    u32 StackSize;           /* Stack Size (in bytes) */
    u32 Stack;               /* Stack pointer */
    s32 (*Init) ( wm_apmInitType_e ); /* Initialisation function */
    s32 (*Parser) ( wm_apmMsg_t * ); /* Parser function */
} wm_apmTask_t;
```

The table has to be defined by the application as below :

```
const wm_apmTask_t wm_apmTask [ WM_APM_MAX_TASK ] =
{
    { StackSize1, Stack1, Init1, Parser1 },
    { StackSize2, Stack2, Init2, Parser2 },
    { StackSize3, Stack3, Init3, Parser3 }
};
```

Note : to use less than 3 tasks, the additional tasks parameters must be set to 0 in the `wm_apmTask` table.

### 3.2.4 Stack Initialization

The following variables are used to define each task stack size :

```
#define StackSize1 1024 // The '1024' value is an example
#define StackSize2 1024 // The '1024' value is an example
#define StackSize3 1024 // The '1024' value is an example

u32 Stack1 [ StackSize1 / 4 ];
u32 Stack2 [ StackSize2 / 4 ];
u32 Stack3 [ StackSize3 / 4 ];
```

These data represent the amount of memory needed by each task call stack.

### 3.2.5 The Init Functions

The `Init` functions are called only once for each embedded application task during initialization.

Their prototype is:

```
s32 Init ( wm_apmInitType_e  InitType );
```

#### 3.2.5.1 Parameter

*InitType:*

Works out the item that triggered the initialization. The corresponding values are:

```
typedef enum
{
    WM_APM_POWER_ON, // normal Power On has occurred
    WM_APM_REBOOT_FROM_EXCEPTION, // the module has restarted after
                                // an exception.
    WM_APM_DOWNLOAD_SUCCESS, // an install process launched by the
                            // wm_adInstall API has succeeded.
    WM_APM_DOWNLOAD_ERROR // an install process launched by the
                        // wm_adInstall API has failed.
} wm_apmInitType_e;
```

The following events may cause an exception:

- a call to the `wm_osDebugFatalError()` function,
- unauthorized RAM access,
- a customer task watchdog.

#### 3.2.5.2 Return Value

The returned value is not relevant.

### 3.2.6 The Parser Functions

The Parser functions are called whenever a message is received by an embedded application task from the Wavecom Core Software.

Their prototype is:

```
s32 Parser ( wm_apmMsg_t * Message );
```

#### 3.2.6.1 Parameter

*Message:*

The *Message* structure depends on its type:

```
typedef struct
{
    s16          MsgTyp;    /* Type of the received message:
                           works out the associated structure of
                           the message body part*/

    wm_apmBody_t Body;     /* Specific message body */
} wm_apmMsg_t;
```

*MsgTyp* may have the following values:

MsgTyp value	Description
<i>WM_AT_RESPONSE</i>	the message includes an AT command response sent by the Embedded Application.
<i>WM_AT_UNSOLICITED</i>	the message includes an unsolicited AT response.
<i>WM_AT_INTERMEDIATE</i>	the message includes an intermediate AT response.
<i>WM_AT_CMD_PRE_PARSER</i>	the message includes an AT command sent by the External Application.
<i>WM_AT_RSP_PRE_PARSER</i>	the message includes a response processed by a Wavecom Core Software AT function.
<i>WM_OS_TIMER</i>	the message is sent when the timer expires.
<i>WM_OS_RELEASE_MEMORY</i>	the message includes the address of a released pointer.
<i>WM_FCM_RECEIVE_BLOCK</i>	the message includes data received by the Embedded Application.
<i>WM_FCM_OPEN_FLOW</i>	the requested flow opening operation is successful.
<i>WM_FCM_CLOSE_FLOW</i>	the requested flow closing operation is successful.
<i>WM_FCM_RESUME_DATA_FLOW</i>	the Embedded Application may resume its data sending operations.
<i>WM_IO_SERIAL_SWITCH_STATE_RSP</i>	includes the response to the serial link mode switching request.

The body structure is given hereafter:

```
typedef union
{
    /* Includes herein the different specific structures
    associated to MsgTyp */
    /* WM_AT_RESPONSE */
    wm_atResponse_t                ATResponse;
    /* WM_AT_UNSOLICITED */
    wm_atUnsolicited_t            ATUnsolicited;
    /* WM_AT_INTERMEDIATE */
    wm_atIntermediate_t          ATIntermediate;
    /* WM_AT_CMD_PRE_PARSER */
    wm_atCmdPreParser_t          ATCmdPreParser;
    /* WM_AT_RSP_PRE_PARSER */
    wm_atRspPreParser_t          ATRspPreParser
    /* WM_OS_TIMER */
    wm_osTimer_t                  OSTimer;
    /* WM_OS_RELEASE_MEMORY */
    wm_osRelease_t                OSRelease;
    /* WM_FCM_RECEIVE_BLOCK */
    wm_fcmReceiveBlock_t         FCMReceiveBlock;
    /* WM_FCM_OPEN_FLOW */
    wm_fcmOpenFlow_t             FCMOpenFlow
    /* WM_FCM_CLOSE_FLOW */
    wm_fcmFlow_e                 FCMCloseFlow
    /* WM_FCM_RESUME_DATA_FLOW */
    wm_fcmFlow_e                 FCMResumeFlow
    /* WM_IO_SERIAL_SWITCH_STATE_RSP */
    wm_ioSerialSwitchStateRsp_t  IOSerialSwitchStateRsp
} wm_apmBody_t;
```

The sub-structures of the message body are listed below:

**Body for WM\_AT\_RESPONSE:**

```
typedef struct {
    wm_atSendRspType_e Type;
    u16 StrLength; /* Length of StrData[] */
    ascii StrData[1]; /* AT response */
} wm_atResponse_t;

typedef enum {
    WM_AT_SEND_RSP_TO_EMBEDDED,
    WM_AT_SEND_RSP_TO_EXTERNAL,
    WM_AT_SEND_RSP_BROADCAST
} wm_atSendRspType_e;
```

(See § 3.3.2 for *wm\_atSendRspType\_e* description).



**Body for WM\_AT\_UNSOLICITED:**

```
typedef struct {
    wm_atUnsolicited_e Type;
    ul6 StrLength;
    ascii StrData[1];
} wm_atUnsolicited_t;
```

```
typedef enum {
    WM_AT_UNSOLICITED_TO_EXTERNAL,
    WM_AT_UNSOLICITED_TO_EMBEDDED,
    WM_AT_UNSOLICITED_BROADCAST
} wm_atUnsolicited_e;
```

(See § 3.3.3 for *wm\_atUnsolicited\_e* description).

**Body for WM\_AT\_INTERMEDIATE:**

```
typedef struct {
    wm_atIntermediate_e Type;
    ul6 StrLength;
    ascii StrData[1];
} wm_atIntermediate_t;
```

```
typedef enum {
    WM_AT_INTERMEDIATE_TO_EXTERNAL,
    WM_AT_INTERMEDIATE_TO_EMBEDDED,
    WM_AT_INTERMEDIATE_BROADCAST
} wm_atIntermediate_e;
```

(See § 3.3.4 for *wm\_atIntermediate\_e* description).

**Body for WM\_AT\_CMD\_PRE\_PARSER:**

```
typedef struct {
    wm_atCmdPreSubscribe_e Type;
    wm_ioPort_e Source;
    ul6 StrLength;
    ascii StrData[1];
} wm_atCmdPreParser_t;
```

```
typedef enum {
    WM_AT_CMD_PRE_WAVECOM_TREATMENT,
    WM_AT_CMD_PRE_EMBEDDED_TREATMENT,
    WM_AT_CMD_PRE_BROADCAST,
    WM_AT_CMD_PRE_APP_CONTROL_WAVECOM,
    WM_AT_CMD_PRE_APP_CONTROL_EMBEDDED
} wm_atCmdPreSubscribe_e;
```

(See § 3.3.5 for *wm\_atCmdPreSubscribe\_e* description).

**Body for WM\_AT\_RSP\_PRE\_PARSER:**

```
typedef struct {
    wm_atRspPreSubscribe_e Type;
    wm_ioPort_e             Dest;
    u16                    StrLength;
    ascii                  StrData[1];
} wm_atRspPreParser_t;

typedef enum {
    WM_AT_RSP_PRE_WAVECOM_TREATMENT, /* Default value */
    WM_AT_RSP_PRE_EMBEDDED_TREATMENT,
    WM_AT_RSP_PRE_BROADCAST
} wm_atRspPreSubscribe_e;
```

*Dest* field (based on following type), is the destination port where the response is to be sent.

```
typedef enum
{
    WM_IO_UART1,
    WM_IO_UART2,
    WM_IO_USB
} wm_ioPort_e;
```

(See § 3.3.6 for *wm\_atRspPreSubscribe\_e* description).

**Body for WM\_OS\_TIMER:**

```
typedef struct {
    u8      Ident;          /* Timer identifier */
} wm_osTimer_t;
```

(See § 3.4.2 for timer identifier description).

**Body for WM\_OS\_RELEASE\_MEMORY:**

```
typedef struct {
    void      *pMemoryBlock;
} wm_osRelease_t;
```

(See § 3.5.5 for this message description).

**Body for WM\_FCM\_RECEIVE\_BLOCK:**

```
typedef struct {
    u16      DataLength;    /* number of bytes received */
    u8      Reserved1[2];
    wm_fcmFlow_e FlowId;    /* IO flow ID */
    u8      Reserved2[7];
    u8      Data[1];       /* data received */
} wm_fcmReceiveBlock_t;
```

(See § 3.5.6 for *wm\_fcmReceiveBlock\_t* description and §3.5.2 for *wm\_fcmFlow\_e* description).

**Body for WM\_FCM\_OPEN\_FLOW:**

```
typedef struct      {
    wm_fcmFlow_e    FlowId;          /* opened IO flow ID */
    u16             DataMaxToSend; /* max length of sent data */
} wm_fcmOpenFlow_t;
```

(See § 3.5.3 for *wm\_fcmOpenFlow\_t* description and §3.5.2 for *wm\_fcmFlow\_e* description).

**Body for WM\_FCM\_CLOSE\_FLOW:**

(See §3.5.2 for *wm\_fcmFlow\_e* description).

**Body for WM\_FCM\_RESUME\_DATA\_FLOW:**

(See §3.5.2 for *wm\_fcmFlow\_e* description).

**Body for WM\_IO\_SERIAL\_SWITCH\_STATE\_RSP:**

```
typedef struct      {
    wm_ioSerialSwitchState_e  SerialMode; /* mode requested */
    s8                       RequestReturn; /* <0 means error */
} wm_ioSerialSwitchStateRsp_t;
```

(See § 3.6.2.1 for *wm\_ioSerialSwitchStateRsp\_t* description).

### 3.2.6.2 Return Values

The return parameter indicates whether the message has been taken into account (OK: 0) or not (ERROR: -1).

### 3.2.6.3 Notes

- any *StrData[]* or *Data[]* parameter present in the body sub-structure is automatically released at the end of the function.
- any *StrData[]* data is terminated by a 0x00 character and any associated *StrLength* includes the 0x00 character.

### 3.3 AT Command API

#### 3.3.1 Required Header

This API is defined in `wm_at.h` header file.  
This file is included by `wm_apm.h`.

#### 3.3.2 The `wm_atSendCommand` Function

The `wm_atSendCommand` function sends AT commands.

Its prototype is:

```
void wm_atSendCommand (    u16                AtStringSize,  
                          wm_atSendRspType_e ResponseType,  
                          ascii          *AtString );
```

##### 3.3.2.1 Parameters

###### **AtString**

Any AT command string in ASCII character (terminated by a 0x00).  
Several strings can be sent at the same time, depending on the type of AT command.

###### **AtStringSize**

Size of the previous parameter, *AtString*. It equals the length + 1 and includes the 0x00 character.

###### *ResponseType:*

Indicates which application receives the AT responses. The corresponding values are:

```
typedef enum {  
    WM_AT_SEND_RSP_TO_EMBEDDED, /* Default value */  
    WM_AT_SEND_RSP_TO_EXTERNAL,  
    WM_AT_SEND_RSP_BROADCAST  
} wm_atSendRspType_e;
```

`WM_AT_SEND_RSP_TO_EMBEDDED` means that all the AT responses will be sent back to the Embedded Application (default mode).

`WM_AT_SEND_RSP_TO_EXTERNAL` means that all the AT responses will be sent back to the External Application (PC).

`WM_AT_SEND_RSP_BROADCAST` means that all the AT responses will be broadcasted to both the Embedded and External Applications (PC).

This parameter's four MSBs indicate also on which serial port the responses have to be sent, in `WM_AT_SEND_RSP_TO_EXTERNAL` or `WM_AT_SEND_RSP_BROADCAST` mode. By default, the responses are sent on UART 1. To use another port, the following type should be used :

```
typedef enum
{
    WM_IO_UART1,
    WM_IO_UART2,
    WM_IO_USB
} wm_ioPort_e;
```

For example, responses will be sent on the UART 2 when `ResponseType` parameter is set to:

```
WM_AT_SEND_RSP_TO_EXTERNAL | (WM_IO_UART2 << 4)
```

This serial port setting is also used to select the current UART in order to set-up specific parameters (speed with AT+IPR, character framing with AT+ICF, etc...).

### 3.3.2.2 Notes

- ❑ As described in the "AT Commands Interface" document, AT commands sent by `wm_atSendCommand()` begin with the "AT" string, and end with a "\r" character (carriage return), except in some cases ("A" command, SMS writing commands ("test\x1A"), ...)
- ❑ AT Command responses are received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with the `MsgTyp` parameter set to `WM_AT_RESPONSE`.
- ❑ A response sent to an External Application cannot be pre-parsed (see §3.3.6). If an Embedded Application wants to filter or spy the response, it must set the `ResponseType` parameter to `WM_AT_SEND_RSP_TO_EMBEDDED` or `WM_AT_SEND_RSP_BROADCAST`.

### 3.3.2.3 Example: Sending AT Commands and Receiving the Corresponding Responses

The Embedded Application sends an AT command and receives the response from the AT functionality of Wavecom Core Software using The `wm_atSendCommand` and The `wm_atSendRspExternalApp` functions.

- ❑ Example of sending an AT command:

```
wm_atSendCommand( 16, WM_AT_SEND_RSP_TO_EMBEDDED,
"ATD0146290800\r" );
```

- Example of receiving an AT response:

```

s32 wm_apmAppliParser (wm_apmMsg_t * Message)
{
    ascii *    strBuffer;
    ul6      nLenBuffer;

    switch (Message->MsgTyp)
    {
        ...
        case WM_AT_SEND_RSP:

            strBuffer    = &(amp;Message->Body.AT_Response.StrData);
            nLenBuffer = Message->Body. AT_Response.StrLength;

            /* Receive AT response for filtering */
            if (Message->Body.ATResponse.Type == AT_RESPONSE_TO_EMBEDDED)
            {
                if (wm_strnicmp(strBuffer, "CONNECT", 7) == 0)
                {
                    /* Local processing */
                    ...
                    wm_atSendRspExternalApp("CONNECT\r", 9);
                }
                else
                {
                    /* Don't modify other responses */
                    wm_atSendRspExternalApp ( wm_strlen(strBuffer),
                                                strBuffer);
                }
            }
            /* Receive AT response for spying */
            else if (Message->Body.ATResponse.Type ==
                    WM_AT_SEND_RSP_BROADCAST)
            { ...
            }
            /* ERROR */
            else
            { ..
            }
            ...
        }
        return OK;
    }
}
    
```

### 3.3.3 The `wm_atUnsolicitedSubscription` Function

If the Embedded Application wants to receive an unsolicited AT response (incoming call, etc.), the `wm_atUnsolicitedSubscription` function is used to subscribe to the corresponding service.

Its prototype is:

```
void wm_atUnsolicitedSubscription (  
    wm_atUnsolicited_e  Unsolicited);
```

#### 3.3.3.1 Parameter

*Unsolicited:*

Indicates which application receives the unsolicited AT response. The corresponding values are:

```
typedef enum {  
    WM_AT_UNSOLICITED_TO_EXTERNAL, /* Default value */  
    WM_AT_UNSOLICITED_TO_EMBEDDED,  
    WM_AT_UNSOLICITED_BROADCAST  
} wm_atUnsolicited_e;
```

WM\_AT\_UNSOLICITED\_TO\_EXTERNAL means any unsolicited AT response will be sent back to the External Application (PC). This is the default mode.

WM\_AT\_UNSOLICITED\_TO\_EMBEDDED means any unsolicited AT response will be sent back to the Embedded Application.

WM\_AT\_UNSOLICITED\_BROADCAST means any unsolicited AT response will be broadcast to both the Embedded and External Applications (PC).

#### 3.3.3.2 Note

An unsolicited AT response is received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with *MsgTyp* parameter set to WM\_AT\_UNSOLICITED.

### 3.3.3.3 Example: Receiving Unsolicited AT Responses

The following example deals with The `wm_atUnsolicitedSubscription` function.

The two stages used to receive unsolicited AT responses are:

- 1) Subscribing to an Embedded Application to receive unsolicited AT responses. Three types of subscriptions are available:
  - default (`WM_AT_UNSOLICITED_TO_EXTERNAL`),
  - filtering (`WM_AT_UNSOLICITED_TO_EMBEDDED`) and
  - spying (`WM_AT_UNSOLICITED_BROADCAST`).

An example of a filter subscription is given below:

```
/* Unsolicited responses are process by Embedded Application */  
wm_atUnsolicitedSubscription (WM_AT_UNSOLICITED_TO_EMBEDDED);
```

- 2) Receiving unsolicited AT responses:

```
s32 wm_apmAppliParser (wm_apmMsg_t * Message)  
{  
    ascii *    strBuffer;  
    u16    nLenBuffer;  
  
    switch (Message->MsgTyp)  
    {  
        ...  
        case WM_AT_UNSOLICITED:  
            strBuffer    = &(Message->Body.ATUnsolicited.StrData);  
            nLenBuffer = Message->Body.ATUnsolicited.StrLength;  
  
            /* Process unsolicited AT response for filtering */  
            if (Message->Body.ATUnsolicited.Type ==  
                WM_AT_UNSOLICITED_TO_EMBEDDED)  
            {  
                /* Embedded processings */  
            }  
  
            /* Process unsolicited AT response for spying */  
            else if (Message->Body.ATUnsolicited.Type ==  
                WM_AT_UNSOLICITED_BROADCAST)  
            {  
                /* Embedded processings */  
            }  
  
        ...  
    }  
    return OK;  
}
```



### 3.3.4 The `wm_atIntermediateSubscription` Function

If the Embedded Application wants to receive an intermediate AT response (alerting the remote party during a mobile-originated call, SMS reading responses, etc.), the `wm_atIntermediateSubscription` function is used to subscribe to the corresponding service.

Its prototype is:

```
void wm_atIntermediateSubscription (
    wm_atIntermediate_e Intermediate );
```

#### 3.3.4.1 Parameter

*Intermediate:*

Indicates which application receives the intermediate AT response. The corresponding values are:

```
typedef enum {
    WM_AT_INTERMEDIATE_TO_EXTERNAL, /* Default value */
    WM_AT_INTERMEDIATE_TO_EMBEDDED,
    WM_AT_INTERMEDIATE_BROADCAST
} wm_atIntermediate_e;
```

`WM_AT_INTERMEDIATE_TO_EXTERNAL` means any intermediate AT response will be sent back to the External Application (PC). This is the default mode.

`WM_AT_INTERMEDIATE_TO_EMBEDDED` means any intermediate AT response will be sent back to the Embedded Application.

`WM_AT_INTERMEDIATE_BROADCAST` means any intermediate AT response will be broadcasted to both the Embedded and External Applications (PC).

#### 3.3.4.2 Note

An intermediate AT response is received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with `MsgTyp` parameter set to `WM_AT_INTERMEDIATE`.

### 3.3.4.3 Example: Receiving Intermediate AT Responses

The following example deals with the `wm_atIntermediateSubscription` function. The two stages which are used to receive intermediate AT responses are:

- 1) Subscribing to an Embedded Application to receive intermediate AT responses. Three types of subscriptions are available: default (`WM_AT_INTERMEDIATE_TO_EXTERNAL`), filtering (`WM_AT_INTERMEDIATE_TO_EMBEDDED`) and spying (`WM_AT_INTERMEDIATE_BROADCAST`).

An example of a filter subscription is given below:

```
/* Intermediate responses are processed by Embedded Application
*/
wm_atIntermediateSubscription (WM_AT_INTERMEDIATE_TO_EMBEDDED);
```

- 2) Receiving intermediate AT responses:

```
s32 wm_apmAppliParser (wm_apmMsg_t * Message)
{
    ascii *    strBuffer;
    ul6      nLenBuffer;

    switch (Message->MsgTyp)
    {
        ...
        case WM_AT_INTERMEDIATE:
            strBuffer    = &(amp;Message->Body.ATIntermediate.StrData);
            nLenBuffer = Message->Body.ATIntermediate.StrLength;

            /* Process intermediate AT response for filtering */
            if (Message->Body.ATIntermediate.Type ==
                WM_AT_INTERMEDIATE_TO_EMBEDDED)
            {
                /* Embedded processing */
            }

            /* Process intermediate AT response for spying */
            else if (Message->Body.ATIntermediate.Type ==
                    WM_AT_INTERMEDIATE_BROADCAST)
            {
                /* Embedded processing */
            }

            ...
        }
    }
    return OK;
}
```

### 3.3.5 The `wm_atCmdPreParserSubscribe` Function

If the Embedded Application wants to perform AT command pre-parsing, it should then subscribe to the corresponding services, using the `wm_atCmdPreParserSubscribe` function.

The AT messages received from the External Application are forwarded to the Pre-parser and sent to the Embedded Application through a `WM_AT_CMD_PRE_PARSER` type message, of which the associated structure is `wm_atCmdPreParser_t`.

Note that, by default, the "AT+WDWL" and "AT+WOPEN" AT commands can not be pre-parsed, so that the User can download a new Embedded software or stop the embedded application whenever he wants.

The `wm_atCmdPreParserSubscribe` function may also be used to pre-parse these commands, using the `WM_AT_CMD_PRE_APP_CONTROL_EMBEDDED` option.

The prototype of this function is:

```
void wm_atCmdPreParserSubscribe (
    wm_atCmdPreSubscribe_e  SubscribeType);
```

#### 3.3.5.1 Parameter

##### *SubscribeType:*

Indicates what happens when an AT command arrives. The corresponding values are:

```
typedef enum          {
    WM_AT_CMD_PRE_WAVECOM_TREATMENT, /* Default value */
    WM_AT_CMD_PRE_EMBEDDED_TREATMENT,
    WM_AT_CMD_PRE_BROADCAST,

    /* Open AT control commands processing */
    WM_AT_CMD_PRE_APP_CONTROL_WAVECOM, /* Default value */
    WM_AT_CMD_PRE_APP_CONTROL_EMBEDDED
} wm_atCmdPreSubscribe_e;
```

`WM_AT_CMD_PRE_WAVECOM_TREATMENT` means the Embedded Application does not want to filter or spy the commands sent by an External Application (default mode).

`WM_AT_CMD_PRE_EMBEDDED_TREATMENT` means the Embedded Application wants to filter the AT commands sent by an External Application.

`WM_AT_CMD_PRE_BROADCAST` means the Embedded Application wants to spy the AT commands sent by an External Application.

`WM_AT_CMD_PRE_APP_CONTROL_WAVECOM` means the +WOPEN and +WDWL commands are always processed by the Wavecom core software ; they can not be filtered by the embedded application (default mode)

`WM_AT_CMD_PRE_APP_CONTROL_EMBEDDED` means +WDWL and +WOPEN commands are processed as other ones, and may be pre-parsed by the embedded application.

### 3.3.5.2 Notes

- Filtered or spied AT commands are received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with the *MsgTyp* parameter set to `WM_AT_CMD_PRE_PARSER`. The source member of the message is the port on which the command has been received.
- 
- The Embedded Application will process the received command and, for instance, will send it back either completely or not to the `wm_atSendCommand()` function. Therefore, the responses may be forwarded to the Wavecom Core Software.
- When a command is pre-parsed for filtering, the User has the responsibility to send the response to the External Application.
- When +WDWL or +WOPEN commands are pre-arsed for filtering, the application has the responsibility to maintain an interface for other applications download and Open AT<sup>®</sup> start/stop mode. For exemple, it should filter +WDWL or +WOPEN command and require a password for download or application stop.

### 3.3.5.3 Example: Filtering or Spying AT Commands Sent by an External Application

The following example deals with the `wm_atCmdPreParserSubscribe()` function.

The two stages which are used to filter or spy AT commands sent by an External Application are:

- 1) Subscribing to a command pre-parsing mechanism to filter or spy the AT commands sent by the External Application.

An example of a filtering subscription is given below:

```
/* Filter subscription */  
wm_atCmdPreParserSubscribe (WM_AT_CMD_PRE_EMBEDDED_TREATMENT);
```

An example of a spying subscription is given below:

```
/* Spy subscription */  
wm_atCmdPreParserSubscribe(WM_AT_CMD_PRE_BROADCAST);
```

29th November 2005

- 2) Receiving and processing the pre-parsed commands (an AT command sent by the External Application) in the Embedded Application:

```

s32 wm_apmAppliParser (wm_apmMsg_t * Message)
{
    ascii *    strBuffer;
    u16      nLenBuffer;

    switch (Message->MsgTyp)
    {
        ...
        case WM_AT_CMD_PRE_PARSER:
            strBuffer    = &(Message->Body.ATCmdPreParser.StrData);
            nLenBuffer = Message->Body. ATCmdPreParser.StrLength;

            /* Process pre-parsed AT command for filtering */
            if (Message->Body.ATCmdPreParser.Type ==
                WM_AT_CMD_PRE_EMBEDDED_TREATMENT)
            {
                /* Filtering Embedded processings */
                ...
            }
            else if (Message->Body.ATCmdPreParser.Type ==
                WM_AT_CMD_PRE_BROADCAST)
            {
                /* Spying Embedded processing */
                ...
            }
            ...
        }
        return OK;
    }
}
    
```

### 3.3.6 The `wm_atRspPreParserSubscribe` Function

If the Embedded Application wants to perform an AT response pre-parsing, it should then subscribe to the corresponding services, using the `wm_atRspPreParserSubscribe` function.

An AT message sent by an External Application and processed by the Wavecom Core Software generates a response. Depending on the subscription type, this response may be forwarded to the Embedded Application through a message of the `WM_AT_RSP_PRE_PARSER` type of which the associated structure is `wm_atRspPreParser_t`.

Its prototype is:

```

void wm_atRspPreParserSubscribe (
    wm_atRspPreSubscribe_e    SubscribeType );
    
```

### 3.3.6.1 Parameter

#### *SubscribeType:*

Indicates what happens when an AT response arrives. The corresponding values are as follows:

```
typedef enum          {  
    WM_AT_RSP_PRE_WAVECOM_TREATMENT, /* Default value */  
    WM_AT_RSP_PRE_EMBEDDED_TREATMENT,  
    WM_AT_RSP_PRE_BROADCAST  
} wm_atRspPreSubscribe_e;
```

WM\_AT\_RSP\_PRE\_WAVECOM\_TREATMENT means the Embedded Application does not want to filter or spy the responses sent to an External Application (default mode).

WM\_AT\_RSP\_PRE\_EMBEDDED\_TREATMENT means the Embedded Application wants to filter the AT responses sent to an External Application.

WM\_AT\_RSP\_PRE\_BROADCAST means the Embedded Application wants to spy the AT responses sent to an External Application.

### 3.3.6.2 Notes

- Filtered or spied AT responses are received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with the *MsgTyp* parameter set to WM\_AT\_RSP\_PRE\_PARSER.
- If the Embedded Application subscribes to WM\_AT\_RSP\_PRE\_EMBEDDED\_TREATMENT, it will process the response and send it to the External Application, using the `wm_atSendRspExternalApp()` function (see § 3.3.7).
- The response pre-parser will only be active if the AT command has not been sent through `wm_atSendCommand()`. In this case, the response is processed as described in the *ResponseType* parameter (see § 3.3.2).

### 3.3.6.3 Example: Filtering or Spying AT Responses Sent to the External Application

The following example deals with the `wm_atRspPreParserSubscribe()` function.

The two stages used to filter or spy the AT response sent to the External Application are:

- 1) Subscribing to the response pre-parsing mechanism in order to filter or spy the AT response sent to the External Application.

An example of a filter subscription is given below:

```
/* Filter subscription */  
wm_atRspPreParserSubscribe (WM_AT_RSP_PRE_EMBEDDED_TREATMENT);
```

An example of a spying subscription is given below:

```
/* Spy subscription */  
wm_atRspPreParserSubscribe(WM_AT_RSP_PRE_BROADCAST);
```

- 2) Processing the pre-parsed response in the Embedded Application:

```
s32 wm_apmAppliParser (wm_apmMsg_t * Message)  
{  
    ascii * strBuffer;  
    ul6     nLenBuffer;  
  
    switch (Message->MsgTyp)  
    {  
        ...  
        case WM_AT_RSP_PRE_PARSER:  
            strBuffer = &(Message->Body.ATRspPreParser.StrData);  
            nLenBuffer = Message->Body.ATRspPreParser.StrLength;  
  
            /* Process pre-parsed AT command for filtering */  
            if(Message->Body.ATRspPreParser.Type ==  
                WM_AT_RSP_PRE_EMBEDDED_TREATMENT)  
            {  
                /* Filtering Embedded processing */  
                ...  
            }  
            else if (Message->Body.ATRspPreParser.Type ==  
                WM_AT_RSP_PRE_BROADCAST) {  
                /* Spying Embedded processing */  
                ...  
            }  
        ...  
    }  
    return OK;  
}
```

### 3.3.7 The `wm_atSendRspExternalApp` Function

The `wm_atSendRspExternalApp` function sends an AT response to the External Application, in case of AT command pre-parsing.

The response is sent to the UART 1 port.

Its prototype is:

```
void wm_atSendRspExternalApp (    u16      AtStringSize,  
                                ascii      *AtString );
```

#### 3.3.7.1 Parameters

*AtString*:

Any AT response string in ASCII characters (terminated by a 0x00 character). This string is sent on the serial link without any change : it should also include “\r\n” characters at the end and/or at the beginning of the string.

*AtStringSize*:

Size of the previous *AtString* parameter. It equals the length + 1 since it includes the 0x00 character.

#### 3.3.7.2 Notes

- This function should be used to transmit to the External Application the responses received by the Embedded Application through the WM\_AT\_RESPONSE message.

### 3.3.8 The `wm_atSendRspExternalAppExt` Function

The `wm_atSendRspExternalAppExt` function sends an AT response to the External Application, in case of AT command pre-parsing.

The response is sent to the required port.

Its prototype is:

```
void wm_atSendRspExternalAppExt ( u16      AtStringSize,  
                                 ascii      *AtString,  
                                 wm_ioPort_e Dest );
```

**Note:** This function should be used to transmit to the External Application the responses received by the Embedded Application through WM\_AT\_RESPONSE message.



### 3.3.8.1 Parameters

**AtString**

Any AT response string in ASCII characters (terminated by a 0x00 character). This string is sent on the serial link without any change : it should also include “\r\n” characters at the end and/or at the beginning of the string.

**AtStringSize**

Size of the previous *AtString* parameter. It equals the length + 1 since it includes the 0x00 character.

**Dest**

Port where to send the provided response, using following type :

```
typedef enum
{
    WM_IO_UART1,
    WM_IO_UART2,
    WM_IO_USB
} wm_ioPort_e;
```

### 3.3.9 The **wm\_atSendUnsolicitedExternalApp** Function

The **wm\_atSendUnsolicitedExternalApp** function sends an AT unsolicited response to the External Application.

The Unsolicited response will be sent to all ports.

Its prototype is:

```
void wm_atSendUnsolicitedExternalApp (    u16      AtStringSize,
                                           ascii    *AtString );
```

### 3.3.9.1 Parameters

**AtString**

Any AT unsolicited response string in ASCII characters (terminated by a 0x00 character). This string is sent on the serial link without any change : it should also include “\r\n” characters at the end and/or at the beginning of the string.

**AtStringSize**

Size of the previous *AtString* parameter. It equals the length + 1 since it includes the 0x00 character.

### 3.3.9.2 Notes

- ❑ An unsolicited response string sent by the `wm_atSendUnsolicitedExternalApp` function will only be displayed on the serial link when the Wavecom AT task is not busy by a command processing. If it is busy in a such processing, the unsolicited response string is stored, and displayed at the end of the process (after the terminal AT response).
- ❑ Sending an AT response by the `wm_atSendRspExternalApp` function will display all previously stored unsolicited responses (after this response display).
- ❑ This function should be used to transmit to the External Application the unsolicited responses received by the Embedded Application through the WM\_AT\_UNSOLICITED message.

### 3.3.10 The `wm_atSendIntermediateExternalApp` Function

The `wm_atSendIntermediateExternalApp` function sends an AT intermediate response to the External Application.

The intermediate response will be sent to UART 1 port.

Its prototype is:

```
void wm_atSendIntermediateExternalApp (    u16      AtStringSize,  
                                           ascii    *AtString );
```

#### 3.3.10.1 Parameters

*AtString:*

Any AT intermediate response string in ASCII characters (terminated by a 0x00 character). This string is sent on the serial link without any change : it should also include “\r\n” characters at the end and/or at the beginning of the string.

*AtStringSize:*

Size of the previous *AtString* parameter. It equals the length + 1 and includes the 0x00 character.

#### 3.3.10.2 Notes

- ❑ An intermediate response string sent by the `wm_atSendIntermediateExternalApp` function will always display this string on the serial link, either the Wavecom AT task is busy on a command processing or not.
- ❑ Previously stored unsolicited responses will not be displayed after a call to the `wm_atSendIntermediateExternalApp` function.
- ❑ This function should be used to transmit to the External Application the intermediate responses received by the Embedded Application through the WM\_AT\_INTERMEDIATE message.

### 3.3.11 The `wm_atSendIntermediateExternalAppExt` Function

The `wm_atSendIntermediateExternalApp` function sends an AT intermediate response to the External Application.

The intermediate response will be sent to the required port.

Its prototype is:

```
void wm_atSendIntermediateExternalAppExt
    (u16          AtStringSize,
     ascii       *AtString,
     wm_ioPort_e Dest );
```

#### 3.3.11.1 Parameters

##### **AtString**

Any AT intermediate response string in ASCII characters (terminated by a 0x00 character). This string is sent on the serial link without any change : it should also include “\r\n” characters at the end and/or at the beginning of the string.

##### **AtStringSize**

Size of the previous *AtString* parameter. It equals the length + 1 and includes the 0x00 character.

##### **Dest**

Port where to send the provided intermediate response, using following type :

```
typedef enum
{
    WM_IO_UART1,
    WM_IO_UART2,
    WM_IO_USB
} wm_ioPort_e;
```

#### 3.3.11.2 Notes

- An intermediate response string sent by the `wm_atSendIntermediateExternalApp` function will always display this string on the serial link, either the Wavecom AT task is busy on a command processing or not.
- Previously stored unsolicited responses will not be displayed after a call to the `wm_atSendIntermediateExternalApp` function.
- This function should be used to transmit to the External Application the intermediate responses received by the Embedded Application through the WM\_AT\_INTERMEDIATE message.

## 3.4 OS API

### 3.4.1 Required Header

This API is defined in `wm_os.h` header file.  
This file is included by `wm_apm.h`.

### 3.4.2 The `wm_osStartTimer` Function

The `wm_osStartTimer` function sets up a timer (in 100ms steps) associated to an existing *TimerId*.

Its prototype is:

```
s32  wm_osStartTimer    ( u8      TimerId,  
                        bool     bCyclic,  
                        u32     TimerValue );
```

#### 3.4.2.1 Parameters

*TimerId*:

Timer identifier: the range 0 to `WM_OS_MAX_TIMER_ID` is accepted.

*BCyclic*:

This parameter may have one of the following values:

- TRUE**: the timer is cyclic and is automatically set up when a cycle is over,
- FALSE**: the timer has only one cycle.

*TimerValue*:

Number of timer units (the timer unit is 100 ms).

#### 3.4.2.2 Return Values

The return parameter is positive or null if the timer is successfully set up and negative in case of failure.

#### 3.4.2.3 Notes

- The timer expiry indication is received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with the *MsgTyp* parameter set to `WM_OS_TIMER`.
- Since the WAVECOM products time granularity is 18.5 ms, the 100 ms steps are emulated, reaching a value as close as possible to the requested one (modulo 18.5). For example, if a `20 * 100ms` timer is required, the real time value will be 1998 ms (`108 * 18.5ms`).

#### 3.4.2.4 Example: Managing a Timer

The range 0 to WM\_OS\_MAX\_TIMER\_ID is accepted for the timer Id. A timer may or may not be cyclic.

An example of setting up a timer is given below:

```
/* Timer start, not cyclic, value = 1second */  
wm_osStartTimer( 1, FALSE, 10 );
```

An example of receiving a timer expiry event is given below:

```
s32 wm_apmAppliParser (wm_apmMsg_t * Message)  
{  
    ascii * strBuffer;  
    u16     nLenBuffer;  
  
    switch (Message->MsgTyp)  
    {  
        ...  
        case WM_OS_TIMER:  
  
        ...  
    }  
    return OK;  
}
```

#### 3.4.3 The wm\_osStopTimer Function

The wm\_osStopTimer function stops the timer identified by TimerId.

Its prototype is:

```
s32 wm_osStopTimer ( u8 TimerId );
```

##### 3.4.3.1 Parameter

*TimerId:*

Timer identifier : the range 0 to WM\_OS\_MAX\_TIMER\_ID is accepted.

##### 3.4.3.2 Return Values

The return parameter is the remaining time (in 100 ms steps) if the timer was still running, and a negative value otherwise.

#### 3.4.4 The wm\_osStartTickTimer Function

The wm\_osStartTickTimer function sets up a timer (in 18.5 ms ticks steps) associated to an existing *TimerId*.

Its prototype is:

```
s32 wm_osStartTickTimer ( u8 TimerId,  
                          bool bCyclic,  
                          u32 TimerValue );
```

### 3.4.4.1 Parameters

*TimerId:*

Timer identifier: the range 0 to WM\_OS\_MAX\_TIMER\_ID is accepted.

*BCyclic:*

This parameter may have one of the following values:

- TRUE:** the timer is cyclic and is automatically set up when a cycle is over,
- FALSE:** the timer has only one cycle.

*TimerValue:*

Number of ticks (18.5 ms steps).

### 3.4.4.2 Return Values

The return parameter is positive or null if the timer is successfully set up and negative if not.

### 3.4.4.3 Note

The timer expiry indication is received by the Embedded Application through a message. This message is available as a parameter of the `wm_apmAppliParser()` function with the *MsgTyp* parameter set to WM\_OS\_TIMER.

### 3.4.4.4 Example: Managing a Timer

The range 0 to WM\_OS\_MAX\_TIMER\_ID is accepted. A timer may or may not be cyclic.

An example of setting up a timer is given below:

```
/* Timer start, not cyclic, value = 37 ms */  
wm_osStartTickTimer( 1, FALSE, 2 );
```

### 3.4.5 The `wm_osStopTickTimer` Function

The `wm_osStopTickTimer` function stops the timer identified by `TimerId`.

Its prototype is:

```
s32 wm_osStopTimer ( u8 TimerId );
```

#### 3.4.5.1 Parameter

*TimerId:*

Timer identifier: the range 0 to WM\_OS\_MAX\_TIMER\_ID is accepted.

#### 3.4.5.2 Return Values

The return parameter is the remaining time (in 18.5 ms tick steps) if the timer was still running, and a negative value otherwise.

### 3.4.6 The `wm_osDebugTrace` Function

The `wm_osDebugTrace` function is aimed at trace managing.

Its prototype is:

```
s32 wm_osDebugTrace ( u8 Level, ascii *Format, ... );
```

#### 3.4.6.1 Parameters

*Level:*

Used to differentiate the traces. The PC trace software gives access to level configuration.

*Format:*

Used to specify a string and the corresponding formats (like the `printf` function), as far as the data to trace is concerned. The supported formats are `'%c'`, `'%x'`, `'%X'`, `'%u'`, `'%d'`.

Up to 6 parameters may be included in the *Format* string.

As the `'%s'` format is not supported, the way to display an `ascii *` string is to replace the *Format* string by this char, without any other parameters.

*....:*

Represents the list of data to be traced.

#### 3.4.6.2 Returned values

A positive or null value indicates that the trace has been sent ; otherwise a negative error value is sent.

#### 3.4.6.3 Example: Inserting Debug Information

Debug information is included in the Embedded Application, and therefore it uses ROM space and CPU resources.

The Target Monitoring Tool is used to display the Debug information.

An example of tracing an informational message is given below:

```
wm_osDebugTrace ( 1, "This is an informational message on level  
1" );  
/* To visualise this, the Target Monitoring Tool must be  
configured to extract level 1 traces */  
  
/* The result string using the Target Monitoring Tool should  
be:  
"This is an informational message on level 1" */
```

Example of tracing an informational message using a decimal parameter:

```
u8 param =12;

wm_osDebugTrace ( 2, "This is an informational message on level
2 with 1 parameter =%d", param );
/* To visualise this, the Target Monitoring Tool must be
configured to extract level 2 traces */

/* The result string using the Target Monitoring Tool should
be:
"This is an informational message on level 2 with 1 parameter
=12" */
```

Example of tracing a string:

```
ascii String[]="Hello World";

wm_osDebugTrace ( 3, String );
/* To visualise this, the Target Monitoring Tool must be
configured to extract level 3 traces */

/* The result string on Target Monitoring Tool should be:
"Hello World" */
```

### 3.4.7 The **wm\_osDebugFatalError** Function

The **wm\_osDebugFatalError** function is the fatal error handling function: it stores the error code and then performs a reset of the product.

Its prototype is:

```
s32  wm_osDebugFatalError ( ascii * Message );
```

#### 3.4.7.1 Parameters

*Message:*

String to be displayed whenever an error occurs.

#### 3.4.7.2 Returned Value

A negative error value indicates that the fatal error did not happen.

#### 3.4.7.3 Note

The reboot is performed after the call to the fatal error function. In order to ensure the downloading of a new binary file after a fatal error has been detected, the User software startup is delayed 20 sec.

Therefore, in order not to miss any event, the application has to handle a startup delay of 20 sec.



### 3.4.8 Important Note on Data Flash Management

An embedded application cannot use more than following sizes, according to product type:

- 5KB on 16 Mbits flash size products.
- 128 KB on 32 Mbits flash size products

A single flash object may use up to 30 Kbytes data.

The identifiers use an u16 value: any value from 0 to 0xFFFF is valid for an object identifier.

However, due to the internal storage implementation, only up to 2000 object identifiers can exist at the same time.

### 3.4.9 The `wm_osWriteFlashData` Function

The `wm_osWriteFlashData` function is used to write data into Flash ROM. The corresponding identifier is assigned to the stored data.

The prototype of this function is:

```
s32 wm_osWriteFlashData ( u16 Id, u16 DataLen, u8 *Data );
```

#### 3.4.9.1 Parameters

*Id:*

Identifier assigned to the stored data.

*DataLen:*

Length of the data to be stored (in bytes).

*Data:*

Pointer to the data to be stored.

#### 3.4.9.2 Return Values

OK on success

ERROR if :

- There is no more free space
- The object size exceeds 30 Kbytes
- There is no more free identifier (2000 objects limit reached)

### 3.4.10 The **wm\_osReadFlashData** Function

The **wm\_osReadFlashData** function is used to read data identified by *Id* from the Flash ROM.

Its prototype is:

```
s32  wm_osReadFlashData ( u16  Id, u16  DataLen, u8  *Data );
```

#### 3.4.10.1 Parameters

*Id*:

Identifier assigned to the stored data.

*DataLen*:

Length of the data to be read (in bytes).

*Data*:

Pointer to the data to be read.

#### 3.4.10.2 Return Values

The return parameter is the length to read and copied to *\*Data* on success, or ERROR if the object does not exist.

### 3.4.11 The **wm\_osGetLenFlashData** Function

The **wm\_osGetLenFlashData** function supplies the length of the data stored in Flash ROM and identified by *Id*.

Its prototype is:

```
s32  wm_osGetLenFlashData ( u16  Id );
```

#### 3.4.11.1 Parameter

*Id*:

Identifier assigned to the stored data.

#### 3.4.11.2 Return Values

The return parameter is the byte length of the data identified by *Id*, or ERROR if the object does not exist.

### 3.4.12 The **wm\_osDeleteFlashData** Function

The **wm\_osDeleteFlashData** function deletes the data stored in Flash ROM and identified by *Id*.

Its prototype is:

```
s32 wm_osDeleteFlashData ( u16 Id );
```

#### 3.4.12.1 Parameter

*Id*:

Identifier assigned to the stored data.

#### 3.4.12.2 Return Values

The return value is OK on success, ERROR if the object does not exist.

### 3.4.13 The **wm\_osGetAllowedMemoryFlashData** Function

The **wm\_osGetAllowedMemoryFlashData** function returns the quantity of allocated memory in Flash ROM.

Its prototype is:

```
s32 wm_osGetAllowedMemoryFlashData ( void );
```

The return parameter is the quantity of allocated memory in Flash ROM (Unit : bytes).

### 3.4.14 The **wm\_osGetFreeMemoryFlashData** Function

The **wm\_osGetFreeMemoryFlashData** function returns the quantity of available memory in Flash ROM.

Its prototype is:

```
s32 wm_osGetFreeMemoryFlashData ( void );
```

The return parameter is the quantity of free memory (expressed in bytes) in Flash ROM.

### 3.4.15 The **wm\_osGetUsedMemoryFlashData** Function

The **wm\_osGetUsedMemoryFlashData** function returns the quantity of used memory by flash objects between the provided start & end IDs.

Its prototype is:

```
s32 wm_osGetUsedMemoryFlashData ( u16 StartId, u16 EndId );
```

#### 3.4.15.1 Parameters

*StartId:*

Range to browse first Id

*EndId:*

Range to browse last Id

#### 3.4.15.2 Return Values

The return parameter is the quantity of used memory (expressed in bytes) by the provided Id range in Flash ROM, or ERROR if StartId is greater than EndId.

### 3.4.16 The **wm\_osDeleteAllFlashData** Function

The **wm\_osDeleteAllFlashData** function deletes all the data previously stored in flash memory by the Embedded Application.

Its prototype is :

```
s32 wm_osDeleteAllFlashData ( void );
```

The return value is the total deleted flash objects data size (0 if there was no objects to delete).

### 3.4.17 The **wm\_osDeleteRangeFlashData** Function

The **wm\_osDeleteRangeFlashData** function deletes all the flash objects between the provided start & end IDs.

Its prototype is :

```
s32 wm_osDeleteRangeFlashData ( u16 StartId, u16 EndId );
```

#### 3.4.17.1 Parameters

*StartId:*

Range to browse first Id

*EndId:*

Range to browse last Id

#### 3.4.17.2 Return Values

The return value is the total deleted flash objects data size (0 if there was no objects to delete), or ERROR if StartID is greater than EndId.

### 3.4.18 The **wm\_osGetHeapMemory** Function

The **wm\_osGetHeapMemory** function gets memory from the Embedded Application heap.

Its prototype is:

```
void *wm_osGetHeapMemory ( u16 MemorySize);
```

#### 3.4.18.1 Parameter

*MemorySize:*

Requested size.

#### 3.4.18.2 Return Values

The return parameter is the memory address, or is NULL if an error has occurred.

### 3.4.19 The **wm\_osReleaseHeapMemory** Function

The **wm\_osReleaseHeapMemory** function releases the previously reserved memory.

Its prototype is:

```
s32 wm_osReleaseHeapMemory (void * ptrData );
```

#### 3.4.19.1 Parameter

*PtrData:*

Points to the reserved memory.

#### 3.4.19.2 Return Values

The return parameter is positive or null if the reserved memory has been released, and negative if not.

### 3.4.20 The **wm\_osSuspend** function

The **wm\_osSuspend** suspend the execution of the Open AT<sup>®</sup> embedded application.

its prototype is :

```
void wm_osSuspend(void)
```

Note : The resume of the application is set with AT+WOPENRES or with the INTERRUPT feature when the PinInterrupt is set (see AT+WFM).

Open AT<sup>®</sup> application running in Remote Task Environment can not be suspended (the function has no effect).

### 3.4.21 The `wm_osGetTask` Function

The `wm_osGetTask` function returns the current task ID.

Its prototype is:

```
wm_osTask_e wm_osGetTask ( void );
```

#### 3.4.21.1 Return Values

The return parameter is the current embedded application task ID.

### 3.4.22 The `wm_osSendMsg` Function

The `wm_osSendMsg` function allows one embedded application task to send a user-defined message to the other application tasks.

Its prototype is:

```
s8 wm_osSendMsg (  wm_osTask_e  Task,  
                   u8           MsgID,  
                   u16          MsgLength,  
                   u8 *         MsgBody );
```

#### Notes:

- The sent message will be received by the destination task as a parameter of the `Parser()` function.
- The received message ID will be (`WM_USER_MSG_BASE` + the `msgID` parameter).
- The received message body will be accessed through the `UserMsg` member of the `wm_apmBody_t` union.

#### 3.4.22.1 Parameters

##### Task

Destination task ID.

##### MsgID

User-defined message ID ; allowed values are from 0 to 0x7F.

##### MsgLength

Message body length.

##### MsgBody

Message body data pointer.

#### 3.4.22.2 Return Values

The return parameter is the current embedded application task ID (Range of values is [0 ;(WM\_APM\_MAX\_TASK – 1)]).

### 3.4.23 Example: Managing Data Flash Objects

5KB of Data Flash objects are available for Embedded Applications. Data Flash objects are organized in Ids and managed by the Embedded Application.

An example related to Data Flash reading/writing is given below:

```
s32 LengthRead;
s32 Length;
u8* Ptr;
u16 Id;
s32 Writen;

FlashId = 112;

/* Get the len */
Length = wm_osGetLenFlashData (FlashId);
if ( Length > 0 )
{
    Ptr = wm_osGetHeapMemory (Length);

    /* Read the Flash Id item */
    LengthRead = wm_osReadFlashData (FlashId, Length, Ptr);

    Ptr[3] = 0x10; /* Change something */

    /* Write the modified Flash Id item */
    Writen = wm_osWriteFlashData (FlashId, Length, Ptr);
}
```

### 3.4.24 Example: RAM management

32 or 128 Kbytes (according to product type) of RAM are available for Embedded Applications and the provided Wavecom library manages this RAM.

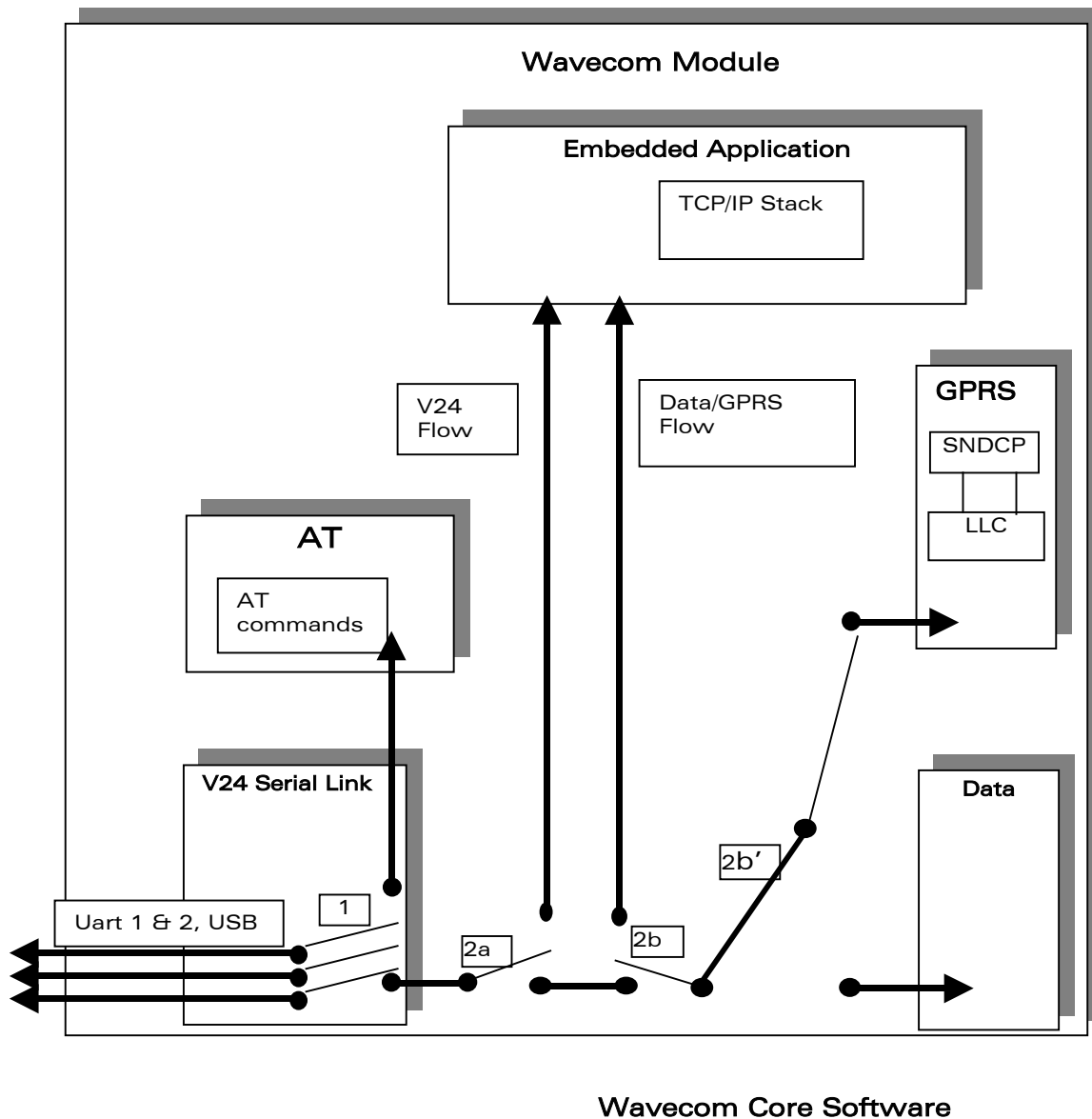
An example of the RAM request function is given below:

```
void *ptr;
ptr = wm_osGetHeapMemory (1000);/* 1000 bytes are requested */
```

An example of the RAM release function is given below:

```
wm_osReleaseHeapMemory (ptr);
```

### 3.5 Flow Control Manager API



**Figure 3: Flow Control Function**

The Flow Control Manager API provides IO flows to the Embedded Application:

- V24 serial link (UART 1, UART 2, USB)
- Data Communication (through the GSM or GPRS air interface).

By default, these flows are closed (in Figure 3, Switches 2a and 2b are closed to transmit all data directly between the V24 serial links and Data or GPRS communication). The Embedded Application can use the functions `wm_fcmOpen()` (see §3.5.3) and `wm_fcmClose()` (see §3.5.4) to open or close these flows.



29th November 2005

Only one serial flow and one data communication flow can be opened at a time (UART 1 & UART 2 flows can not be opened together, as GSM data and GPRS flow, but UART 2 and GPRS flows can be opened at the same time, for example)

The Switch 1 function is described in § 3.6.2.1: “The `wm_ioSerialSwitchState` Function”

### Important note

GPRS provides only **packet** mode transmission. This means that you can only send **IP packets to the GPRS flow**.

### **3.5.1 Required Header**

This API is defined in `wm_fcm.h` header file.

This file is included by `wm_apm.h`.

### **3.5.2 The `wm_fcmFlow_e` enum type**

```
typedef enum
{
    WM_FCM_DATA,           /* gsm network      */
    WM_FCM_GPRS,          /* gprs network     */
    WM_FCM_V24,           /* serial interface : default UART 1 */
    WM_FCM_V24_UART1 = WM_FCM_V24, // Uart 1 flow
    WM_FCM_V24_UART2,    // Uart 2 flow
    WM_FCM_USB,           // USB flow
    WM_FCM_LAST_FLOW
} wm_fcmFlow_e;
```

### 3.5.3 The `wm_fcmOpen` Function

The `wm_fcmOpen` function opens the requested flow between the Embedded Application and a serial link port, or a Data communication port.

Its prototype is :

```
s32  wm_fcmOpen (  wm_fcmFlow_e  FlowID,  
                  u16           DataMaxToReceive );
```

#### 3.5.3.1 Parameters

##### Flow

The flow to be opened, using the `wm_fcmFlow_e` type.

##### DataMaxToReceive

Maximum block size to be sent to the Embedded Application from the requested flow. This size can not exceed **120 bytes** for a serial link flow, **270 bytes** for the GSM data flow, and is not used for the GPRS flow.

#### 3.5.3.2 Return value

- WM\_FCM\_OK if successful.
- WM\_FCM\_ERR\_NO\_LINK if the requested flow can't be opened (the GSM and GPRS flows can't be opened together, as the serial link / USB flows).
- WM\_FCM\_ERR\_UNKNOWNFLOW if the used flow ID is unknown.

#### 3.5.3.3 Notes

- ❑ The flow opening response is received by the Embedded Application through a message. This message is available as a parameter of the `Parser()` function of the task which has called the `wm_fcmOpen` function, with the `MsgTyp` parameter set to `WM_FCM_OPEN_FLOW`.
- ❑ The `DataMaxToSend` parameter of the `WM_FCM_OPEN_FLOW` message informs the Embedded Application of the maximum data block size it can send on this flow. If this parameter is 0, there is no size limitation.
- ❑ The `wm_fcmOpen()` function on the GSM data flow **must** be called **before** using the "ATD" command to set up a data call.
- ❑ The `wm_fcmOpen()` function on the GPRS flow **must** be called **AFTER** using the `wm_gprsOpen()` function, followed by "ATD\*99" or +CGACT or +CGDATA commands to set up a GPRS session.
- ❑ After the end of a data call or GPRS session (on NO CARRIER unsolicited response, or on ATH command), the `wm_fcmClose()` function must be called before setting up a new data call / GPRS session.

### 3.5.4 The `wm_fcmClose` Function

The `wm_fcmClose` function closes the requested flow between the Embedded Application and a serial link port, or a Data communication port.

Its prototype is:

```
s32 wm_fcmClose (wm_fcmFlow_e FlowID );
```

#### 3.5.4.1 Parameters

##### Flow

The flow to be closed, using the `wm_fcmFlow_e` type (see §3.5.2 for `wm_fcmFlow_e` description).

#### 3.5.4.2 Return Value

- WM\_FCM\_OK if successful.
- WM\_FCM\_ERR\_NO\_LINK if the requested flow is not opened.
- WM\_FCM\_KO if the closing of data flow has failed.

#### 3.5.4.3 Notes

- The flow closing response is received by the Embedded Application through a message. This message is available as a parameter of the `Parser()` function of the task which has used the `wm_fcmOpen` function, with `MsgTyp` parameter set to `WM_FCM_CLOSE_FLOW`.
- The `wm_fcmClose` function **must** be called **after** any data call or GPRS session release.

### 3.5.5 The `wm_fcmSubmitData` Function

The `wm_fcmSubmitData` function submits a data block to the Flow Control Manager.

Its prototype is:

```
s32 wm_fcmSubmitData (    wm_fcmFlow_e      Flow,
                          wm_fcmSendBlock_t *  fcmDataBlock );
```

#### 3.5.5.1 Parameters

##### Flow

Specifies the IO flow where the data are sent; See §3.5.2 for `wm_fcmFlow_e` description.

##### *fcmDataBlock*:

Pointer on a `wm_fcmSendBlock_t` structure, allocated (see § 3.4.18: "The `wm_osGetHeapMemory` ") and filled by the Embedded Application before sending.

For example, to send 10 data bytes, the buufer must be allocated as follows :

```
fcmDataBlock = (wm_fcmSendBlock_t *) wm_osGetHeapMemory (
sizeof ( wm_fcmSendBlock_t ) + 10);
```

The definition of this structure is as follows:

```
typedef struct {
    u16 Reserved1[4];
    u16 DataLength;      /* number of byte of data to send */
    u16 Reserved2[5];
    u8 Data[1];        /* data buffer to send */
} wm_fcmSendBlock_t;
```

#### 3.5.5.2 Returned Values

Returned Value	Description
WM_FCM_OK	the data block is sent, the memory allocated for <code>fcmDataBlock</code> is released, and the Embedded Application may go on sending more data blocks.
WM_FCM_EOK_NO_CREDIT	the data block is sent and the memory allocated for <code>fcmDataBlock</code> is released, but the Embedded Application must wait for the <code>WM_FCM_RESUME_DATA_FLOW</code> message before sending more data blocks. This message is available as a parameter of the <code>wm_apmAppliParser()</code> function.
WM_FCM_ERR_NO_CREDIT	the data block is not sent and the memory allocated for <code>fcmDataBlock</code> is not released. The Embedded Application must wait for the <code>WM_FCM_RESUME_DATA_FLOW</code> message before sending more data blocks. This message is available as a parameter of the <code>wm_apmAppliParser()</code> function.
WM_FCM_ERR_NO_LINK	the flow is not opened. The data block is not sent and the memory allocated for <code>fcmDataBlock</code> is not released.

Returned Value	Description
WM_FCM_ERR_UNKNOW_N_FLOW	the Embedded Application used an incorrect flow ID. The data block is not sent and the memory allocated for fcmDataBlock is not released.
WM_FCM_ERR_NO_LINK	the requested flow is not opened or can't be opened (the GSM and GPRS flows can't be opened together).

### 3.5.5.3 Notes

- ❑ A successful data send by the **wm\_fcmSubmitData()** function (with WM\_FCM\_OK or WM\_FCM\_EOK\_NO\_CREDIT return code) will result in the reception of a WM\_OS\_RELEASE\_MEMORY message by the Embedded Application. This message is available as a parameter of the **wm\_apmAppliParser()** function with the *MsgTyp* parameter set to WM\_OS\_RELEASE\_MEMORY.
- ❑ You should not call the **wm\_fcmSubmitData()** function more than once in the same message treatment. The Embedded Application should set a timer between each data block sending on the IO flows.
- ❑ Set a timer between the last data block sending on an IO flow, and this flow closing operation. Also, a timer should be set between the last data block sending on the V24 flow, and a call to the **wm\_ioSwitchSerialState (WM\_IO\_SERIAL\_AT\_MODE)** function.
- ❑ In remote task mode, as the serial link is strongly used (AT commands and responses, traces and messages between the remote task and the target software), a data send operation on the V24 flow with high speed rate will not work. The Embedded Application should send data blocks on the V24 flow a very **low speed rate**, in remote task mode.

### 3.5.6 Receive Data Blocks

The Embedded Application may receive data blocks from an opened Data or V24 IO flow, through the WM\_FCM\_RECEIVE\_BLOCK message. This message is available as a parameter of the **wm\_apmAppliParser()** function.

#### 3.5.6.1 Message Parameters

This is the WM\_FCM\_RECEIVE\_BLOCK message structure:

```
typedef struct {
    u16      DataLength;    /* number of bytes received */
    u8       Reserved1[2];
    wm_fcmFlow_e FlowId;    /* IO flow ID */
    u8       Reserved2[7];
    u8       Data[1];      /* received data buffer */
} wm_fcmReceiveBlock_t;
```

**DataLength**

Number of data bytes received in **Data** parameter from this flow. This size will not exceed **DataMaxToReceive** parameter of the **wm\_fcmOpen()** function (see §3.5.3).

**FlowID**

Specifies the opened IO flow from where the data are received. See §3.5.2 for **wm\_fcmFlow\_e** description.

**Data**

Data block received from the IO flow. The memory allocated for **Data** parameter will be released at the end of the **Parser()** function (see § 3.2.6).

**3.5.6.2 Notes**

- When the Embedded Application has treated one or more data blocks, it should inform the Flow Control Manager to release credits, in order to receive more data, by using the **wm\_fcmCreditToRelease()** function (see § 3.5.7: "The **wm\_fcmCreditToRelease**").

**3.5.7 The **wm\_fcmCreditToRelease** Function**

The **wm\_fcmCreditToRelease** function informs the Flow Control Manager that the Embedded Application has treated some data blocks, and is ready to receive more data. This credit release system provides more security for the data transfer.

Its prototype is :

```
s32 wm_fcmCreditToRelease (   wm_fcmFlow_e   Flow,  
                             u8               Credits );
```

**3.5.7.1 Parameters***Flow:*

Specifies the IO flow on which the Flow Control Manager may release credits. See §3.5.2 for **wm\_fcmFlow\_e** description.

*Credits:*

Specifies the number of credits the Embedded Application wants the Flow Control Manager to release. This represents the number of data blocks received and treated by the Embedded Application. For example: when the Embedded Application has received and treated 3 data blocks (i.e. 3 **WM\_FCM\_RECEIVE\_BLOCK** messages), it should inform the Flow Control Manager by calling **wm\_fcmCreditToRelease()** function with the **Credits** parameter set to 3.

**3.5.7.2 Returned Values**

The returned value is positive or zero if the credits are successfully released.

**WM\_FCM\_ERR\_NO\_LINK** if the requested flow is not opened or can not be opened (the GSM and GPRS flows can not be opened together).

### 3.5.8 The `wm_fcmQuery` Function

The `wm_fcmQuery` function informs the Embedded Application of the FCM buffers status.

Its prototype is :

```
s32 wm_fcmQuery (  wm_fcmFlow_e  Flow,  
                  wm_fcmWay_e   Way );
```

#### 3.5.8.1 Parameters

*Flow:*

Specifies the IO flow from which the buffer status is requested. See §3.5.2 for `wm_fcmFlow_e` description.

*Way:*

As flows have two way (*from* Embedded application, and *to* Embedded application), this parameter specifies the way from which the buffer status is requested. The possible values are:

```
typedef enum      {  
    WM_FCM_WAY_FROM_EMBEDDED,  
    WM_FCM_WAY_TO_EMBEDDED  
} wm_fcmWay_e;
```

#### 3.5.8.2 Returned Values

The returned value is `WM_FCM_BUFFER_EMPTY`, the requested flow & way buffer is empty.

The returned value is `WM_FCM_BUFFER_NOT_EMPTY`, the requested flow & way buffer is not empty ; the Flow Control Manager is still processing data on this flow.

A negative returned value means that an error occurred.

## 3.6 Input Output API

This API manages Serial Link State and GPIO operations.

### 3.6.1 Required Header

This API is defined in `wm_io.h` header file.

This file is included by `wm_apm.h`.

### 3.6.2 Serial Link State functions

#### 3.6.2.1 The `wm_ioSerialSwitchState` Function

The `wm_ioSerialSwitchState` function sets the serial link mode:

- AT command computing, or
- direct data transmission through the V24 Serial Link Flow.

Its prototype is:

```
void wm_ioSerialSwitchState
    (wm_ioPort_e Port
     wm_ioSerialSwitchState_e SerialState);
```

##### 3.6.2.1.1 Parameters

###### Port

Specifies the IO port to switch the state, using following type :

```
typedef enum
{
    WM_IO_UART1,
    WM_IO_UART2,
    WM_IO_USB
} wm_ioPort_e;
```

###### SerialState

Specifies the requested state of the Serial Link. The possible values are defined below:

```
typedef enum {
    WM_IO_SERIAL_AT_MODE,
    WM_IO_SERIAL_DATA_MODE,
    WM_IO_SERIAL_ATO
} wm_ioSerialSwitchState_e;
```

`WM_IO_SERIAL_AT_MODE` represents the AT commands computing mode. In this mode, data received from V24 serial link is parsed and treated like AT commands.

`WM_IO_SERIAL_DATA_MODE` represents the direct data transmission mode. In this mode, data received from V24 serial link is transmitted without treatment through the V24 Serial Link Flow.

`WM_IO_SERIAL_ATO` is used only if the External Application sent a "+++" string, in order to switch the V24 interface in "ONLINE" mode (see next paragraph "Notes")



### 3.6.2.1.2 Notes

- The serial mode switching response is received by the Embedded Application through a message. This message is available as a parameter of the `Parser()` function with the **MsgTyp** parameter set to `WM_IO_SERIAL_SWITCH_STATE_RSP` (see § 3.2.6). The **SerialMode** parameter of this message is the requested Serial Link Mode; if the **RequestReturn** parameter is negative, an error occurred, and the Serial Link Mode does not change.
- The `wm_ioSerialSwitchState()` function is not allowed if the V24 Serial Link and the Data Flows are not opened by the Embedded Application (see § 3.5.3). In this case, the `WM_IO_SERIAL_SWITCH_STATE_RSP` message will always return a negative **RequestReturn** parameter.
- In Figure 2 (see § 3.5: "Flow Control Manager API"), the `wm_ioSerialSwitchState()` function controls Switch 1.

### VERY IMPORTANT NOTES

- Sending the "+++" sequence from an External Application while the serial link is in `WM_IO_SERIAL_DATA_MODE` state will switch it to `WM_IO_SERIAL_AT_MODE` state after the **OK** response, during or out of a data call. The "+++" sequence **must** be preceded and followed by a period of one second without character sending, in order to allow the serial link to switch to `WM_IO_SERIAL_AT_MODE` state.

### 3.6.2.2 The `wm_ioSerialGetSignal` Function

The `wm_ioSerialGetSignal` function allows to get the current values of the CTS and DSR signals of the serial line.

Its prototype is :

```
s32 wm_ioSerialGetSignal(wm_ioSerialGetSignal_e SerialSignal)
```

#### 3.6.2.2.1 Parameters

*SerialSignal:*

Value designating the signal to get, using following type :

```
typedef enum
{
    WM_IO_SERIAL_CTS,
    WM_IO_SERIAL_DSR
}wm_ioSerialGetSignal_e;
```

#### 3.6.2.2.2 Returned Values

- 1: The signal is on (active)
- 0 : The signal is off

### 3.6.3 GPIO types and functions

#### 3.6.3.1 Types

##### 3.6.3.1.1 The `wm_ioConfig_t` structure

This structure is used by the `wm_ioAllocate` function in order to set the reserved GPIO parameters.

```
typedef struct
{
    wm_ioLabel_u      eLabel;
    u32               Pad;
    wm_ioDirection_e eDirection;
    wm_ioState_e      eState;
} wm_ioConfig_t;
```

The `eLabel` member represents the GPIO label.  
The `eDirection` member represents the GPIO direction.  
The `eState` member represents the GPIO state.

##### 3.6.3.1.2 The `wm_ioLabel_u` union

This union represents the different GPIO labels, depending on the used product.

```
typedef union
{
    wm_ioLabel_Q24X0_e Q24X0_Label;
    wm_ioLabel_Q24X3_e Q24X3_Label;
    wm_ioLabel_Q24X6_e Q24X6_Label;
    wm_ioLabel_P32X3_e P32X3_Label;
    wm_ioLabel_P32X6_e P32X6_Label;
    wm_ioLabel_Q31X6_e Q31X6_Label;
    wm_ioLabel_P51X6_e P51X6_Label;
    wm_ioLabel_Q25X1_e Q25X1_Label;
} wm_ioLabel_u;
```

The `Q24X0_Label` member must be used on Wismo Quik Q24x0 products.  
The `Q24X3_Label` member must be used on Wismo Quik Q24x3 products.  
The `Q24X6_Label` member must be used on Wismo Quik Q24X6 products.  
The `P32X3_Label` member must be used on Wismo Pac P3xx3 based products.  
The `P32X6_Label` member must be used on Wismo Pac P32X6 based products.  
The `Q31X6_Label` member must be used on Wismo Quik P31X6 products.  
The `P51X6_Label` member must be used on Wismo Pac P5186 products.  
The `Q24X1_Label` member must be used on Wismo Quik Q25X1 products.

## WISMO QUIK Q24X0 GPIO LABELS

The Gpio labels for Wismo Quik Q24X0 products are defined by the values below :

```
typedef enum
{
    WM_IO_Q24X0_GPI      = 0x00000001, // GPI ID
    WM_IO_Q24X0_GPO_0   = 0x00000002, // GPO IDs
    WM_IO_Q24X0_GPO_1   = 0x00000004,
    WM_IO_Q24X0_GPO_2   = 0x00000008,
    WM_IO_Q24X0_GPO_3   = 0x00000010,
    WM_IO_Q24X0_GPIO_0  = 0x00000020, // GPIO IDs
    WM_IO_Q24X0_GPIO_4  = 0x00000200,
    WM_IO_Q24X0_GPIO_5  = 0x00000400
} wm_ioLabel_Q24X0_e;
```

## WISMO QUIK Q2XX3 GPIO LABELS

The Gpio labels for Wismo Quik Q2XX3 products are defined by the values below :

```
typedef enum
{
    WM_IO_Q24X3_GPI     = 0x00000001, // GPI ID
    WM_IO_Q24X3_GPO_1   = 0x00000004, // GPO IDs
    WM_IO_Q24X3_GPO_2   = 0x00000008,
    WM_IO_Q24X3_GPIO_0  = 0x00000010, // GPIO IDs
    WM_IO_Q24X3_GPIO_4  = 0x00000100,
    WM_IO_Q24X3_GPIO_5  = 0x00000200
} wm_ioLabel_Q24X3_e;
```

## WISMO QUIK Q24X6 GPIO LABELS

The Gpio labels for Wismo Quik Q2406 products are defined by the values below :

```
typedef enum
{
    WM_IO_Q24X6_GPI     = 0x00000001, // GPI ID
    WM_IO_Q24X6_GPO_0   = 0x00000002, // GPO IDs
    WM_IO_Q24X6_GPO_1   = 0x00000004,
    WM_IO_Q24X6_GPO_2   = 0x00000008,
    WM_IO_Q24X6_GPO_3   = 0x00000010,
    WM_IO_Q24X6_GPIO_0  = 0x00000020, // GPIO IDs
    WM_IO_Q24X6_GPIO_4  = 0x00000200,
    WM_IO_Q24X6_GPIO_5  = 0x00000400
} wm_ioLabel_Q24X6_e;
```

29th November 2005

## WISMO PAC P3XX3 GPIO LABELS

The Gpio labels for Wismo Pac P3XX3 products are defined by the values below :

```
typedef enum
{
    WM_IO_P32X3_GPI                = 0x00000001, // GPI ID
    WM_IO_P32X3_GPIO_0 = 0x00000008, // GPIO IDs
    WM_IO_P32X3_GPIO_2 = 0x00000020,
    WM_IO_P32X3_GPIO_3 = 0x00000040,
    WM_IO_P32X3_GPIO_4 = 0x00000080,
    WM_IO_P32X3_GPIO_5 = 0x00000100
} wm_ioLabel_P32X3_e;
```

## WISMO PAC P32X6 GPIO LABELS

The Gpio labels for Wismo Pac P32X6 products are defined by the values below :

```
typedef enum
{
    WM_IO_P32X6_GPI                = 0x00000001, // GPI ID
    WM_IO_P32X6_GPO_0 = 0x00000002, // GPO ID
    WM_IO_P32X6_GPIO_0 = 0x00000008, // GPIO IDs
    WM_IO_P32X6_GPIO_2 = 0x00000020,
    WM_IO_P32X6_GPIO_3 = 0x00000040,
    WM_IO_P32X6_GPIO_4 = 0x00000080,
    WM_IO_P32X6_GPIO_5 = 0x00000100,
    WM_IO_P32X6_GPIO_8 = 0x00000800
} wm_ioLabel_P32X6_e;
```

## WISMO QUIK Q31X6 GPIO LABELS

The Gpio labels for Wismo Quik Q31X6 products are defined by the values below :

```
typedef enum
{
    WM_IO_Q31X6_GPI                = 0x00000001, // GPI ID
    WM_IO_Q31X6_GPO_1 = 0x00000004, // GPO IDs
    WM_IO_Q31X6_GPO_2 = 0x00000008,
    WM_IO_Q31X6_GPIO_3 = 0x00000080, // GPIO IDs
    WM_IO_Q31X6_GPIO_4 = 0x00000100,
    WM_IO_Q31X6_GPIO_5 = 0x00000200,
    WM_IO_Q31X6_GPIO_6 = 0x00000400,
    WM_IO_Q31X6_GPIO_7 = 0x00000800
} wm_ioLabel_Q31X6_e;
```

#### WISMO PAC P5186 GPIO LABELS

The Gpio labels for Wismo Pac P5186 products are defined by the values below :

```
typedef enum
{
    WM_IO_P5186_GPO_0 = 0x00000001, // GPO ID
    WM_IO_P5186_GPO_1 = 0x00000002,
    WM_IO_P5186_GPIO_0 = 0x00000020, // GPIO IDs
    WM_IO_P5186_GPIO_4 = 0x00000200,
    WM_IO_P5186_GPIO_5 = 0x00000400,
    WM_IO_P5186_GPIO_8 = 0x00002000,
    WM_IO_P5186_GPIO_9 = 0x00004000,
    WM_IO_P5186_GPIO_10 = 0x00008000,
    WM_IO_P5186_GPIO_11 = 0x00010000,
    WM_IO_P5186_GPIO_12 = 0x00020000
} wm_ioLabel_P5186_e;
```

#### WISMO QUIK Q25X1 GPIO LABELS

The Gpio labels for Wismo Quik Q25X1 products are defined by the values below :

```
typedef enum
{
    WM_IO_Q25X1_GPI = 0x00000001,
    WM_IO_Q25X1_GPO_0 = 0x00000002,
    WM_IO_Q25X1_GPO_1 = 0x00000004,
    WM_IO_Q25X1_GPO_2 = 0x00000008,
    WM_IO_Q25X1_GPO_3 = 0x00000010,
    WM_IO_Q25X1_GPIO_0 = 0x00000020,
    WM_IO_Q25X1_GPIO_1 = 0x00000040,
    WM_IO_Q25X1_GPIO_2 = 0x00000080,
    WM_IO_Q25X1_GPIO_3 = 0x00000100,
    WM_IO_Q25X1_GPIO_4 = 0x00000200,
    WM_IO_Q25X1_GPIO_5 = 0x00000400,
    WM_IO_Q25X1_PAD = 0x7FFFFFFF
} wm_ioLabel_Q25X1_e;
```

#### 3.6.3.1.3 The `wm_ioDirection_e` type

This type represents the direction used for a GPIO.

```
typedef enum
{
    WM_IO_OUTPUT,
    WM_IO_INPUT,
    WM_IO_NORMAL
} wm_ioDirection_e;
```

The `WM_IO_OUTPUT` constant is used to set a GPIO as an output.  
The `WM_IO_INPUT` constant is used to set a GPIO as an input.

**A GPI must always be allocated with the `WM_IO_INPUT` direction.  
A GPO must always be allocated with the `WM_IO_NORMAL` direction.**

### 3.6.3.1.4 The `wm_ioState_e` type

This type represents the state of a GPIO.

```
typedef enum
{
    WM_IO_LOW,
    WM_IO_HIGH
} wm_ioState_e;
```

The `WM_IO_LOW` constant represents the low state of a GPIO.

The `WM_IO_HIGH` constant represents the high state of a GPIO.

### 3.6.3.1.5 The `wm_ioSetDirection_t` structure

This type is used by the `wm_ioSetDirection` function to set a GPIO to a new direction.

```
typedef struct
{
    wm_ioLabel_u    eLabel;
    wm_ioDirection_e eDirection;
} wm_ioSetDirection_t;
```

The `eLabel` member represents the GPIO label.

The `eDirection` member represents the new GPIO direction.

### 3.6.3.1.6 Return values definition

Return value	Definition
<code>WM_IO_PROC_DONE</code>	the function processing is done successfully.
<code>WM_IO_UNKNOWN_TYPE</code>	a direction parameter has an incorrect value.
<code>WM_IO_INPUT_CANT_BE_SET</code>	the function failed to set an Input pin.
<code>WM_IO_OUTPUT_CANT_BE_READ</code>	the function failed to read an Output pin.
<code>WM_IO_NO_MORE_HANDLES_LEFT</code>	no more free handle to allocate the requested GPIOs.
<code>WM_IO_EXCEED_MAX_NUMBER</code>	a parameter exceeded the allowed range value.
<code>WM_IO_UNALLOCATED_HANDLE</code>	a handle parameter has an incorrect value.
<code>WM_IO_INCOHERENCE_BETWEEN_HANDLE_AND_MASK</code>	the function tried to use a GPIO mask with an incorrect handle.
<code>WM_IO_INCOHERENCE_BETWEEN_DIRECTION_AND_MASK</code>	the function tried to set an input pin direction to output, or an output pin direction to input.
<code>WM_IO_IO_ALREADY_USED</code>	the function tried to allocate a GPIO already allocated on another handle.
<code>WM_IO_INCOHERENCE_BETWEEN_HANDLE_AND_IO_NUMBER</code>	the function tried to use a GPIO value with an incorrect handle.

### 3.6.3.2 The `wm_ioAllocate` Function

The `wm_ioAllocate` function reserves one or more GPIO(s) for the Embedded Application use.

Its prototype is :

```
s32 wm_ioAllocate ( u32 NbGpioToAllocate,  
                  wm_ioConfig_t * GpioCustomerConfig );
```

#### 3.6.3.2.1 Parameters

*NbGpioToAllocate* :

Size of the `GpioCustomerConfig` array.

*GpioCustomerConfig* :

Array of values, defined by the `wm_ioConfig_t` structure (see §3.6.2.1.1).

For each member of this array :

- `eLabel` represents the label of the requested GPIO, GPI or GPO, depending on the product used.
- `eDirection` represents the direction used for this GPIO.
- `eState` represents the state of the requested GPIO.

#### 3.6.3.2.2 Returned Values

If the GPIO allocation operation is successful, the returned value is a positive or null handle, which must be used in all further operations on the reserved GPIO. Otherwise, a negative returned value represents an error (cf §3.6.3.1.6).

#### 3.6.3.2.3 Notes

- ❑ The `eDirection` member of the `wm_ioConfig_t` structure is only significant for GPIO pins. GPI pins should be always set as an input and GPO pins should be always set as an output. Otherwise, the `eDirection` parameter is not taken into account.
- ❑ The `eState` member of the `wm_ioConfig_t` structure is only significant for pins set as an output by the `eDirection` parameter. Otherwise, the `eState` parameter is not taken into account.
- ❑ After a successful allocation, GPIO allocated by the Embedded Application are no more available for AT commands (AT+WIOR, AT+WIOW, AT+WIOM).

### 3.6.3.3 The `wm_ioRelease` Function

The `wm_ioRelease` function allows to release one or more GPIO reserved by the `wm_ioAllocate` function.

Its prototype is :

```
s32 wm_ioRelease ( s32 Handle,  
                  u32 NbGpioToRelease,  
                  wm_ioLabel_u * GpioCustomerLabel );
```

#### 3.6.3.3.1 Parameters

*Handle* :

Handle returned by the `wm_ioAllocate` function. All GPIOs of `GpioCustomerLabel` parameter must be related to this Handle.

*NbGpioToRelease* :

Size of the `GpioCustomerLabel` array.

*GpioCustomerLabel* :

Array of values, defined by the `wm_ioLabel_u` union (see §3.6.3.1.2).

Each member of this array represents the label of one GPIO to release.

#### 3.6.3.3.2 Returned Values

OK: successful completion

Otherwise, a negative returned value represents an error (cf §3.6.3.1.6).

#### Notes

- ❑ If one of the given GPIO labels is not related to the given Handle, the `wm_ioRelease` function will fail.
- ❑ After a successful release, GPIO released control is resumed by AT commands (AT+WIOR, AT+WIOW, AT+WIOM).

### 3.6.3.4 The `wm_ioSetDirection` Function

The `wm_ioSetDirection` function allows to change the direction of an allocated GPIO.

Its prototype is :

```
s32 wm_ioSetDirection ( s32 Handle,  
                        u32 NbGpioToChangeDir,  
                        wm_ioSetDirection_t * GpioDirection );
```

#### 3.6.3.4.1 Parameters

*Handle* :

Handle returned by the `wm_ioAllocate` function. All GPIOs of `GpioDirection` parameter must be related to this Handle.

*NbGpioToChangeDir* :

Size of the `GpioDirection` array.

*GpioDirection* :

Array of values, defined by the `wm_ioSetDirection_t` structure (see § 3.6.3.1.5).



For each member of this array :

- eLabel represents the label of the GPIO, GPI or GPO to change direction, depending on the used product.
- eDirection represents the new direction to use for this GPIO.

#### 3.6.3.4.2 Returned Values

OK: successful completion

Otherwise, a negative returned value represents an error (cf §3.6.3.1.6).

#### Notes

- If one of the given GPIO labels is not related to the given Handle, the `wm_ioSetDirection` function will fail.
- This function is only useful for GPIO pins. GPI or GPO pins direction should not be changed.

#### 3.6.3.5 The `wm_ioRead` Function

The `wm_ioRead` function allows to read the current state of one or more allocated GPIO(s).

Its prototype is :

```
s32 wm_ioRead (    s32 Handle,
                  u32 Gpio,
                  u32 * GpioState );
```

##### 3.6.3.5.1 Parameters

*Handle* :

Handle returned by the `wm_ioAllocate` function. All GPIOs of "Gpio" parameter must be related to this Handle.

*Gpio* :

Mask designating the GPIO(s) to read. This mask is obtained by performing a logical OR with members of the `wm_ioLabel_u` union.

*GpioState* :

Mask used to return the read states. Each bit of this mask represents the state of the corresponding GPIO in the "Gpio" parameter.

##### 3.6.3.5.2 Returned Values

OK: successful completion

Otherwise, a negative returned value represents an error (cf §3.6.3.1.6).

#### Notes

- If one of the given GPIO labels is not related to the given Handle, the `wm_ioRead` function will fail.

#### 3.6.3.6 The `wm_ioSingleRead` Function

The `wm_ioSingleRead` function allows to read the current state of one single allocated GPIO.

Its prototype is :

```
s32 wm_ioSingleRead (    s32 Handle,
                        u32 Gpio );
```

### 3.6.3.6.1 Parameters

*Handle :*

Handle returned by the `wm_ioAllocate` function. The "Gpio" parameter must be related to this Handle.

*Gpio :*

Value designating the GPIO to read, member of the `wm_ioLabel_u` union.

### 3.6.3.6.2 Returned Values

If the read operation is successful, the function returns the GPIO state, as defined in `wm_ioState_e` type.

Otherwise, a negative returned value represents an error (cf § 3.6.3.1.6: "Return values definition").

### Notes

- If the given GPIO label is not related to the given Handle, the `wm_ioSingleRead` function will fail.

### 3.6.3.7 The `wm_ioWrite` Function

The `wm_ioWrite` function allows to define a new state for one or more allocated GPIO(s).

Its prototype is :

```
s32 wm_ioWrite ( s32 Handle,  
                u32 Gpio,  
                u32 GpioState );
```

#### 3.6.3.7.1 Parameters

*Handle :*

Handle returned by the `wm_ioAllocate` function. All GPIOs of "Gpio" parameter must be related to this Handle.

*Gpio :*

Mask designating the GPIO(s) to write. This mask is obtained by performing a logical OR with members of the `wm_ioLabel_u` union.

*GpioState :*

Mask used to indicate the different states to write. Each bit of this mask represents the state of the corresponding GPIO in the "Gpio" parameter.

#### 3.6.3.7.2 Returned Values

OK: successful completion

Otherwise, a negative returned value represents an error (cf § 3.6.3.1.6: "Return values definition").

### Notes

- If one of the given GPIO labels is not related to the given Handle, the `wm_ioWrite` function will fail.

### 3.6.3.8 The `wm_ioSingleWrite` Function

The `wm_ioSingleWrite` function allows to define a new state for one single allocated GPIO.

Its prototype is :

```
s32 wm_ioSingleWrite ( s32 Handle,  
                      u32 Gpio  
                      u32 State );
```

#### 3.6.3.8.1 Parameters

*Handle* :

Handle returned by the `wm_ioAllocate` function. The "Gpio" parameter must be related to this Handle.

*Gpio* :

Value designating the GPIO to write, member of the `wm_ioLabel_u` union.

*State* :

Value designating the State to write (as defined by the `wm_ioState_e` type).

#### 3.6.3.8.2 Returned Values

OK: successful completion

Otherwise, a negative returned value represents an error (cf § 3.6.3.1.6: "Return values definition").

#### Notes

- If the given GPIO label is not related to the given Handle, the `wm_ioSingleWrite` function will fail.

## 3.7 GPRS API

A set of AT commands to manage the GPRS is provided.  
These commands are described in the AT Command Interface Guide.

### 3.7.1 GPRS Overview

#### 3.7.1.1 Introduction

The General Packet Radio Service (GPRS) is a set of GSM services that provides **packet** mode transmission within the Public Land Mobile Network (PLMN) and **interworks** with **external networks**. GPRS allows the subscriber to send and receive data in an end-to-end packet transfer mode, without using network resources in circuit-switched mode. GPRS enables the cost-effective and efficient use of network resources for **packet data applications** as :

- application with intermittent, non periodic data transmission
- frequent transmissions of small volumes of data
- infrequent transmissions of larger volumes of data

Based on standardized network protocols supported by the GPRS bearer services, a GPRS network operator may offer a set of additional services including :

- Retrieval services that provide the capability of accessing information stored in database centers. The information is sent to the user on demand only. Web is a good example of such services.
- Messaging services which offer communication between individual users via storage units with store and forward mailbox as e-mail client.
- Conversational services which provide bi-directional communication by means of real time end-to-end information transfer such as telnet application (download of melodies, games and more).
- Tele-action services which are characterized by low data volume transactions, such as credit card validation, bank account transaction, stock trading, electronic monitoring, utility meter reading and surveillance system.

GPRS permit to optimize the cost (the user is billed for the volume of data transferred and not for the connection duration) and a best interworking with external packet network.

Wavecom Mobile Equipment is GPRS class B compliant.

### 3.7.1.2 Definition of a PDP context

Before transferring any data packet between the mobile and the network, a PDP context (Packet Data Protocol) must be defined and activated by the mobile. These activation and deactivation procedures over the GPRS network are considered as signaling phases.

A PDP context is a structure which identifies a PDP (IP or X25 type, but Wavecom uses only IP context) which is like a virtual channel between the mobile and the GGSN (the GPRS Gateway which provide access to an external network). We communally call "GPRS session" an activated PDP.

Note that a PDP context is a logical channel which does not cost anything on idle (unlike GSM data call). It allows permanent data connection.

A PDP context is associated with a specific Quality Of Service.

A set of AT commands are available in order to activate, accept, deactivate and abort PDP contexts.

The PDP context activation may be initiated by the mobile or may be requested by the Network.

The mobile user can define more than one PDP contexts (up to 4 simultaneous) but can activate only one at a time.

The parameters which define a PDP context are :

- Cid is the identifier of the define PDP context (ie 1 to 4)
- PDP Type organization : IETF (IP type)
- PDP Address Information : Mobile address (static or dynamic) that identifies the ME in the address space applicable to the PDP
- QOS Profile requested : QOS requested by the user (mobile equipment)
- QOS Profile Minimum : QOS minimum accepted by the ME
- DCOMP : Data compression or not
- HCOMP : header compression or not
- Access Point Name : Access Point Name of the External Network which is a logical name that is used to select the GGSN or the external packet data network (ex web.sfr.fr). Provided by the GPRS operator.

Please refer to the definitions of GPRS AT commands for more informations.

#### IMPORTANT NOTE :

The `wm_fcmOpenGPRSAndV24()` function **must** be called **AFTER** using the `wm_gprsOpen()` function followed by "ATD\*99" or +CGACT or +CGDATA commands to set up a GPRS session.

### 3.7.2 The `wm_gprsAuthentication` function

This command sets the authentication parameters login/password to use with a particular Cid during a PDP activation.

Its prototype is :

```
s32 wm_gprsAuthentication(u8 Cid, ascii *login, ascii *password)
```

#### 3.7.2.1 Parameters

*Cid:*

(PDP Context Identifier) a numeric parameter (1-4) which specifies a particular PDP context definition (see AT Commands Interface Guide).

*Login and Password:*

The login and the password authentication parameters in ASCII character (terminated by a 0x00 character) used to authenticate the user during a PDP activation. **The maximum length of each authentication string is limited to 50 characters** (including the terminal 0x00 character). The string is truncated if its length is more than 25 characters.

#### Note

- These parameters **must be set before each PDP activation**.
- They **are optional** and depend of your subscription setup.

#### 3.7.2.2 Required Header

```
Wm_gprs.h
```

#### 3.7.2.3 Return value

0 if successful

WM\_GPRS\_CID\_NOT\_DEFINED If the Cid is not defined

WM\_NO\_GPRS\_SERVICE if the GPRS service is not supported

### 3.7.3 The `wm_gprsIPCPInformations` function

This command gets the current IPCP information to use with a particular Cid after a PDP activation.

These parameters are not saved in memory, and are only available during the life of the PDP context.

Its prototype is :

```
s32 wm_gprsIPCPInformations (  
    u8 Cid,  
    u32* DNS1,  
    u32* DNS2,  
    u32* Gateway)
```

#### 3.7.3.1 Parameters

*Cid:*

(PDP Context Identifier) a numeric parameter (1-4) which specifies a particular PDP context definition (see AT Commands Interface Guide).

*DNS1 and DNS2 and Gateway:*

Return values in native u32 format which are IPV4 addresses provided by the network. If the network doesn't provide them, the values are equal to 0.

#### **Note**

These parameters are optional and depend of the operator setup.

#### 3.7.3.2 Required Header

```
Wm_gprs.h
```

#### 3.7.3.3 Return value

0 if successful

WM\_GPRS\_CID\_NOT\_DEFINED If the Cid is not defined

WM\_NO\_GPRS\_SERVICE if the GPRS service is not supported

### 3.7.4 The **wm\_gprsOpen** function

This command sets Open AT<sup>®</sup> as the user of the GPRS bearer associated with the parameter Cid.

Its prototype is :

```
s32 wm_gprsOpen(u8 Cid)
```

#### **Note**

This interface **must be used before each PDP activation and before opening the FCM flows.**

#### 3.7.4.1 Parameters

*Cid:*

(PDP Context Identifier) a numeric parameter (1-4) which specifies a particular PDP context definition (see AT Commands Interface Guide).

#### 3.7.4.2 Required Header

```
Wm_gprs.h
```

#### 3.7.4.3 Return value

0 if successful

WM\_GPRS\_CID\_NOT\_DEFINED If the Cid is not defined

WM\_NO\_GPRS\_SERVICE if the GPRS service is not supported

### 3.7.5 The **wm\_gprsClose** function

This command unsets Open AT<sup>®</sup> as the user of the GPRS bearer associated with the parameter Cid.

Its prototype is :

```
s32 wm_gprsClose(u8 Cid)
```

#### **Note**

This interface **must be used after closing the PDP context and closing the FCM flows.**

#### 3.7.5.1 Parameters

*Cid:*

(PDP Context Identifier) a numeric parameter (1-4) which specifies a particular PDP context definition (see AT Commands Interface Guide).

#### 3.7.5.2 Required Header

```
Wm_gprs.h
```

#### 3.7.5.3 Return value

0 if successful

WM\_GPRS\_CID\_NOT\_DEFINED If the cid is not defined

WM\_NO\_GPRS\_SERVICE if the GPRS is not supported



### 3.8 BUS API

This API manages the I2C Soft, SPI and parallel bus operations.

Note: for bus management operations, the Q25x1 series module behaves as Q2406 modules.

#### 3.8.1 Required Header

This API is defined in `wm_bus.h` header file.

This file is included by `wm_apm.h`.

#### 3.8.2 Returned values definition

Returned Value	Description
WM_BUS_PROC_DONE	the function processing is successfully done.
WM_BUS_MODE_UNKNOWN_TYPE	unknown open mode type.
WM_BUS_UNKNOWN_TYPE	unknown bus type.
WM_BUS_BAD_PARAMETER	a parameter has an illegal value.
WM_BUS_SPI1_ALREADY_USED	the SPI bus is already open.
WM_BUS_I2C_SOFT_ALREADY_USED	the I2C Soft bus is already open.
WM_BUS_PARALLEL_ALREADY_USED	the parallel bus is already open.
WM_BUS_UNKNOWN_HANDLE	the handle used has an incorrect value.
WM_BUS_HANDLE_NOT_OPENED	no existing handle for this bus.
WM_BUS_NO_MORE_HANDLE_FRE	no more available handle for this bus.
WM_BUS_NOT_CONNECTED_ON_I2C	no peripheral connected on I2C Soft bus.
WM_BUS_NOT_ALLOWED_ADDRESS	unknown address.
WM_BUS_I2C_SOFT_GPIO_NOT_GPIO	the function tried to Open I2C Soft bus with a GPI or a GPO (not an adequate GPIO).
WM_BUS_SPI_LCDEN_NOT_FREE	the SPI bus has already been opened with the SPI_EN pin selected.
WM_BUS_SPI_AUX_NOT_FREE	the SPI bus has already been opened with the SPI_AUX pin selected.
WM_BUS_SPI_GPIO_CS_NOT_GPIO	the considered chip select pin is not a GPIO or a GPO.
WM_BUS_SPI_GPIO_CS_NOT_FREE	the considered chip select GPIO is not free.

### 3.8.3 The `wm_busOpen` Function

The `wm_busOpen` function allows to allocate a Handle on the required bus, and to open it for further read/write operations.

Its prototype is :

```
s32 wm_busOpen (    u32 BusType,  
                  u32 Mode  
                  wm_busSettings_u * Settings );
```

#### 3.8.3.1 Parameters

*BusType:*

Type of the bus to open. Defined values are :

- WM\_BUS\_SPI1 for SPI 1 bus (*not available on P5186 products*);
- WM\_BUS\_SPI3 for SPI 3 bus (*only available on P5186 products*);
- WM\_BUS\_SOFT\_I2C for I2C software bus.
- WM\_BUS\_PARALLEL for parallel bus (all WISMO products except Q2xxx products).

*Mode :*

Bus mode : the only defined value is WM\_BUS\_MODE\_STANDARD.

*Settings :*

Pointer on settings union, defined as below.

```
typedef union  
{  
    wm_busSPI1Settings_t      Spi1;  
    wm_busI2CSoftSettings_t  I2C_Soft;  
    wm_busParallelSettings_t Parallel;  
} wm_busSettings_u;
```

##### 3.8.3.1.1 SPI bus settings

To open the SPI bus you must use the SPI member of this union, defined as below:

```
typedef struct  
{  
    u32      Clk_Speed;  
    u32      Clk_Mode;  
    u32      ChipSelect;  
    u32      ChipSelectPolarity;  
    u32      LsbFirst;  
    u32      GpioChipSelect;  
    u32      WriteByteHandling;  
} wm_busSPI1Settings_t;
```

29th November 2005

- The "*Clk\_Speed*" parameter is the SPI clock speed. Defined values are defined in the table below:

Speed constant	Allowed on Q2XX3 and P3XX3 products	Allowed on QXX6 and P3XX6 products	Allowed on P5186 products
WM_BUS_SPI_SCL_SPEED_13Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_6_5Mhz		Yes	Yes
WM_BUS_SPI_SCL_SPEED_4_33Mhz		Yes	Yes
WM_BUS_SPI_SCL_SPEED_3_25Mhz	Yes	Yes	Yes
WM_BUS_SPI_SCL_SPEED_2_6Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_2_167Mhz		Yes	Yes
WM_BUS_SPI_SCL_SPEED_1_857Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_1_625Mhz	Yes	Yes	
WM_BUS_SPI_SCL_SPEED_1_44Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_1_3Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_1_181Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_1_083Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_1Mhz		Yes	
WM_BUS_SPI_SCL_SPEED_926Khz		Yes	
WM_BUS_SPI_SCL_SPEED_867Khz		Yes	
WM_BUS_SPI_SCL_SPEED_812Khz	Yes	Yes	
WM_BUS_SPI_SCL_SPEED_101Khz	Yes		

- The "*Clk\_Mode*" parameter is the SPI clock mode ; defined values are:
  - WM\_BUS\_SPI\_SCK\_MODE\_0 rest state 0, data valid on rising edge
  - WM\_BUS\_SPI\_SCK\_MODE\_1 rest state 0, data valid on falling edge
  - WM\_BUS\_SPI\_SCK\_MODE\_2 rest state 1, data valid on rising edge
  - WM\_BUS\_SPI\_SCK\_MODE\_3 rest state 1, data valid on falling edge

- The "*ChipSelect*" parameter selects the valid pin; defined values are:

WM_BUS_SPI_ADDRESS_SPI_EN	SPI_EN is the selected pin <i>only for Q2XX3 and P3XX3 products ; for Q24X6 and Q25X1 products, the GPO 3 pin must be used ; for P32X6 product, the GPIO 8 pin must be used ; not available on Q31X6 and P5186 products)</i>
WM_BUS_SPI_ADDRESS_SPI_EN_Q31	SPI_EN is the selected pin, <i>only available for Q31X6 products)</i>
WM_BUS_SPI_ADDRESS_SPI_AUX	SPI_AUX is the selected pin <i>only for Q2XX3 and P3XX3 products ; for Q24X6 and P32X6 products, the GPO 0 pin must be used; not available on Q31X6 and P5186 products)</i>
WM_BUS_SPI_ADDRESS_CS_GPIO	Use a GPIO as ChipSelect, the <i>GpioChipSelect</i> and <i>WriteByteHandling</i> parameters must be used) ;
WM_BUS_SPI_ADDRESS_CS_NONE	This Chip Select signal is not handled by the Open AT® BUS API. The application should allocate a GPIO to handle itself the Chip Select signal.

- The "*ChipSelectPolarity*" parameter sets the polarity of the ChipSelect; defined values are :

- WM\_BUS\_SPI\_CS\_POL\_LOW (active low) ;
- WM\_BUS\_SPI\_CS\_POL\_HIGH (active high) ;

- The "*LsbFirst*" parameter sets whether the data sent/received through the SPI bus is LSB or MSB ; this parameter applies only to the data, the opcode and address sent are always MSB first ; defined values are :

- WM\_BUS\_SPI\_LSB\_FIRST ;
- WM\_BUS\_SPI\_MSB\_FIRST

- The "*GpioChipSelect*" parameter is used only if the "*ChipSelect*" parameter is set to the WM\_BUS\_SPI\_ADDRESS\_CS\_GPIO value ; it is the GPIO label to use as a chip select. It must be a member of the `wm_ioLabel_u` union (see §3.6.3.1.2).

- The "*WriteByteHandling*" parameter is used only if the "*ChipSelect*" parameter is set to the WM\_BUS\_SPI\_ADDRESS\_CS\_GPIO value ; defined values are :

- WM\_BUS\_SPI\_BYTE\_HANDLING (GPIO signal state change on each written or read byte) ;
- WM\_BUS\_SPI\_FRAME\_HANDLING (GPIO signal works as other chip select pins).

### 3.8.3.1.2 I2CSoft bus

To open the I2C Soft bus you must use the "*I2C\_Soft*" parameter of the union, defined as below :

```
typedef struct
{
    u32      Scl_Gpio;
    u32      Sda_Gpio;
} wm_busI2CSoftSettings_t;
```

The *Scl\_Gpio* parameter is the label of the GPIO used to handle the SCL signal. The *Sda\_Gpio* parameter is the label of the GPIO used to handle the SDA signal.

Each of these labels must be a member of the *wm\_ioLabel\_u* union (see §3.6.3.1.2).

### 3.8.3.1.3 Parallel bus

To open the parallel bus you must use the "*Parallel*" parameter of the union, defined as below :

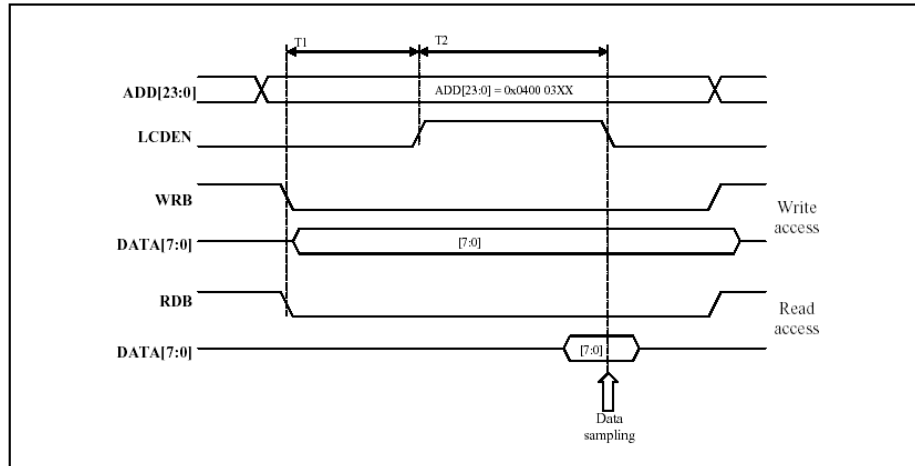
```
typedef struct
{
    u32      ChipSelect;
    u32      Lcd_AddressSetUpTime;
    u32      Lcd_LcdenSignalPulseDuration;
    u32      Lcd_PolarityControl;
    u32      Csusr_NbWaitState;
    u32      ReverseOrDirectOrder;
} wm_busParaSettings_t;
```

- The "*ChipSelect*" parameter selects the valid pin; defined values are:
  - WM\_BUS\_PARA\_CSUSER\_AS\_CS (Gpio 5 is the selected pin);
  - WM\_BUS\_PARA\_LCDEN\_AS\_CS (LCD\_EN is the selected pin);

- The "Lcd\_AddressSetUpTime" parameter sets the time between the setting of an address for the parallel bus and the activation of the LCD\_EN pin (only if LCD\_EN is the Chip Select). It is the T1 time on the figure 3 below. The allowed values are from 0 to 31. The resulting time is :

For P32X3 product : ( X \* 38.5 ) ns ;

For P32X6 product : ( 1 + 2 X ) \* 19 ns.



**Figure 4: Parallel bus chronogram**

- The "Lcd\_LcdenSignalPulseDuration" parameter sets the time during which the LCD\_EN pin is valid (only if LCD\_EN is the Chip Select). It is the T2 time on the figure 3 above. The allowed values are from 0 to 31. The resulting time is :

For P32X3 product : ( X + 1.5 ) \* 38.5 ns ;

For P32X6 product : ( 1 + 2 \* ( X + 1 ) ) \* 19 ns.

*(Important Warning, for this product, the 0 value is considered as 32).*

- The "Lcd\_PolarityControl" parameter sets the polarity of the ChipSelect. If LCD\_EN is the ChipSelect; the defined values are:
  - WM\_BUS\_PARA\_LCDEN\_POLARITY\_LOW  
*data is sampled on the rising edge from low state to high state of LCD\_EN.*
  - WM\_BUS\_PARA\_LCDEN\_POLARITY\_HIGH  
*data is sampled on the falling edge from high state to low state of LCD\_EN.*

If the GPIO 5 is the ChipSelect, the defined value is:

- WM\_BUS\_PARA\_LCDEN\_NOT\_USED ;

- The "CsUser\_NbWaitState" parameter sets the time during which the data is valid on the bus (only if the GPIO 5 is the Chip Select) ; defined values are :

- WM\_BUS\_PARA\_CSUSR\_0\_WAIT\_STATE (time of 62 ns) ;
- WM\_BUS\_PARA\_CSUSR\_1\_WAIT\_STATE (time of 100 ns)
- WM\_BUS\_PARA\_CSUSR\_2\_WAIT\_STATE (time of 138 ns)
- WM\_BUS\_PARA\_CSUSR\_3\_WAIT\_STATE (time of 176 ns)

- The “*ReverseOrDirectOrder*” parameter sets whether the data are sent as written in the buffer, or reversed before being sent ; defined values are:
  - WM\_BUS\_PARA\_DATA\_DIRECT\_ORDER ;
  - WM\_BUS\_PARA\_DATA\_REVERSE\_ORDER ;

### 3.8.3.2 Returned Values

On successful completion, the function returns a positive or null Handle, to use for further Read / Write / Close operations on this bus. Otherwise, the function will return a negative error value (cf § 3.8.2).

### 3.8.3.3 Notes

- For I2C Soft bus, the two GPIOs labels provided in the “Settings” parameter must **not** be allocated by the Embedded Application for another purpose ; only GPIOs are allowed, using GPI or GPO to open the I2C bus will result as an error.
- For SPI bus, if the ChipSelect is a GPIO, it must **not** be allocated by the Embedded Application for another purpose ; only GPIO and GPO are allowed, using GPI to open the SPI bus will result as an error.
- For Parallel bus, if the Chip Select is the GPIO 5, it must **not** be allocated by the Embedded Application for other purpose. On P32X6 product, the LCD\_EN chip select is available only if the GPIO 8 is not allocated by any application.
- A bus is available only if it was not opened before by AT commands with the same parameters (AT+WBM), otherwise, the wm\_busOpen will result as an error. If a bus is opened by the Embedded Application, it won't be available to AT commands, until the use of wm\_busClose function.

### 3.8.4 The wm\_busClose Function

The wm\_busClose function allows to close a bus previously allocated by the wm\_busOpen function.

Its prototype is :

```
s32 wm_busClose ( s32 Handle );
```

#### 3.8.4.1 Parameters

*Handle* :

Handle of the bus to close, returned by wm\_busOpen function.

#### 3.8.4.2 Returned Values

On successful completion, the function returns 0.

Otherwise, the function will return a negative error value (cf § 3.8.2: “Returned values definition”).

**Note:** If the bus was opened with some GPIOs as settings, the GPIOs labels passed in the “Settings” parameter of the wm\_busOpen function are available again after the return of the wm\_busClose function.

### 3.8.5 The `wm_busWrite` Function

The `wm_busWrite` function allows to write on a bus previously allocated by the `wm_busOpen` function.

Its prototype is :

```
s32 wm_busWrite (    s32 Handle
                    wm_busAccess_t * pAccessMode,
                    void * pDataToWrite,
                    u32 NbBytes );
```

#### 3.8.5.1 Parameters

*Handle* :

Handle of the bus device to write on, returned by `wm_busOpen` function.

*pAccessMode* :

Mode to use to access the device.

This parameter is defined using the following type :

```
typedef struct
{
    u32 Address;
    u32 Opcode;
    u8 OpcodeLength;
    u8 AddressLength;
} wm_busAccess_t;
```

This parameter is processed differently according the bus type :

- **For SPI bus :**

For Q24X3 and P32X3 products :

- one byte can be sent through the "Opcode" parameter (only the LSByte is used ; if "OpcodeLength" is less than 8 bits, only the MSBits of the LSByte are used),
- two bytes can be sent through the "Address" parameter (only the two LSBytes are used ; if OpcodeLength is less than 24 bits, only the MSBits of the two LSBytes are used),
- the OpcodeLength is the sum of Opcode and Address lengths in bits  
(if OpcodeLength is 0, nothing is sent ;  
if OpcodeLength < 9, just Opcode is sent ;  
if 8 < OpcodeLength < 25, Opcode then Address are sent),
- the "AddressLength" parameter is not used.

For Q24X6 and P32X6 products :

Up to 32 bits can be sent through the "Opcode" parameter, according to the "OpcodeLength" parameter (in bits).  
if OpcodeLength is less than 32 bits, only MSBits are used.

Up to 32 bits can be sent through the "Address" parameter, according to the "AddressLength" parameter (in bits).  
if AddressLength is less than 32 bits, only MSBits are used.



- **For I2C soft bus :**  
Only the "Address" parameter is used.  
This parameter is the slave address byte. This is a 7-bits address, shift to left from 1 bit, padded with the LSB set to 0 (to write), and sent on the I2C bus before performing the writing operation.
- **For Parallel bus :**  
Only the "Address" parameter is used.  
This parameter is used to set the A2 pin value ; it can be set to following values :  
WM\_BUS\_PARA\_ADDRESS\_A2\_SET, to set the A2 pin ;  
WM\_BUS\_PARA\_ADDRESS\_A2\_RESET, to reset the A2 pin

*pDataToWrite :*

Buffer containing data to write on the requested bus.

*NbBytes*

Size (in bytes) of the pDataToWrite buffer.

### 3.8.5.2 Returned Values

On successful completion, the function returns the number of bytes written.

Otherwise, the function will return a negative error value (cf § 3.8.2).

### 3.8.6 The `wm_busRead` Function

The `wm_busRead` function allows to read on a bus previously allocated by the `wm_busOpen` function.

Its prototype is :

```
s32 wm_busRead (      s32 Handle
                    wm_busAccess_t * pAccessMode,
                    void * pDataToRead,
                    u32 NbBytes );
```

#### 3.8.6.1 Parameters

*Handle :*

Handle of the bus device to read from, returned by `wm_busOpen` function.

*pAccessMode :*

Mode to use to access the device.

This parameter is defined using the following type :

```
typedef struct
{
    u32 Address;
    u32 Opcode;
    u8 OpcodeLength;
    u8 AddressLength;
} wm_busAccess_t;
```

This parameter is processed differently according the bus type :

- **For SPI bus :**

For Q24X3 and P32X3 products :

one byte can be sent through the "Opcode" parameter (only the LSByte is used ; if OpcodeLength is less than 8 bits, only the MSBits of the LSByte are used),

two bytes can be sent through the "Address" parameter (only the two LSBytes are used ; if OpcodeLength is less than 24 bits, only the MSBits of the two LSBytes are used),

the OpcodeLength is the sum of Opcode and Address lengths in bits: if OpcodeLength = 0, nothing is sent ;  
if OpcodeLength < 9, Opcode only is sent ;  
if 8 < OpcodeLength < 25, Opcode first and then Address are sent,

the "AddressLength" parameter is not used.

For Q24X6 and P32X6 products :

Up to 32 bits can be sent through the "Opcode" parameter, according to the "OpcodeLength" parameter (in bits).

If OpcodeLength is less than 32 bits, only MSBits are used.

Up to 32 bits can be sent through the "Address" parameter, according to the AddressLength parameter (in bits). If AddressLength is less than 32 bits, only MSBits are used.

- **For I2C soft bus :**

Only the "Address" parameter is used as slave address byte.

This is a 7-bits address, shift to left from 1 bit, padded with the LSB set to 1 (to read), and sent on the I2C bus before performing the reading operation.

- **For Parallel bus :**

Only the "Address" parameter is used to set the A2 pin value; the possible values are:

WM\_BUS\_PARA\_ADDRESS\_A2\_SET, to set the A2 pin;

WM\_BUS\_PARA\_ADDRESS\_A2\_RESET, to reset the A2 pin

*pDataToRead :*

Buffer containing data read from the requested bus.

*NbBytes*

Size (in bytes) of the pDataToRead buffer.

### 3.8.6.2 Returned Values

On successful completion, the function returns the number of bytes read.

Otherwise, the function will return a negative error value (cf §3.8.2).

### 3.8.7 Error codes values

Here are some numerical error codes:

Error code	Error value
WM_BUS_PROC_DONE	0
WM_BUS_MODE_UNKNOWN_TYPE	-1
WM_BUS_UNKNOWN_TYPE	-11
WM_BUS_BAD_PARAMETER	-12
WM_BUS_SPI1_ALREADY_USED	-13
WM_BUS_I2C_SOFT_ALREADY_USED	-15
WM_BUS_PARALLEL_ALREADY_USED	-16
WM_BUS_UNKNOWN_HANDLE	-21
WM_BUS_HANDLE_NOT_OPENED	-22
WM_BUS_NO_MORE_HANDLE_FREE	-23
WM_BUS_NOT_CONNECTED_ON_I2C	-31
WM_BUS_NOT_ALLOWED_ADDRESS	-32
WM_BUS_I2C_SOFT_GPIO_NOT_GPIO	-33
WM_BUS_SPI_LCDEN_NOT_FREE	-34
WM_BUS_SPI_AUX_NOT_FREE	-35
WM_BUS_SPI_GPIO_CS_NOT_GPIO	-36
WM_BUS_SPI_GPIO_CS_NOT_FREE	-37

## 3.9 Scratch Memory API

The Scratch Memory API does no more exists since Open AT® V3.00; to implement the Over The Air download, the Application & Data Storage API has to be used instead.

The Scratch Memory functions below still exist, but all will always return the WM\_SCRATCH\_MEM\_NOTAVAIL error code :

```
s32 wm_scmOpen ( u8 Mode );  
s32 wm_scmClose ( void );  
s32 wm_scmRead ( u32 Size, void * Data );  
s32 wm_scmWrite ( u32 Size, void * Data );  
s32 wm_scmSeek ( s32 Pos );  
s32 wm_scmInstall ( void );
```

### 3.9.1 Required Header

This API is defined in `wm_scmem.h` header file.

This file is included by `wm_apm.h`.

### 3.9.2 Returned values definition

WM\_SCRATCH\_MEM\_NOTAVAIL: the Scratch Memory is not available.

## 3.10 Lists management API

### 3.10.1 Required Header

This API is defined in `wm_list.h` header file.  
This file is included by `wm_apm.h`.

### 3.10.2 Types definition

#### 3.10.2.1 The `wm_lst_t` type

This type is used to handle a list created by the list API.

```
typedef void * wm_lst_t;
```

#### 3.10.2.2 The `wm_lstTable_t` structure

This structure is used to define a comparison callback and an Item destruction callback :

```
typedef struct  
{  
    s16 ( * CompareItem ) ( void *, void * );  
    void ( * FreeItem ) ( void * );  
} wm_lstTable_t;
```

The `CompareItem` callback is called every time the lists API needs to compare two items.

It returns:

- OK if both provided elements are considered to be similar.
- -1 if the first element is considered to be smaller than the second one.
- 1 if the first element is considered to be greater than the second one.

If the `CompareItem` callback is set to NULL, the `wm_strcmp` function will be used by default.

The `FreeItem` callback is called each time the list API needs to delete an item. It should then perform its specific processing before releasing the provided pointer.

If the `FreeItem` callback is set to NULL, the `wm_osReleaseMemory` function will be used by default.

### 3.10.3 The `wm_lstCreate` Function

The `wm_lstCreate` function allows to create a list, using the provided attributes and callbacks.

Its prototype is :

```
wm_lst_t wm_lstCreate (    u16 Attr,  
                          wm_lstTable_t * funcTable );
```

#### 3.10.3.1 Parameters

*Attr :*

List attributes, which can be combined by a logical OR among following defined values :

- `WM_LIST_NONE` : no specific attribute ;
- `WM_LIST_SORTED` : this list is a sorted one (see `wm_lstAddItem` and `wm_lstInsertItem` descriptions for more details) ;
- `WM_LIST_NODUPLICATES` : this list does not allow duplicate items (see `wm_lstAddItem` and `wm_lstInsertItem` descriptions for more details).

*funcTable :*

Pointer on a structure containing the comparison and the item destruction callbacks.

#### 3.10.3.2 Returned Values

This function will return a list pointer corresponding to the created list. This one must be used in all further operations on this list.

### 3.10.4 The `wm_lstDestroy` Function

The `wm_lstDestroy` function allows to clear and then destroy the provided list.

Its prototype is :

```
void wm_lstDestroy ( wm_lst_t list );
```

*list :*

The list to destroy.

#### **Note**

This function calls the `FreeItem` callback (if defined) on each item to delete it, before destroying the list.

### 3.10.5 The `wm_lstClear` Function

The `wm_lstClear` function allows to clear all the provided list items, without destroying the list itself (please refer to `wm_lstDeleteItem()` function for notes on item deletion).

Its prototype is :

```
void wm_lstClear ( wm_lst_t list );
```

*list* : the list to clear.

#### Note

This function calls the `FreeItem` callback (if defined) on each item to delete it.

### 3.10.6 The `wm_lstGetCount` Function

The `wm_lstGetCount` function returns the current item count.

Its prototype is :

```
u16 wm_lstGetCount ( wm_lst_t list );
```

#### 3.10.6.1 Parameters

*list* :

The list from which to get the item count.

#### 3.10.6.2 Returned Values

The number of items of the provided list. The function returns 0 if the list is empty.

### 3.10.7 The `wm_lstAddItem` Function

The `wm_lstAddItem` function allows to add an item to the provided list.

Its prototype is :

```
s16 wm_lstAddItem (    wm_lst_t list
                      void * item );
```

#### 3.10.7.1 Parameters

*list* : The list to add an item to.

*item* : The item to add to the list.

#### 3.10.7.2 Returned Values

The position of the added item, or ERROR if an error occurred.

#### Notes:

- ❑ The *item* pointer should not point on a `const` or local buffer, as it will be released in any item destruction operation.
- ❑ If the list has the `WM_LIST_SORTED` attribute, the item will be inserted in the appropriate place after calling of the `CompareItem` callback (if defined). Otherwise, the item is appended at the end of the list.
- ❑ If the list has the `WM_LIST_NODUPLICATES`, the item will not be inserted if the `CompareItem` callback (if defined) returns 0 on any previously added item. In this case, the returned index is the existing item's one.

### 3.10.8 The `wm_lstInsertItem` Function

The `wm_lstInsertItem` function allows to insert an item to the provided list at the given location.

Its prototype is :

```
s16 wm_lstInsertItem (  wm_lst_t list
                       void * item
                       u16 index );
```

#### 3.10.8.1 Parameters

*list* : The list to add an item to.

*item* : The item to add to the list.

*index* : The location where to add the item.

#### 3.10.8.2 Returned Values

The position of the added item, or ERROR if an error occurred.

#### 3.10.8.3 Notes

- ❑ The *item* pointer should not point on a `const` or local buffer, as it will be released in any item destruction operation.
- ❑ This function does not care of the list attributes and will always insert the provided item at the given index.

### 3.10.9 The `wm_lstGetItem` Function

The `wm_lstGetItem` function allows to read an item from the provided list, at the given index.

Its prototype is :

```
void * wm_lstGetItem ( wm_lst_t list  
                      u16 index );
```

#### 3.10.9.1 Parameters

*list* :

The list from which to get the item.

*index* :

The location where to get the item.

#### 3.10.9.2 Returned Values

A pointer on the requested item, or NULL if the index was not valid.

### 3.10.10 The `wm_lstDeleteItem` Function

The `wm_lstDeleteItem` function allows to delete an item of the provided list at the given indexes.

Its prototype is :

```
s16 wm_lstDeleteItem ( wm_lst_t list  
                      u16 index );
```

#### 3.10.10.1 Parameters

*list* :

The list to delete an item from.

*index* :

The location where to delete the item.

#### 3.10.10.2 Returned Values

The number of remaining items in the list, or ERROR if an error did occur.

#### **Note**

This function calls the `FreeItem` callback (if defined) on the requested item to delete it.



### 3.10.11 The `wm_lstFindItem` Function

The `wm_lstFindItem` function allows to find out an item in the provided list.

Its prototype is :

```
s16 wm_lstFindItem (  wm_lst_t list
                    void * item );
```

#### 3.10.11.1 Parameters

*list* : The list where to search.

*item* : The item to find.

#### 3.10.11.2 Returned Values

The index of the found item if any, ERROR otherwise.

#### Note

This function calls the `CompareItem` callback (if defined) on each list item, until it returns 0.

### 3.10.12 The `wm_lstFindAllItem` Function

The `wm_lstFindAllItem` function allows to find all items matching the provided one, in the given list.

Its prototype is :

```
s16 * wm_lstFindAllItem ( wm_lst_t list
                        void * item );
```

#### 3.10.12.1 Parameters

*list* : The list where to search.

*item* : The item to find.

#### 3.10.12.2 Returned Values

A s16 buffer containing the indexes of all the items found, and terminated by ERROR.

Important remark : this buffer should be released by the application when its processing is done.

#### Notes

- ❑ This function calls the `CompareItem` callback (if defined) on each list item to get all those which match with the provided item.
- ❑ This function should be used only if the list can not be changed during the resulting buffer processing. Otherwise the `wm_lstFindNextItem` should be used.

### 3.10.13 The `wm_lstFindNextItem` Function

The `wm_lstFindNextItem` function allows to find the next item index of the given list, which correspond with the provided one.

Its prototype is :

```
s16 wm_lstFindNextItem (    wm_lst_t list
                           void * item );
```

#### 3.10.13.1 Parameters

*list* : The list to search in.

*item* : The item to find.

#### 3.10.13.2 Returned Values

The index of the next found item if any, otherwise ERROR.

#### Note

- This function calls the `CompareItem` callback (if defined) on each list item to get those which match with the provided item. It should be called until it returns ERROR, in order to get the index of all items corresponding to the provided one. The difference with the `wm_lstFindAllItem` function is that, even if the list is updated between two calls to `wm_lstFindNextItem`, the function will not return a previously found item. To restart a search with the `wm_lstFindNextItem`, the `wm_lstResetItem` should be called first.

### 3.10.14 The `wm_lstResetItem` Function

The `wm_lstResetItem` function allows to reset all previously found items by the `wm_lstFindNextItem` function.

Its prototype is :

```
void wm_lstResetItem (    wm_lst_t list
                          void * item );
```

#### 3.10.14.1 Parameters

*list* : The list to search in.

*item* : The item to search, in order to reset all previously found items.

#### Note

- This function calls the `CompareItem` callback (if defined) on each list item to get those which match with the provided one.

## 3.11 Sound API

### 3.11.1 Required header

This API is defined in `wm_snd.h` header file.

This file is included by `wm_apm.h`.

### 3.11.2 The `wm_sndTonePlay` Function

This function allows a tone to be played on the current speaker or on the buzzer. Frequency, gain and duration can be specified.

Its prototype is:

```
s32 wm_sndTonePlay (  wm_snd_dest_e Destination,
                      u16      Frequency,
                      u8       Duration,
                      u8       Gain );
```

#### 3.11.2.1 Parameters

*Destination:*

Destination of the requested tone to play: speaker or buzzer.

```
typedef enum          {
WM_SND_DEST_BUZZER,
WM_SND_DEST_SPEAKER,
WM_SND_DEST_GSM      /* do not use */
} wm_snd_dest_e;
```

*Frequency:*

For speaker : range is 1 Hz to 3999 Hz.

For buzzer : range is 1 Hz to 50000 Hz.

*Duration:*

This parameter sets tone duration (in unit of 20 ms).

Remark : when `<duration>` = 0, the duration is infinite, and the tone should be stopped by `wm_sndToneStop`.

**Gain:**

This parameter sets the tone gain.  
Range of values is from 0 to 15.

<gain>	Speaker (db)	Buzzer (db)
0	0	-0.25
1	-0.5	-0.5
2	-1	-1
3	-1.5	-1.5
4	-2	-2
5	-3	-3
6	-6	-6
7	-9	-9
8	-12	-12
9	-15	-15
10	-18	-18
11	-24	-24
12	-30	-30
13	-36	-40
14	-42	-infinite
15	-infinite	-infinite

**3.11.2.2 Returned values**

OK on success, or a negative error value

**3.11.2.3 Example:**

An example of playing tone:

```
wm_sndTonePlay ( WM_SND_DEST_BUZZER, 1000, 0, 9 );
```

### 3.11.3 The `wm_sndToneStop` Function

This function stops playing a tone on the current speaker or on the buzzer.

Its prototype is:

```
s32 wm_sndToneStop ( wm_snd_dest_e Destination );
```

#### 3.11.3.1 Parameters

*Destination:*

Destination of the current playing tone to stop: speaker or buzzer.

#### 3.11.3.2 Returned values

OK on success, or a negative error value

#### 3.11.3.3 Example:

An example of stopping tone:

```
wm_sndToneStop ( WM_SND_DEST_BUZZER );
```

### 3.11.4 The `wm_sndDtmfPlay` Function

This function allows a DTMF tone to be played on the current speaker or over the GSM network (in communication only). DTMF, gain (only for speaker) and duration can be specified.

Remark: it is not possible to play DTMF on buzzer.

Its prototype is:

```
s32  wm_sndDtmfPlay (  wm_snd_dest_e Destination,
                      ascii      Dtmf,
                      u8         Duration,
                      u8         Gain );
```

#### 3.11.4.1 Parameters

*Destination:*

Destination of the requested DTMF tone to play: speaker or/and over the GSM network (in communication only).

```
typedef enum          {
WM_SND_DEST_BUZZER, /* do not use */
WM_SND_DEST_SPEAKER,
WM_SND_DEST_GSM
} wm_snd_dest_e;
```

*Dtmf:*

Value must be in { '0' - '9', '\*', '#', 'A', 'B', 'C', 'D' }

*Duration:*

This parameter sets tone duration (in unit of 20 ms).

Remark : when **<duration>** = 0, the duration is infinite, and the tone should be stopped by `wm_sndDtmfStop`.

*Gain:*

Only for speaker.

This parameter sets the tone gain.

Range of values is from 0 to 15. (see § 3.11.2)

#### 3.11.4.2 Returned values

OK on success, or a negative error value

#### 3.11.4.3 Example:

An example of playing DTMF:

```
wm_sndDtmfPlay ( WM_SND_DEST_SPEAKER, 'A', 100, 9 );
```

### 3.11.5 The `wm_sndDtmfStop` Function

This function stops playing a dtmf on the current speaker or over the GSM network (in communication only).

Its prototype is:

```
s32 wm_sndDtmfStop ( wm_snd_dest_e Destination );
```

#### 3.11.5.1 Parameters

*Destination:*

Destination of the current playing tone to stop: speaker or GSM network (in communication only).

#### 3.11.5.2 Returned values

OK on success, or a negative error value

#### 3.11.5.3 Example:

An example of stopping DTMF:

```
wm_sndDtmfStop ( WM_SND_DEST_SPEAKER );
```

### 3.11.6 The `wm_sndMelodyPlay` Function

This function plays a melody. Destination, Melody, Tempo, Cycle and gain can be specified.

Its prototype is:

```
s32  wm_melody_play (  wm_snd_dest_e Destination,
                      u16*      Melody,
                      u16      Tempo,
                      u8       Cycle,
                      u8       Gain );
```

#### 3.11.6.1 Parameters

*Destination:*

Destination of the melody to play: speaker or buzzer.

```
typedef enum          {
    WM_SND_DEST_BUZZER,
    WM_SND_DEST_SPEAKER,
    WM_SND_DEST_GSM      /* do not use */
} wm_snd_dest_e;
```

*Melody:*

Melody to play. A melody is defined by an u16 table, where each element defines a note event, with a duration and a sound definition.

```
// Melody sample
const u16 MyMelody [ ]=
{
    WM_SND_E1 | WM_SND_QUAVER ,
    WM_SND_F1 | WM_SND_MBLACK ,
    WM_SND_G6S | WM_SND_QUAVER ,
};

typedef enum {
    WM_SND_C0 , // C0
    WM_SND_C0S , // C0#
    WM_SND_D0 , // D0
    WM_SND_D0S , // D0#
    WM_SND_E0 , // E0
    WM_SND_F0 , // F0
    WM_SND_F0S , // F0#
    WM_SND_G0 , // G0
    WM_SND_G0S , // G0#
    WM_SND_A0 , // A0
    WM_SND_A0S , // A0#
    WM_SND_B0 , // B0
    WM_SND_C1 , // C1
    ...
    WM_SND_NO_SOUND=0xFF
} wm_sndNote_e;
```



```
#define WM_SND_ROUND      0x1000
#define WM_SND_MWHITEP   0x0C00
#define WM_SND_MWHITE    0x0800
#define WM_SND_MBLACKP   0x0600
#define WM_SND_MBLACK    0x0400
#define WM_SND_QUAVERP   0x0300
#define WM_SND_QUAVER    0x0200
#define WM_SND_MSHORT    0x0100
```

*Tempo:*

Tempo to apply (duration a black x 20 ms).

*Cycle:*

number of times that the melody should be played (0 = infinite)

*Gain:*

Volume to apply, range of values is 0 to 15.

### 3.11.6.2 Returned values

OK on success, or a negative error value

### 3.11.6.3 Example:

An example of playing melody:

```
wm_sndMelodyPlay ( WM_SND_DEST_SPEAKER, MyMelody, 6, 1, 9 );
```

### 3.11.7 The `wm_sndMelodyStop` Function

This function stops playing a melody on the current speaker or on the buzzer.

Its prototype is:

```
s32 wm_sndMelodyStop ( wm_snd_dest_e Destination );
```

#### 3.11.7.1 Parameters

*Destination:*

Destination of the current playing melody to stop: speaker or buzzer.

#### 3.11.7.2 Returned values

OK on success, or a negative error value

#### 3.11.7.3 Example:

An example of stopping melody:

```
wm_sndMelodyStop ( WM_SND_DEST_SPEAKER );
```

## 3.12 Standard Library

### 3.12.1 Required Header

This API is defined in `wm_stdio.h` header file.  
This file is included by `wm_apm.h`.

### 3.12.2 Standard C function set

The available standard APIs are defined below:

```

ascii * wm_strcpy      ( ascii * dst, ascii * src );
ascii * wm_strncpy    ( ascii * dst, ascii * src, u32 n );
ascii * wm_strcat     ( ascii * dst, ascii * src );
ascii * wm_strncat    ( ascii * dst, ascii * src, u32 n );
u32    wm_strlen      ( ascii * str );
s32    wm_strcmp      ( ascii * s1, ascii * s2 );
s32    wm_strncmp     ( ascii * s1, ascii * s2, u32 n );
s32    wm_stricmp     ( ascii * s1, ascii * s2 );
s32    wm_strnicmp    ( ascii * s1, ascii * s2, u32 n );
ascii * wm_memset     ( ascii * dst, ascii c, u32 n );
ascii * wm_memcpy     ( ascii * dst, ascii * src, u32 n );
s32    wm_memcmp      ( ascii * dst, ascii * src, u32 n );
ascii * wm_itoa       ( s32 a, ascii * szBuffer );
s32    wm_atoi        ( ascii * p );
u8     wm_sprintf     ( ascii * buffer, ascii * fmt, ... );
    
```

Important remark about GCC compiler :

When using GCC compiler, due to internal standard C library architecture, it is strongly not recommended to use the "%f" mode in the `wm_sprintf` function in order to convert a float variable to a string. This will lead to an ARM exception (product reset).  
A workaround for this conversion will be :

```

float MyFloat; // float to display
ascii MyString [ 100 ]; // destination string
s16 d,f;
d = (s16) MyFloat * 1000; // Decimal precision : 3 digits
f = ( MyFloat * 1000 ) - d; // Decimal precision : 3 digits
wm_sprintf ( MyString, "%d.%03d", (s16)MyFloat, f ); // Decimal
precision : 3 digits
    
```

### 3.12.3 String processing function set

Some string processing functions are also available in this standard API.

**Note :** all following functions will result as an ARM exception if a requested `ascii *` parameter is NULL.

```

ascii   wm_isascii      ( ascii c );
Returns c if it is an ascii character ( 'a'/'A' to 'z'/'Z'), 0 otherwise.
ascii   wm_isdigit     ( ascii c );
Returns c if it is a digit character ( '0' to '9'), 0 otherwise.
ascii   wm_ishexa      ( ascii c );
Returns c if it is a hexadecimal character ( '0' to '9', 'a'/'A' to 'f'/'F'), 0
otherwise.
bool    wm_isnumstring ( ascii * string );
Returns TRUE if string is a numeric one, FALSE otherwise.
bool    wm_ishexastring ( ascii * string );
Returns TRUE if string is a hexadecimal one, FALSE otherwise.
bool    wm_isphonestring ( ascii * string );
Returns TRUE if string is a valid phone number (national or international
format), FALSE otherwise.
u32    wm_hexatoi      ( ascii * src, u16 iLen );
If src is a hexadecimal string, converts it to a returned u32 of the given
length, and 0 otherwise. As an example: wm_hexatoi ("1A", 2) returns 26,
wm_hexatoi ("1A", 1) returns 1
u8 *    wm_hexatoibuf   ( u8 * dst, ascii * src );
If src is a hexadecimal string, converts it to an u8 * buffer and returns a
pointer on dst, and NULL otherwise. As an example, wm_hexatoibuf (dst,
"1F06") returns a 2 bytes buffer: 0x1F and 0x06 )
ascii * wm_itohexa      ( ascii * dst, u32 nb, u8 len );
Converts nb to a hexadecimal string of the given length and returns a
pointer on dst. For example, wm_itohexa ( dst, 0xD3, 2 ) returns "D3",
wm_itohexa ( dst, 0xD3, 4 ) returns "00D3".
ascii * wm_ibuftohexa   ( ascii * dst, u8 * src, u16 len );
Converts the u8 buffer src to a hexadecimal string of the given length and
returns a pointer on dst. Example with the src buffer filled with 3 bytes
(0x1A, 0x2B and 0x3C), wm_ibuftohexa (dst, src, 3) returns "1A2B3C".
u16    wm_strSwitch     ( const ascii * strTest, ... );
This function must be called with a list of strings parameters, terminated by
NULL. strTest is compared with each of these strings (on the length of each
string, with no matter of the case), and returns the index (starting from 1) of
the string which matches if any, 0 otherwise.
Example :
wm_strSwitch ("TEST match", "test", "no match", NULL) returns 1,
wm_strSwitch ("nomatch", "nomatch a", "nomatch b", NULL) returns 0.
ascii * wm_strRemoveCRLF ( ascii * dst, ascii * src, u16 size );
Copy in dst buffer the content of src buffer, removing CR (0x0D) and LF
(0x0A) characters, from the given size, and returns a pointer on dst.
ascii * wm_strGetParameterString ( ascii * dst,
const ascii * src,
u16 Position );
If src is a string formatted as an AT response (for example "+RESP: 1,2,3")
or as an AT command (for example "AT+CMD=1,2,3"), the function copies
the parameter at Position offset (starting from 1) if it is present in the dst
buffer, and returns a pointer on dst. It returns NULL otherwise.
Example :
wm_strGetParameterString (dst, "+WIND: 4", 1) returns "4",
wm_strGetParameterString (dst, "+WIND: 5,1", 2) returns "1",
wm_strGetParameterString (dst, "AT+CMGL=\"ALL\"", 1) returns "ALL".
    
```

### 3.13 Application & Data storage API

This API provides storage cells, where to store data or "dwl" files in order to update the product software (a "dwl" file may be a Wavecom Core Software update, an Open AT<sup>®</sup> application, or an E2P configuration file).

The total Application & Data Storage volume size is 512KBytes

#### 3.13.1 Required Header

This API is defined in `wm_ad.h` header file.

This file is included by `wm_apm.h`.

#### 3.13.2 Returned values definition

WM_AD_ERROR_UNDEFINED	Generic error code ;
WM_AD_BAD_ARGS	Function arguments error ;
WM_AD_BAD_FUNCTION	Bad function call ;
WM_AD_FORBIDDEN	Access denied or illegal operation attempt ;
WM_AD_OVERFLOW	Memory overflow ;
WM_AD_REACHED_END	No more elements to enumerate ;
WM_AD_NOT_AVAILABLE	Function not available (no initialisation done or operation not supported) ;
WM_AD_CLEANING_RQD	A cleaning operation is required to perform the requested command.

### 3.13.3 The `wm_adAllocate` Function

The `wm_adAllocate` function allows to allocate a new cell in the Application & Data storage space.

Its prototype is :

```
s32 wm_adAllocate ( u32          CellId,  
                  u32          Size,  
                  wm_adHandle_t * Handle );
```

#### 3.13.3.1 Parameters

**CellId**

Unique identifier of the cell to allocate.

**Size**

Size in bytes of the cell to allocate.

The real used size in flash memory will be the data size + the header size. The header size is variable, with an average of 16 bytes.

If the Cell size is unknown at allocation time, the `WM_AD_UNDEFINED` may be used. In this case, the next `wm_adAllocate` function calls will all fail, until the undefined size cell is finalized.

**Handle**

Returned handle on the new allocated cell.

#### 3.13.3.2 Returned Values

This function will return OK if successful, otherwise, it will return an error value (please refer to § [3.9.2 "Returned values definition"](#)).

### 3.13.4 The `wm_adRetrieve` Function

The `wm_adRetrieve` function allows to initialize a handle on an already allocated cell.

Its prototype is :

```
s32 wm_adRetrieve ( u32          CellId,  
                  wm_adHandle_t * Handle );
```

#### 3.13.4.1 Parameters

**CellId**

Unique identifier of the cell to retrieve.

**Handle**

Returned handle on the retrieved cell.

#### 3.13.4.2 Returned Values

This function will return OK if successful, otherwise, it will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.5 The `wm_adFindInit` Function

The `wm_adFindInit` function initializes a cell search, between the two provided cell identifiers.

Its prototype is :

```
s32 wm_adFindInit ( u32           MinCellId,  
                  u32           MaxCellId,  
                  wm_adBrowse_t * BrowseInfo );
```

#### 3.13.5.1 Parameters

**MinCellId**

Minimum value for wanted cell identifiers.

**MaxCellId**

Maximum value for wanted cell identifiers.

**BrowseInfo**

Returned browse information, to use with the `wm_adFindNext()` function.

#### 3.13.5.2 Returned Values

This function will return OK if successful, otherwise, it will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.6 The `wm_adFindNext` Function

The `wm_adFindNext` function performs a search on the browse informations provided by the `wm_adFindInit()` function.

Its prototype is :

```
s32 wm_adFindNext ( wm_adBrowse_t * BrowseInfo  
                  wm_adHandle_t * Handle );
```

#### 3.13.6.1 Parameters

**BrowseInfo**

Browse informations, returned by the `wm_adFindInit()` function.

**Handle**

Next found cell handle.

#### 3.13.6.2 Returned Values

This function will return OK if an handle is found, or `WM_AD_REACHED_END` if there are no more corresponding handles.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.7 The `wm_adWrite` Function

The `wm_adWrite` function appends data in an allocated cell.

Its prototype is :

```
s32 wm_adWrite ( wm_adHandle_t * Handle,  
                u32          Size,  
                void *      Data );
```

#### 3.13.7.1 Parameters

**Handle**

Handle on the allocated cell (returned by the `wm_adAllocate` or the `wm_adResume` functions).

**Size**

Number of bytes to write.

**Data**

Data source buffer.

#### 3.13.7.2 Returned Values

This function will return OK if successful, otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.8 The `wm_adFinalise` Function

The `wm_adFinalise` function finalises the creation of a new record. Once completed, nothing more can be written in the cell.

Its prototype is :

```
s32 wm_adFinalise ( wm_adHandle_t * Handle );
```

#### 3.13.8.1 Parameters

**Handle**

Handle on the allocated cell (returned by the `wm_adAllocate` or the `wm_adResume` functions) to finalise.

#### 3.13.8.2 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.9 The `wm_adResume` Function

The `wm_adResume` function allows to resume an interrupted write operation, on a non-yet finalised call.

Its prototype is :

```
s32 wm_adResume (    wm_adHandle_t *    Handle );
```

#### 3.13.9.1 Parameters

##### Handle

Handle on the non-yet finalized cell (returned by the `wm_adFindNext` or the `wm_adRetrieve` functions).

#### 3.13.9.2 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.10 The `wm_adInfo` Function

The `wm_adInfo` function provides informations on the requested handle.

Its prototype is :

```
s32 wm_adInfo (    wm_adHandle_t *    Handle  
                  wm_adInfo_t *    Info );
```

#### 3.13.10.1 Parameters

##### Handle

Handle on the allocated cell from which to get information.

##### Info

Data returned on the provided handle, using following type :

```
typedef struct  
{  
    u32      ID,                // Cell identifier  
    u32      size,             // Cell size  
    void *   data,             // Pointer on stored data  
    u32      remaining,        // Remaining writable space  
    bool     finalised         // TRUE if entry is finalised  
} wm_adInfo_t;
```

#### 3.13.10.2 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).



### 3.13.11 The `wm_adDelete` Function

The `wm_adStats` function allows to delete the requested record. The cell is not physically deleted ; it will be on next recompaction process.

Its prototype is :

```
s32 wm_adDelete ( wm_adHandle_t * Handle );
```

#### 3.13.11.1 Parameters

##### Handle

Handle on the cell to delete.

#### 3.13.11.2 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.12 The `wm_adStats` Function

The `wm_adStats` function provides global Application & Data space informations.

Its prototype is :

```
s32 wm_adStats ( wm_adStats_t * Info );
```

#### 3.13.12.1 Parameters

##### Info

Informations returned on the provided handle, using following type :

```
typedef struct
{
    u32      freemem,           // Free memory size
    u32      deletedmem,      // Deleted memory size
    u32      totalmem,        // Total memory size
    u16      numobjects,      // Number of objects
    u16      numdeleted,      // Number of deleted objects
    bool     need_recovery    // Set to TRUE, either if the volume
                             // state is not set to WM_AD_READY on
                             // startup, or if a cell allocated
                             // with an undefined size was not
                             // finalized before a product reset.
} wm_adStats_t;
```

#### 3.13.12.2 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.13 The `wm_adSpaceState` Function

The `wm_adSpaceState` function provides the Application & Data space current state.

Its prototype is :

```
wm_adSpaceState_e wm_adSpaceState ( void );
```

#### 3.13.13.1 Returned Values

This return value uses the following type :

```
typedef enum
{
    WM_AD_READY = 1,      // Space is ready
    WM_AD_NOTAVAIL,      // Space is not available
    WM_AD_REPAIR,        // A product reset has occurred since last
                        // recompactation process. The application has
                        // to call wm_adRecompactInit to continue
                        // this process.
} wm_adSpaceState_e;
```

### 3.13.14 The `wm_adFormat` Function

The `wm_adFormat` function destroys the whole Application & Data space stored data.

Its prototype is :

```
s32 wm_adFormat ( void );
```

#### 3.13.14.1 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.15 The `wm_adRecompactInit` Function

The `wm_adRecompactInit` function starts the recompactation process. The process steps are then done by the `wm_adRecompact()` function.

Its prototype is :

```
s32 wm_adRecompactInit ( void );
```

Note: when `wm_adRecompactInit` is called, **no other A&D function should be called** (except `wm_adRecompact`) **before recompactation completion**. If the recompactation is interrupted by a product reset, `wm_adSpaceState` function will return WM\_AD\_REPAIR state.

#### 3.13.15.1 Returned Values

This function will return OK if successful.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.16 The `wm_adRecompact` Function

The `wm_adRecompact` function performs a new recompaction step. The recompaction process has to be initialised by the `wm_adRecompactInit()` function.

Its prototype is :

```
s32 wm_adRecompact ( void );
```

#### 3.13.16.1 Returned Values

This function will return the completed percentage if successful. It must be called until the returned value is 100.

Otherwise, the function will return an error value (cf § [3.9.2 "Returned values definition"](#)).

### 3.13.17 The `wm_adInstall` Function

The `wm_adInstall` function allows to install the content of the provided cell, if it is a "dwl" file (a Wavecom Core Software update, an Open AT<sup>®</sup> application, or an E2P configuration file).

Its prototype is :

```
s32 wm_adInstall ( wm_adHandle_t * Handle );
```

#### 3.13.17.1 Parameters

##### Handle

Handle on the cell to install.

#### 3.13.17.2 Returned Values

This function will reset the product and install the "dwl" file on success.

The `InitType` parameter of all the `Init` functions will be set to either

`WM_APM_DOWNLOAD_SUCCESS` (on install success) or

`WM_APM_DOWNLOAD_ERROR` (if the ".dwl" file is corrupted).

Otherwise, the function will return an error value (cf § 3.9.2 Returned values definition).

## 3.14 [Deprecated] WAP API

The WAP feature has been removed since the Open AT<sup>®</sup> v3.03 version. The WAP APIs do not exist anymore in the Basic library.

### 3.15 GPS API

This API provides a GPS interface to Open AT<sup>®</sup> applications downloaded on a Q2501 product. This API is only enabled on this product, and only if the GPS device is set in internal mode (controlled by the Wavecom module, i.e. the AT+WGPSCONF=0,1 mode has to be set; when this parameter value is changed, the product has to be reset to take the new value into account).

#### 3.15.1 Required Header

This API is defined in `wm_gps.h` header file.  
This file is included by `wm_apm.h`.

#### 3.15.2 The `wm_gpsGetPosition` Function

The `wm_gpsGetPosition` function allows the Open AT<sup>®</sup> application to retrieve the current position read from the GPS device.

Its prototype is :

```
s8 wm_gpsGetPosition ( wm_gpsPosition_t * Position );
```

##### 3.15.2.1 Parameters

###### Position

GPS position read parameters, based on the type below :

```
typedef struct
{
    ascii UTC_time [S_UTC_TIME];           // hhmmss.sss
    ascii date [S_DATE];                   // ddmmyy
    ascii latitude [S_POSITION];           // ddmm.mmmmm
    ascii latitude_Indicator[S_INDICATOR]; // N - S
    ascii longitude [S_POSITION];          // dddmm.mmmmm
    ascii longitude_Indicator[S_INDICATOR]; // E - W
    ascii status[S_INDICATOR];
    ascii P_Fix[S_INDICATOR];
    ascii sat_used [S_SAT];                 // Satellites used
    ascii HDOP [S_HDOP]; // Horizontal Dilution of Precision
    ascii altitude [S_ALTITUDE];           // MSL Altitude
    ascii altitude_Unit[S_INDICATOR];
    ascii geoid_Sep [S_GEOID_SEP];         // geoid correction
    ascii geoid_Sep_Unit[S_INDICATOR];
    ascii Age_Dif_Cor [S_AGE_DIF_COR];     // Age of Differential
                                           // correction
    ascii Dif_Ref_ID [S_DIF_REF_ID];       // Diff Ref station ID
    ascii magneticVariation[S_COURSE];     // magnetic variation:
                                           // not available for
                                           // sirf technology
} wm_gpsPosition_t;
```

All fields are ascii zero terminated strings containing GPS information.

##### 3.15.2.2 Returned Values

This function will return ERROR if the current product is not a Q2501 one, or if the internal mode is not enabled. Otherwise, it will reply OK.

### 3.15.3 The `wm_gpsGetSpeed` Function

The `wm_gpsGetSpeed` function allows the Open AT<sup>®</sup> application to retrieve the current speed read from the GPS device.

Its prototype is :

```
s8 wm_gpsGetSpeed ( wm_gpsSpeed_t * Speed );
```

#### 3.15.3.1 Parameters

##### Speed

GPS speed read parameters, based on the type below :

```
typedef struct  
{  
    ascii course [S_COURSE];           // Degrees from true North  
    ascii speed_knots [S_SPEED];       // Speed in knots  
    ascii speed_km_p_hour [S_SPEED];  // Speed in km/h  
} wm_gpsSpeed_t;
```

All fields are ascii zero terminated strings containing GPS information.

#### 3.15.3.2 Returned Values

This function will return ERROR if the current product is not a Q2501 one, or if the internal mode is not enabled. Otherwise, it will reply OK.

### 3.15.4 The `wm_gpsGetSatview` Function

The `wm_gpsGetSatview` function allows the Open AT<sup>®</sup> application to retrieve the current satellite view read from the GPS device.

Its prototype is :

```
s8 wm_gpsGetSatview ( wm_gpsSatView_t * SatView );
```

#### 3.15.4.1 Parameters

##### SatView

GPS satellite view read parameters, based on the type below :

```
typedef struct
{
    u8 id;           // range 1 to 32
    u8 elevation;   // maximum 90
    u32 azimuth;    // range 0 to 359
    s8 SNR;         // range 0 to 99, -1 when not tracking
} wm_gpsSatellite_t;
```

All fields are integers containing GPS information about current satellite.

```
typedef struct
{
    u8 NB_Msg ;           // Number of messages
    u8 MSG_Number ;      // Message Number
    u8 Sat_view ;        // Satellites in view
    wm_gpsSatellite_t sat [NB_SAT_MAX]; // array for informations
                                        // about differents
                                        // satellites
} wm_gpsSatView_t;
```

The different fields contain information about the current satellite view. Each satellite information details are contained in the "sat" field.

#### 3.15.4.2 Returned Values

This function will return ERROR if the current product is not a Q2501 one, or if the internal mode is not enabled. Otherwise, it will reply OK.

## 4 FUNCTIONING

There are three different functioning modes, depending on the type of application. They are described in the following paragraphs.

### 4.1 Standalone External Application

This mode corresponds to the standard operation mode: no Embedded Application is active.

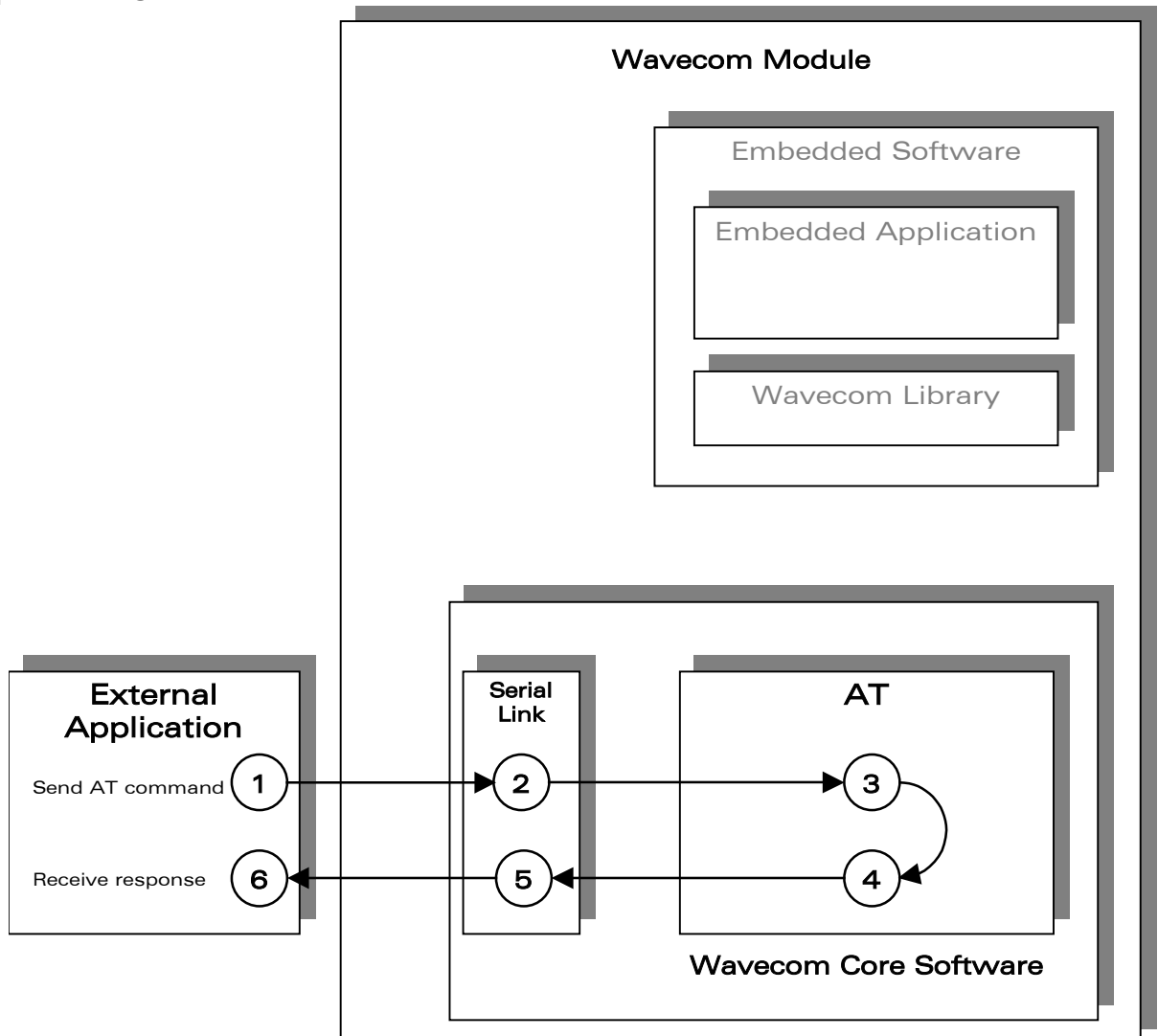


Figure 5: Standalone external application function

The steps are performed in the following sequence:

- 1) the External Application sends an AT command,
- 2) the serial link transmits the command to the AT processor function of the Wavecom Core Software,
- 3) the AT function processes the command,
- 4) the AT function sends an AT response to the External Application,
- 5) this response is sent through the serial link, and
- 6) the External Application receives the response.

**Note:**

This mode is also compatible with the mode described in § 4.2, where the AT function is in charge of dispatching the responses to the appropriate application.



### 4.2 Embedded Application in Standalone Mode

This mode is based on an Embedded Application driving the GSM product independently.

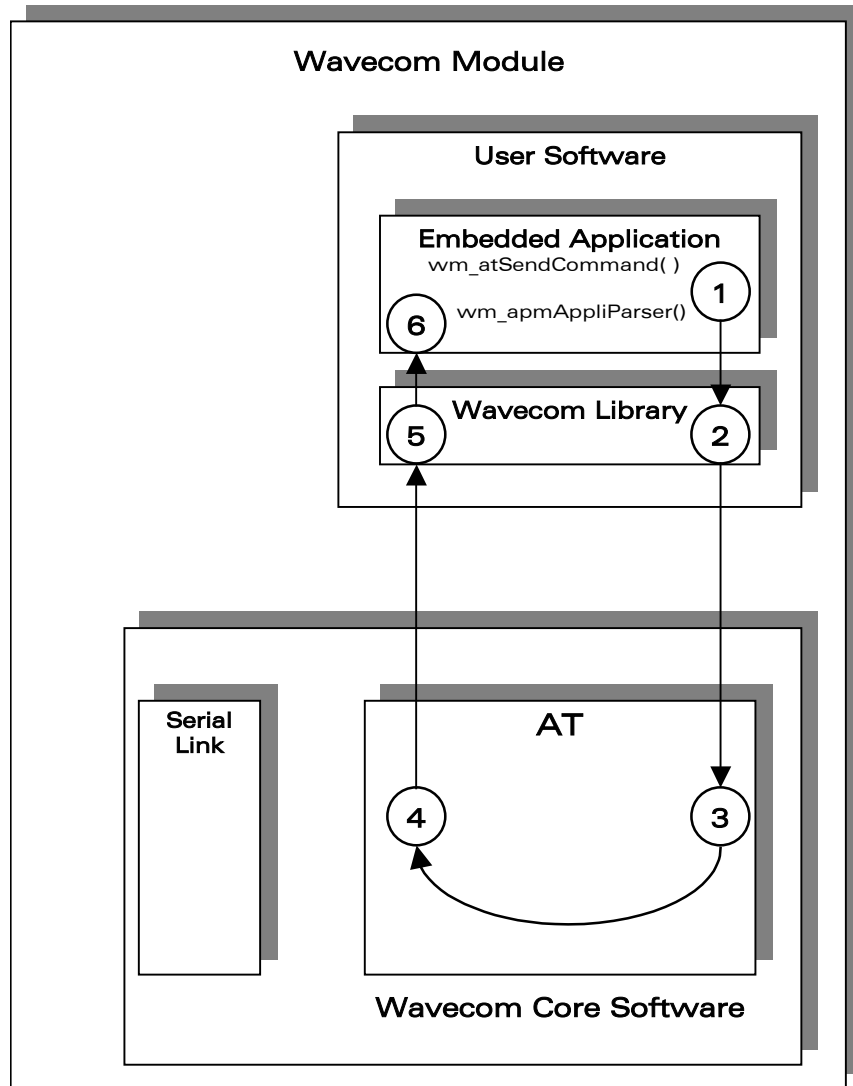


Figure 6: Embedded Application in standalone mode function

The steps are performed in the following sequence:

- 1) The Embedded Application calls the "wm\_atSendCommand" function to send an AT command.  
The response parameter is WM\_AT\_SEND\_RSP\_TO\_EMBEDDED,
- 2) The Wavecom library calls the appropriate AT function from the Wavecom Core Software,
- 3) The AT function processes the command,
- 4) The AT function sends the AT response to the Embedded Application,
- 5) This response is dispatched by the Wavecom library which calls the "wm\_apmAppliParser" function of the Embedded Application,
- 6) The "wm\_apmAppliParser" function processes the response (the AT response is a parameter of the function). The Message type is WM\_AT\_RESPONSE.

### Example: appli.c file of a Standalone Mode Embedded Application

```
/* *****  
/* Appli.c - Copyright Wavecom S.A. (c) 2003 */  
/* *****  
  
#include "wm_types.h"  
#include "wm_apm.h"  
  
#define TIMER 01  
  
/* *****  
/* Mandatory Functions */  
/* *****  
  
/* *****  
/* wm_apmAppliInit */  
/* Embedded Application initialisation */  
/* *****  
s32 wm_apmAppliInit ( wm_apmInitType_e InitType )  
{  
    wm_osDebugTrace(1, "Embedded: Appli Init" );  
    wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );  
    return OK;  
}
```

```

/*****
/*  wm_apmAppliParser          */
/*  Embedded Application message parser  */
/*****
s32 wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
    wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

    switch ( pMessage->MsgTyp )
    {
        case WM_OS_TIMER:
            wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
            if ( pMessage->Body.OSTimer.Ident == TIMER )
            {
                wm_atSendCommand ( 4, WM_AT_SEND_RSP_TO_EMBEDDED,
                                   "AT\r" );
                wm_osDebugTrace ( 1, "Send command \"AT\\r\" );
            }
            break;

        case WM_AT_RESPONSE:
            wm_osDebugTrace ( 1, "WM_AT_RESPONSE received" );
            if ( pMessage->Body.ATResponse.Type ==
                WM_AT_SEND_RSP_TO_EMBEDDED )
            {
                wm_osDebugTrace ( 1, "Response received:" );
                wm_osDebugTrace ( 1, pMessage->Body.ATResponse.StrData );
            }
            break;
    }

    return OK;
}

```

```

/*****
/*  Mandatory Variables  */
/*****

#define StackSize 1024
u32 Stack [ StackSize / 4 ];

// Tasks table
const wm_apmTask_t wm_apmTask [] =
{
    { StackSize, Stack, wm_apmAppliInit, wm_apmAppliParser },
    { 0,          NULL,  NULL,          NULL          },
    { 0,          NULL,  NULL,          NULL          },
};

```

Target Monitoring Tool traces with this example:

Trace	CUS	1	Embedded: Appli Init
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	Send command "AT\r"
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_RESPONSE received
Trace	CUS	1	Response received:
Trace	CUS	1	<CR><LF>OK<CR><LF>

### 4.3 Cooperative Mode

This mode corresponds to the interaction between an External Application and an Embedded Application.

Whenever the Embedded Application wants to filter or spy **the commands** sent by the External Application, it can use the **command pre-parsing** mechanism.

Three types of subscription are available. They define the level of information required by the Embedded Application:

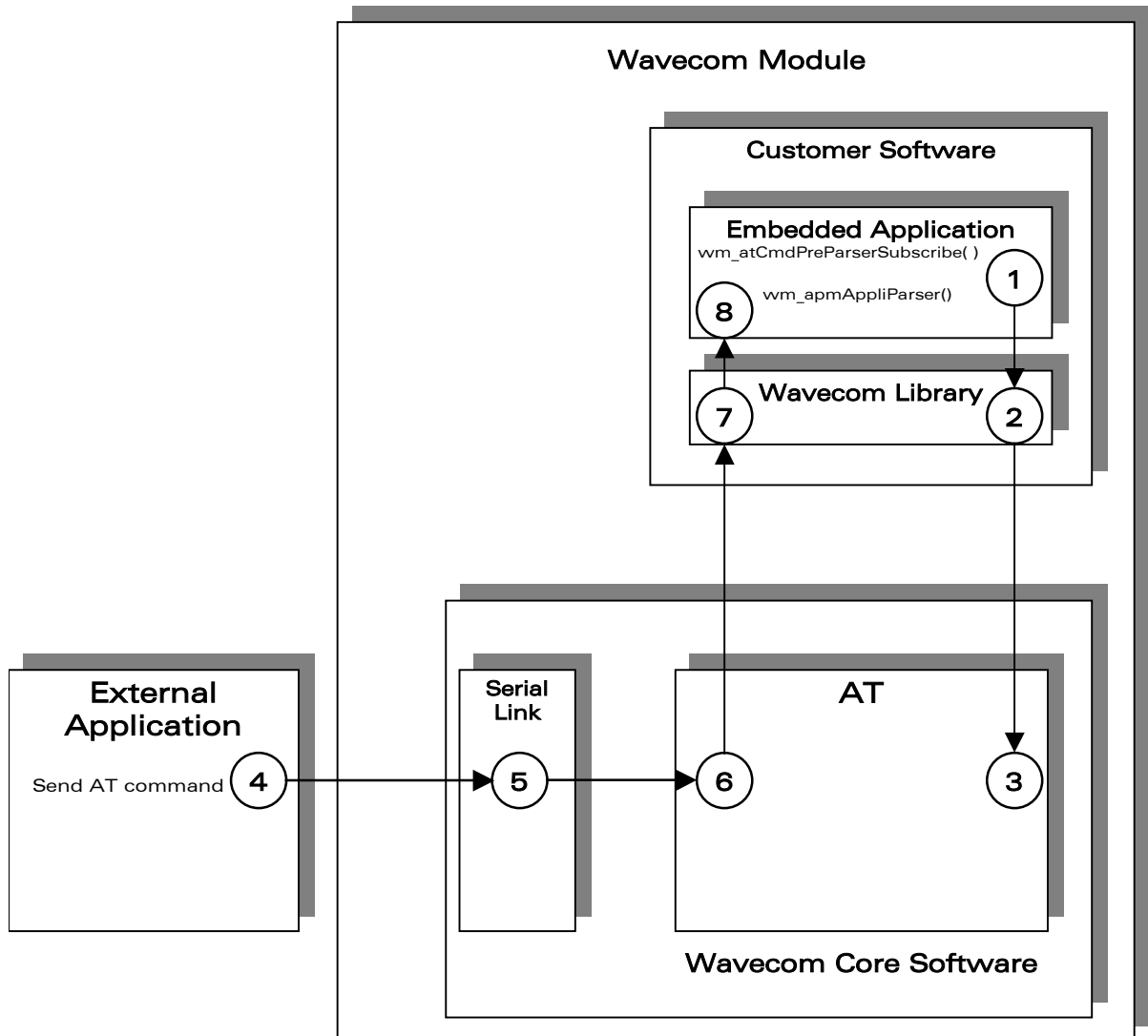
- ❑ The Embedded Application does not want to filter or spy the commands sent by the External Application: this is done using **WM\_AT\_CMD\_PRE\_WAVECOM\_TREATMENT**.
- ❑ The Embedded Application wants to filter the AT commands sent by the External Application: this is done using **WM\_AT\_CMD\_PRE\_EMBEDDED\_TREATMENT**.  
In this configuration, it is up to the Embedded Application to process or not the AT command and to send a response to the External Application.
- ❑ The Embedded Application wants only to spy the AT commands sent by the External Application: this is done using **WM\_AT\_CMD\_PRE\_BROADCAST**.

Whenever the Embedded Application wants to filter or spy the **responses** sent to the External Application, it can use the **response pre-parsing** mechanism.

Three types of subscription are available. They define the level of information required by the Embedded Application:

- ❑ The Embedded Application does not want to filter or spy the responses sent to the External Application: this is done using **WM\_AT\_RSP\_PRE\_WAVECOM\_TREATMENT**.
- ❑ The Embedded Application wants to filter the AT responses sent to the External Application: this is done using **WM\_AT\_RSP\_PRE\_EMBEDDED\_TREATMENT**.  
In this configuration, it is up to the Embedded Application to send a response to the External Application.
- ❑ The Embedded Application wants only to spy the AT responses sent to the External Application: this is done using **WM\_AT\_RSP\_PRE\_BROADCAST**.

**4.3.1 Command Pre-Parsing Subscription Mechanism:  
WM\_AT\_CMD\_PRE\_EMBEDDED\_TREATMENT**



**Figure 7: WM\_AT\_CMD\_PRE\_EMBEDDED\_TREATMENT**

The steps in a Pre-Parsing subscription are performed in the following sequence:

- 1) The Embedded Application subscribes to the command pre-parsing service, by calling the `wm_atCmdPreParserSubscribe()` function,
- 2) The Wavecom library calls the appropriate function from the Wavecom Core Software,
- 3) The AT function sets the subscription.

The steps in AT command processing are performed in the following sequence:

- 4) The External Application sends an AT command,
- 5) The serial link transmits the command to the AT processor function in the Wavecom Core Software,
- 6) The AT function does not process the command but transmits it to the Embedded Application,
- 7) The command is routed by the Wavecom library which calls the "wm\_apmAppliParser" function of the Embedded Application (the Message type is WM\_AT\_CMD\_PRE\_PARSER),
- 8) This function processes the command: the parameters of the function include the AT command and an indication that the command comes from an External Application.

**Example: appli.c file of a WM\_AT\_CMD\_PRE\_EMBEDDED\_TREATMENT Mode Embedded Application**

**Example: appli.c file of a WM\_AT\_CMD\_PRE\_EMBEDDED\_TREATMENT Mode Embedded Application**

```
/* **** */
/* Appli.c - Copyright Wavecom S.A. (c) 2003 */
/* **** */

#include "wm_types.h"
#include "wm_apm.h"

#define TIMER 01

/* **** */
/* Mandatory Functions */
/* **** */

/* **** */
/* wm_apmAppliInit */
/* Embedded Application initialisation */
/* **** */
s32 wm_apmAppliInit ( wm_apmInitType_e InitType )
{
    wm_osDebugTrace(1, "Embedded: Appli Init" );
    wm_atCmdPreParserSubscribe (
        WM_AT_CMD_PRE_EMBEDDED_TREATMENT );
    wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );
    return OK;
}
```

```

/*****/
/*  wm_apmAppliParser                               */
/* Embedded Application message parser */
/*****/
s32 wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
    wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

    switch ( pMessage->MsgTyp )
    {
        case WM_OS_TIMER:
            wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
            break;

        case WM_AT_CMD_PRE_PARSER:
            wm_osDebugTrace ( 1, "WM_AT_CMD_PRE_PARSER received" );
            if ( pMessage->Body.ATCmdPreParser.Type ==
                WM_AT_CMD_PRE_EMBEDDED_TREATMENT )
            {
                wm_osDebugTrace ( 1, "command received:" );
                wm_osDebugTrace ( 1, pMessage->Body.ATCmdPreParser.StrData
);

                if ( !wm_strncmp ( pMessage->Body.ATCmdPreParser.StrData,
                    "AT-W", 4 ) )
                {
                    /* filter Specific embedded application command */
                    wm_osDebugTrace ( 1, "Specific embedded application
command" );

                    /* send response to external application */
                    wm_atSendRspExternalApp ( 10, "\r\n->WOK\r\n" );
                }
                else
                {
                    /* command must be treated by AT Software */
                    wm_osDebugTrace ( 1, "Wavecom Core Software command" );
                    wm_atSendCommand (
                        pMessage->Body.ATCmdPreParser.StrLength,
                        WM_AT_SEND_RSP_TO_EXTERNAL,
                        pMessage->Body.ATCmdPreParser.StrData );
                }
            }
            break;
    }

    return OK;
}

```

```
/* *****  
/* Mandatory Variables */  
/* *****  
  
#define StackSize 1024  
u32 Stack [ StackSize / 4 ];  
  
// Tasks table  
const wm_apmTask_t wm_apmTask [] =  
{  
  { StackSize, Stack, wm_apmAppliInit, wm_apmAppliParser },  
  { 0,          NULL,  NULL,          NULL          },  
  { 0,          NULL,  NULL,          NULL          }  
};
```

An AT command log for the external application with this example:

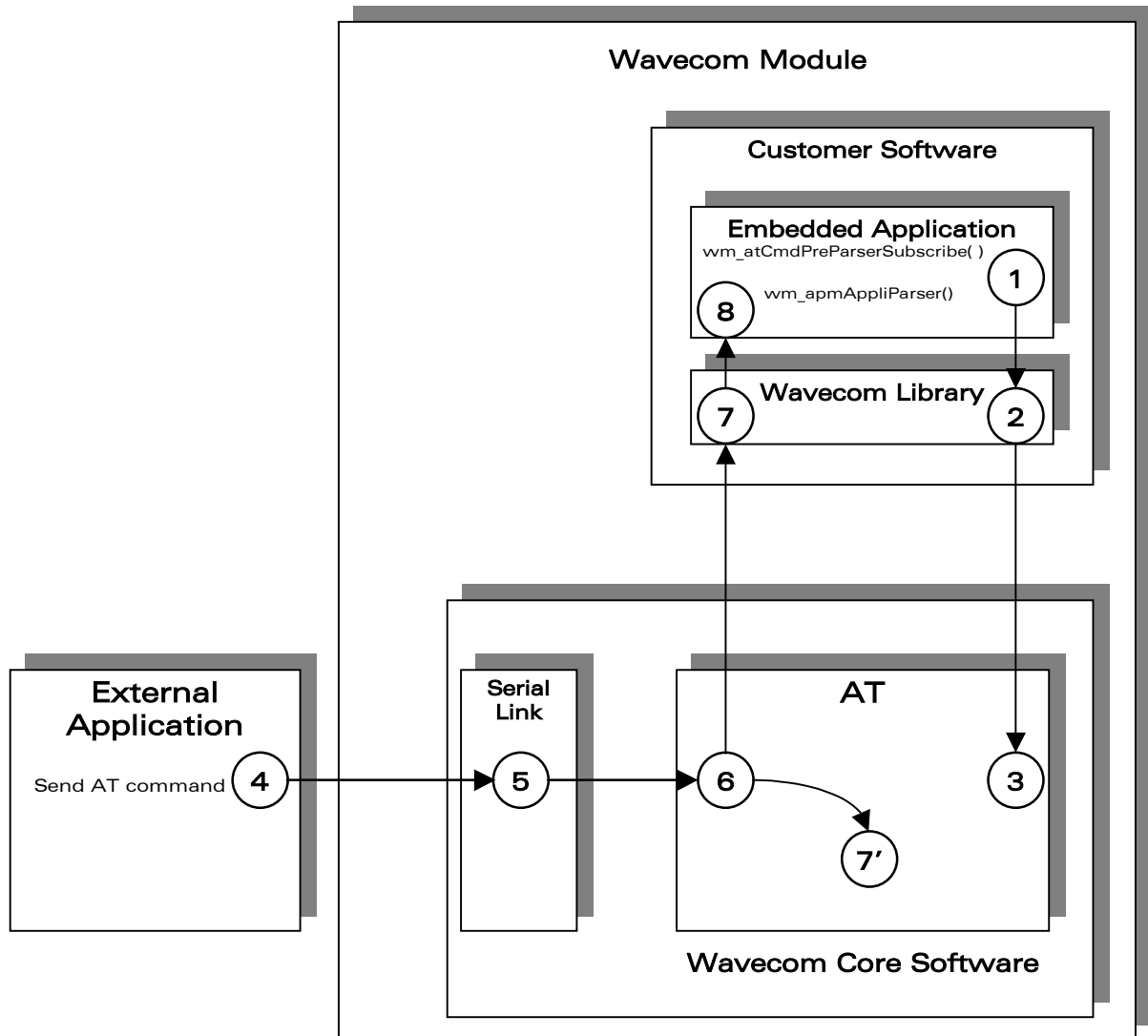
```
AT  
OK  
AT-W  
->WOK
```

Target Monitoring Tool traces with this example:

Trace	CUS	1	Embedded: Appli Init
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_CMD_PRE_PARSER received
Trace	CUS	1	command received:
Trace	CUS	1	AT<CR>
Trace	CUS	1	Wavecom Core Software command
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_CMD_PRE_PARSER received
Trace	CUS	1	command received:
Trace	CUS	1	AT-W<CR>
Trace	CUS	1	Specific embedded application command



**4.3.2 Command Pre-Parsing Subscription Process:  
WM\_AT\_CMD\_PRE\_BROADCAST**



**Figure 8: WM\_AT\_CMD\_PRE\_BROADCAST**

The steps in a Pre-Parsing subscription are performed in the following sequence:

- 1) The Embedded Application subscribes to the command pre-parsing service, by calling the `wm_atCmdPreParserSubscribe()` function,
- 2) The Wavecom library calls the appropriate function in the Wavecom Core Software, and
- 3) The AT function sets the subscription.

The steps in AT command processing are performed in the following sequence:

- 4) The External Application sends an AT command,
- 5) The serial link transmits the command to the AT function of the Wavecom Core Software,
- 6) This AT function checks the subscription status of the "external" AT command,
- 7) This external AT command is dispatched by the Wavecom library which calls the "wm\_apmAppliParser" function of the Embedded Application,
- 7') Meanwhile, the AT function processes the command,
- 8) The "wm\_apmAppliParser" function spies the command: the parameters include the AT command and the indication of whether or not the command is a copy (the Message type is WM\_AT\_CMD\_PRE\_PARSER).

**Example: appli.c file of a WM\_AT\_CMD\_PRE\_BROADCAST Mode Embedded Application**

**Example: appli.c file of a WM\_AT\_CMD\_PRE\_BROADCAST Mode Embedded Application**

```
/* *****  
/* Appli.c - Copyright Wavecom S.A. (c) 2001 */  
/* *****  
  
#include "wm_types.h"  
#include "wm_apm.h"  
  
#define TIMER 01  
  
/* *****  
/* Mandatory Functions */  
/* *****  
  
/* *****  
/* wm_apmAppliInit */  
/* Embedded Application initialisation */  
/* *****  
s32 wm_apmAppliInit ( wm_apmInitType_e InitType )  
{  
    wm_osDebugTrace(1, "Embedded: Appli Init" );  
    wm_atCmdPreParserSubscribe ( WM_AT_CMD_PRE_BROADCAST );  
    wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );  
    return OK;  
}
```

```

/*****
/*  wm_apmAppliParser                               */
/*  Embedded Application message parser */
/*****
s32 wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
    wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

    switch ( pMessage->MsgTyp )
    {
        case WM_OS_TIMER:
            wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
            break;

        case WM_AT_CMD_PRE_PARSER:
            wm_osDebugTrace ( 1, "WM_AT_CMD_PRE_PARSER received" );
            if ( pMessage->Body.ATCmdPreParser.Type ==
                WM_AT_CMD_PRE_BROADCAST )
            {
                /* spy command sent by external application */
                wm_osDebugTrace ( 1, "command received from external
application" );
                wm_osDebugTrace ( 1, pMessage->Body.ATCmdPreParser.StrData
);
            }
            break;
    }

    return OK;
}

```

```

/*****
/*  Mandatory Variables */
/*****

#define StackSize 1024
u32 Stack [ StackSize / 4 ];

// Tasks table
const wm_apmTask_t wm_apmTask [] =
{
    { StackSize, Stack, wm_apmAppliInit, wm_apmAppliParser },
    { 0,          NULL,  NULL,          NULL          },
    { 0,          NULL,  NULL,          NULL          },
};

```

AT command log for the external application with this example:

```

at
OK

```

Target Monitoring Tool traces with this example:

Trace	CUS	1	Embedded: Appli Init
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_CMD_PRE_PARSER received
Trace	CUS	1	command received from external application
Trace	CUS	1	at<CR>

**4.3.3 Response      Pre-Parsing      Subscription      Process:**  
**WM\_AT\_RSP\_PRE\_EMBEDDED\_TREATMENT**

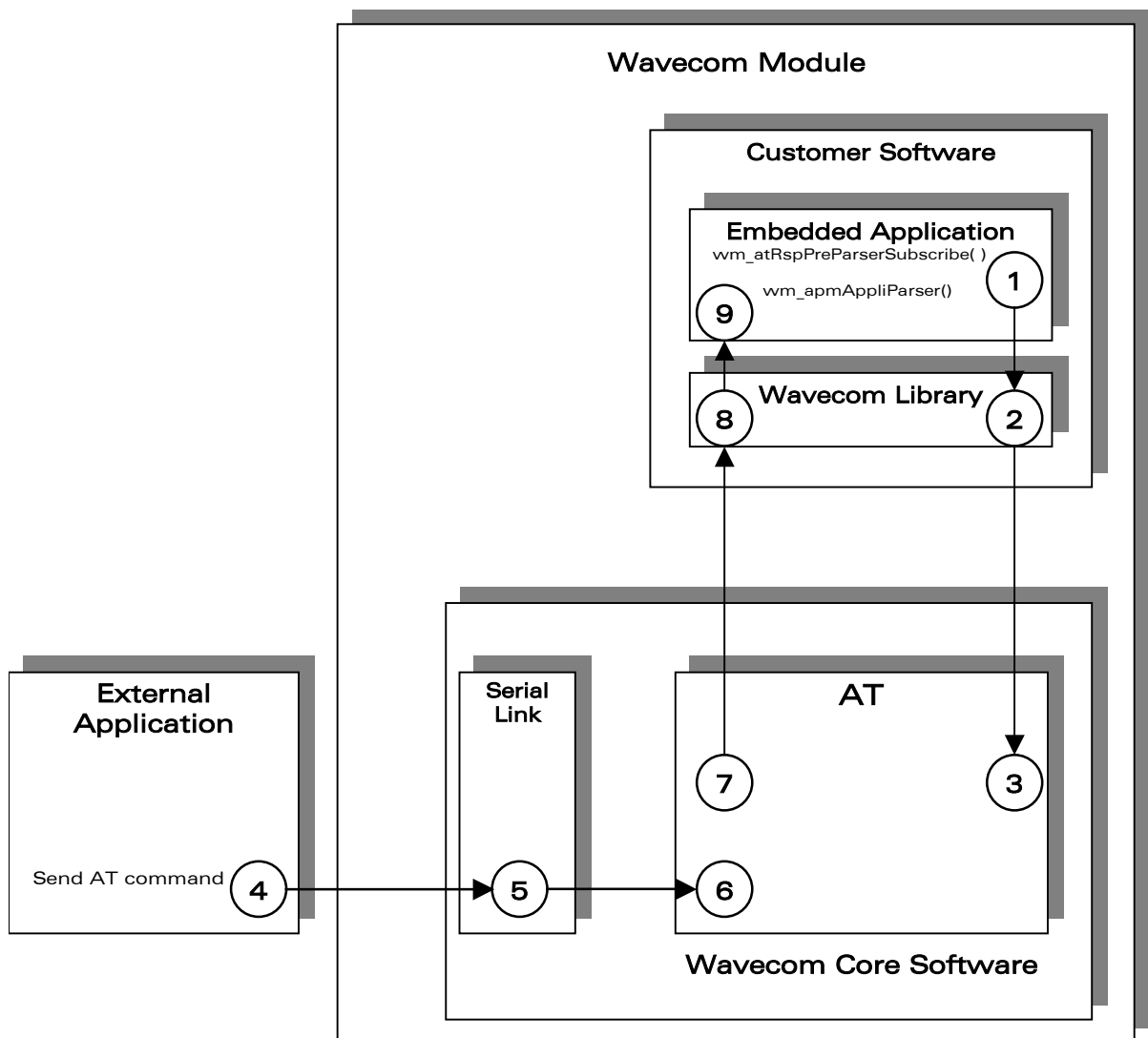


Figure 9: WM\_AT\_RSP\_PRE\_EMBEDDED\_TREATMENT

The steps in a Pre-Parsing subscription are performed in the following sequence:

- 1) The Embedded Application subscribes to the response pre-parsing facility, by calling the `wm_atRspPreParserSubscribe()` function,
- 2) The Wavecom library calls the appropriate function from the Wavecom Core Software, and
- 3) The AT function sets the subscription.

The steps in AT command processing are performed in the following sequence:

- 4) The External Application sends an AT command,
- 5) The serial link transmits the command to the AT function of the Wavecom Core Software,
- 6) This configuration does not rely on command pre-parsing. The AT function processes the command,
- 7) The AT function checks the subscription status of the response and does not send the response to the External Application. Instead, it sends the response to the Embedded Application,
- 8) The response is dispatched by the Wavecom library which calls the "wm\_apmAppliParser" function of the Embedded Application (the Message type is `WM_AT_RSP_PRE_PARSER`),
- 9) This function processes the response (the parameters of the function include an indication of the response filtering).

#### Example: appli.c file of a WM\_AT\_RSP\_PRE\_EMBEDDED\_TREATMENT Mode Embedded Application

```
/* *****  
/* Appli.c - Copyright Wavecom S.A. (c) 2001 */  
/* *****  
  
#include "wm_types.h"  
#include "wm_apm.h"  
#define TIMER 01  
  
/* *****  
/* Mandatory Functions */  
/* *****  
/* *****  
/* wm_apmAppliInit */  
/* Embedded Application initialisation */  
/* *****  
s32 wm_apmAppliInit ( wm_apmInitType_e InitType )  
{  
    wm_osDebugTrace(1, "Embedded: Appli Init" );  
    wm_atRspPreParserSubscribe ( WM_AT_RSP_PRE_EMBEDDED_TREATMENT );  
    wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );  
    return OK;  
}
```

```
/* ***** */
/* wm_apmAppliParser */
/* Embedded Application message parser */
/* ***** */
s32 wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
    wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

    switch ( pMessage->MsgTyp )
    {
        case WM_OS_TIMER:
            wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
            break;

        case WM_AT_RSP_PRE_PARSER:
            wm_osDebugTrace ( 1, "WM_AT_RSP_PRE_PARSER received" );
            wm_osDebugTrace ( 1, pMessage->Body.ATRspPreParser.StrData );

            if ( pMessage->Body.ATRspPreParser.Type ==
                WM_AT_RSP_PRE_EMBEDDED_TREATMENT )
            {
                if ( !wm_strncmp ( "\r\nOK\r\n",
                    pMessage->Body.ATRspPreParser.StrData, 6 ) )
                {
                    wm_osDebugTrace ( 1, "OK response modified for external
                        application" );
                    wm_atSendRspExternalApp ( 10, "\r\n->WOK\r\n" );
                }
                else
                {
                    wm_osDebugTrace ( 1, "no modified response" );
                    wm_atSendRspExternalApp (
                        pMessage->Body.ATRspPreParser.StrLength,
                        pMessage->Body.ATRspPreParser.StrData );
                }
            }
            break;
    }

    return OK;
}
```

```

/*****/
/* Mandatory Variables */
/*****/

#define StackSize 1024
u32 Stack [ StackSize / 4 ];

// Tasks table
const wm_apmTask_t wm_apmTask [] =
{
{ StackSize, Stack, wm_apmAppliInit, wm_apmAppliParser },
{ 0,          NULL,  NULL,          NULL          },
{ 0,          NULL,  NULL,          NULL          }
};
    
```

AT commands log for the external application with this example:

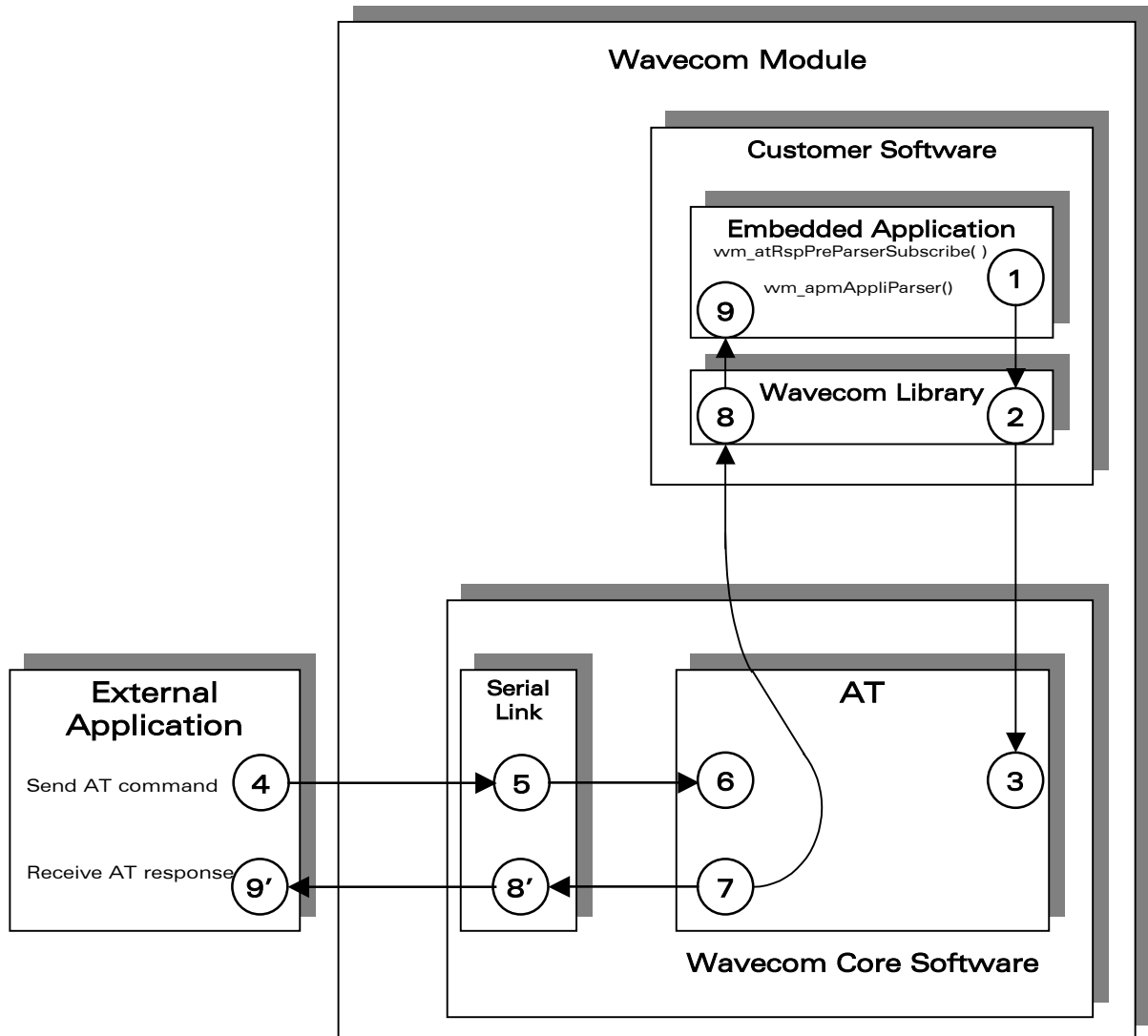
```

at
->WOK
at+wopen?
+WOPEN: 1
->WOK
    
```

Target Monitoring Tool traces with this example:

Trace	CUS	1	Embedded: Appli Init
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_RSP_PRE_PARSER received
Trace	CUS	1	<CR><LF>OK<CR><LF>
Trace	CUS	1	OK response modified for external application
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_RSP_PRE_PARSER received
Trace	CUS	1	<CR><LF>+WOPEN: 1<CR><LF>
Trace	CUS	1	no modified response
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_RSP_PRE_PARSER received
Trace	CUS	1	<CR><LF>OK<CR><LF>
Trace	CUS	1	OK response modified for external application

**4.3.4 Response Pre-Parsing Subscription Process:**  
**WM\_AT\_RSP\_PRE\_BROADCAST**



**Figure 10: WM\_AT\_RSP\_PRE\_BROADCAST**

The steps in a Pre-Parsing subscription are performed in the following sequence:

- 1) The Embedded Application subscribes to the response pre-parsing facility, by calling the `wm_atRspPreParserSubscribe()` function,
- 2) The Wavecom library calls the appropriate function in the Wavecom Core Software, and
- 3) The AT function sets the subscription.



The steps in AT command processing are performed in the following sequence:

- 4) The External Application sends an AT command,
- 5) The serial link transmits the command to the AT function of the Wavecom Core Software,
- 6) This configuration does not rely on command pre-parsing. The AT function processes the command,
- 7) The AT function checks the subscription status of the response and sends it to both the External Application and the Embedded Application,
- 8) The response is dispatched by the Wavecom library, which calls the "wm\_apmAppliParser" function of the Embedded Application (the Message type is WM\_AT\_RSP\_PRE\_PARSER),
- 9) This function processes the response (the parameters of the function include a broadcast response indication),
- 8') This response is sent through the serial link,
- 9') The External Application receives the response.

**Example: appli.c file of a WM\_AT\_RSP\_PRE\_BROADCAST Mode Embedded Application**

```
/*
*****
/* Appli.c - Copyright Wavecom S.A. (c) 2001 */
*****
*/

#include "wm_types.h"
#include "wm_apm.h"
#define TIMER 01

/*
*****
/* Mandatory Functions */
*****
*/

/*
*****
/* wm_apmAppliInit */
/* Embedded Application initialisation */
*****
*/
s32 wm_apmAppliInit ( wm_apmInitType_e InitType )
{
    wm_osDebugTrace(1, "Embedded: Appli Init" );
    wm_atRspPreParserSubscribe ( WM_AT_RSP_PRE_BROADCAST );
    wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );
    return OK;
}
```

```

/*****/
/*  wm_apmAppliParser                               */
/*  Embedded Application message parser */
/*****/
s32 wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
    wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

    switch ( pMessage->MsgTyp )
    {
        case WM_OS_TIMER:
            wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
            break;

        case WM_AT_RSP_PRE_PARSER:
            wm_osDebugTrace ( 1, "WM_AT_RSP_PRE_PARSER received" );

            if ( pMessage->Body.ATRspPreParser.Type ==
                WM_AT_RSP_PRE_BROADCAST )
            {
                /* spy response sent to external application */
                wm_osDebugTrace ( 1, "response sent to external
application" );
                wm_osDebugTrace ( 1, pMessage->Body.ATRspPreParser.StrData );
            }
            break;
    }

    return OK;
}

```

```

/*****/
/*  Mandatory Variables */
/*****/

#define StackSize 1024
u32 Stack [ StackSize / 4 ];

// Tasks table
const wm_apmTask_t wm_apmTask [] =
{
    { StackSize, Stack, wm_apmAppliInit, wm_apmAppliParser },
    { 0,          NULL,  NULL,          NULL },
    { 0,          NULL,  NULL,          NULL }
};

```

AT command log for the external application with this example:

```

at
OK

```

Target Monitoring Tool traces with this example:

Trace	CUS	1	Embedded: Appli Init
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_AT_RSP_PRE_PARSER received
Trace	CUS	1	response sent to external application
Trace	CUS	1	<CR><LF>OK<CR><LF>

#### 4.3.5 Example: Embedded Application Using the Different Functioning Modes

```

/*****
/*  Appli.c  -  Copyright Wavecom S.A. (c) 2001  */
/*****

#include "wm_types.h"
#include "wm_apm.h"

#define TIMER 01

typedef enum
{
    STANDALONE,
    CMD_PREPARSING_EMBEDDED,
    CMD_PREPARSING_BROADCAST,
    RSP_PREPARSING_EMBEDDED,
    RSP_PREPARSING_BROADCAST,
} wm_AtMode_e;

/*****
/*  Global Variables  */
/*****

wm_AtMode_e AtMode = STANDALONE;
    
```

```
/* Global Function */
void AtAutomate(state)
{
    switch(state)
    {
        case STANDALONE:
            wm_osDebugTrace(1, "STANDALONE" );
            wm_atCmdPreParserSubscribe(WM_AT_CMD_PRE_WAVECOM_TREATMENT);
            wm_atRspPreParserSubscribe(WM_AT_RSP_PRE_WAVECOM_TREATMENT);
            wm_atSendRspExternalApp(16,"STANDALONE mode");
            wm_atSendRspExternalApp(18,"send an at command");
            break;

        case CMD_PREPARSING_EMBEDDED:
            wm_osDebugTrace(1, "CMD_PREPARSING_EMBEDDED" );
            wm_atCmdPreParserSubscribe(WM_AT_CMD_PRE_EMBEDDED_TREATMENT);
            wm_atRspPreParserSubscribe(WM_AT_RSP_PRE_WAVECOM_TREATMENT);
            wm_atSendRspExternalApp(29,"CMD_PREPARSING_EMBEDDED mode");
            wm_atSendRspExternalApp(18,"send an at command");
            break;

        case CMD_PREPARSING_BROADCAST:
            wm_osDebugTrace(1, "CMD_PREPARSING_BROADCAST" );
            wm_atCmdPreParserSubscribe(WM_AT_CMD_PRE_BROADCAST);
            wm_atRspPreParserSubscribe(WM_AT_RSP_PRE_WAVECOM_TREATMENT);
            wm_atSendRspExternalApp(30,"CMD_PREPARSING_BROADCAST mode");
            wm_atSendRspExternalApp(18,"send an at command");
            break;

        case RSP_PREPARSING_EMBEDDED:
            wm_osDebugTrace(1, "RSP_PREPARSING_EMBEDDED" );
            wm_atCmdPreParserSubscribe(WM_AT_CMD_PRE_WAVECOM_TREATMENT);
            wm_atRspPreParserSubscribe(WM_AT_RSP_PRE_EMBEDDED_TREATMENT);
            wm_atSendRspExternalApp(29,"RSP_PREPARSING_EMBEDDED mode");
            wm_atSendRspExternalApp(18,"send an at command");
            break;

        case RSP_PREPARSING_BROADCAST:
            wm_osDebugTrace(1, "RSP_PREPARSING_BROADCAST" );
            wm_atCmdPreParserSubscribe(WM_AT_CMD_PRE_WAVECOM_TREATMENT);
            wm_atRspPreParserSubscribe(WM_AT_RSP_PRE_BROADCAST );
            wm_atSendRspExternalApp(30,"RSP_PREPARSING_BROADCAST mode");
            wm_atSendRspExternalApp(18,"send an at command");
            break;

        default:
            wm_osDebugTrace(1, "mode unexpected" );
            break;
    }
}
```

```

/*****/
/* Mandatory Functions */
/*****/

/*****/
/* wm_apmAppliInit */
/* Embedded Application initialisation */
/*****/
s32 wm_apmAppliInit ( wm_apmInitType_e InitType )
{
    wm_osDebugTrace(1, "Embedded: Appli Init" );
    wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );
    return OK;
}

/*****/
/* wm_apmAppliParser */
/* Embedded Application message parser */
/*****/
s32 wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
    wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

    switch ( pMessage->MsgTyp )
    {
        case WM_OS_TIMER:
            wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
            AtAutomate(AtMode);
            if (AtMode!=RSP_PREPARSING_BROADCAST)
            {
                AtMode++;
                wm_osStartTimer (TIMER, FALSE, WM_S_TO_TICK(10));
            }
            break;

        case WM_AT_RESPONSE:
            wm_atSendRspExternalApp( 33, "message WM_AT_RESPONSE
                                     received:");
            wm_strncpy(strReceived, pMessage->Body.ATResponse.StrData,
                       pMessage->Body.ATResponse.StrLength);
            strReceived[pMessage->Body.ATResponse.StrLength] = '\0';
            wm_atSendRspExternalApp( pMessage->Body.ATResponse.StrLength+1,
                                     strReceived );
            break;

        case WM_AT_CMD_PRE_PARSER:
            wm_atSendRspExternalApp(39, "message WM_AT_CMD_PRE_PARSER
                                     received:");
            wm_strncpy(strReceived, pMessage->Body.ATCmdPreParser.StrData,
                       pMessage->Body.ATCmdPreParser.StrLength);
            strReceived[pMessage->Body.ATCmdPreParser.StrLength] = '\0';
            wm_atSendRspExternalApp(pMessage->Body.ATResponse.StrLength+1,
                                     strReceived );
            break;
    }
}

```

```
    case WM_AT_RSP_PRE_PARSER:
        wm_atSendRspExternalApp(39, "message WM_AT_RSP_PRE_PARSER
                                   received:");
        wm_strncpy(strReceived, pMessage->Body.ATRspPreParser.StrData,
                  pMessage->Body.ATRspPreParser.StrLength);
        strReceived[pMessage->Body.ATRspPreParser.StrLength] = '\0';
        wm_atSendRspExternalApp(pMessage->Body.ATResponse.StrLength +
                               1, strReceived );
        break;
    }

    return TRUE;
}

/*****
/*  Mandatory Variables  */
*****/

#define StackSize 1024
u32 Stack [ StackSize / 4 ];

// Tasks table
const wm_apmTask_t wm_apmTask [] =
{
{ StackSize, Stack, wm_apmAppliInit, wm_apmAppliParser },
{ 0,          NULL,  NULL,          NULL          },
{ 0,          NULL,  NULL,          NULL          }
};
```

**AT command log for the external application with this example:**

```

STANDALONE mode
at          no interaction between external
OK       and embedded application

CMD_PREPARSING_EMBEDDED mode
send an at command
at          command sent to embedded application
message WM_AT_CMD_PRE_PARSER received:
at          and not to Wavecom AT Software

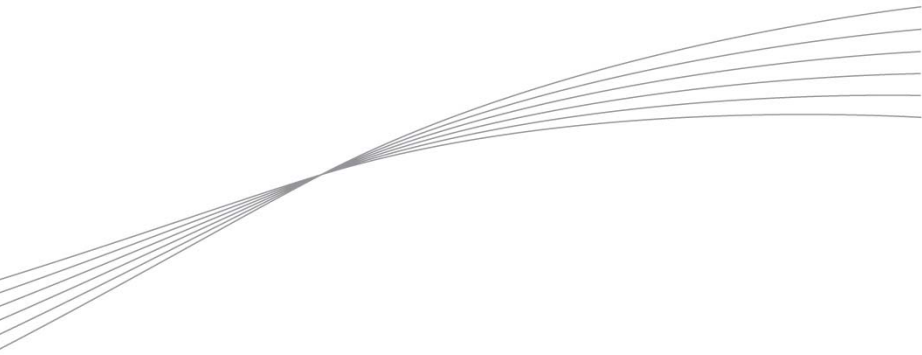
CMD_PREPARSING_BROADCAST mode
send an at command
at          command sent to both
OK       response of Wavecom AT Software
message WM_AT_CMD_PRE_PARSER received:
at          command received by embedded application

RSP_PREPARSING_EMBEDDED mode
send an at command
at          command sent to Wavecom AT Software
message WM_AT_RSP_PRE_PARSER received:
OK       response sent to embedded application

RSP_PREPARSING_BROADCAST mode
send an at command
at          command sent to Wavecom AT Software
OK       response sent to external application
message WM_AT_RSP_PRE_PARSER received:
OK       response sent to embedded application
    
```

**Target Monitoring Tool traces with this example:**

Trace	CUS	1	Embedded: Appli Init
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	STANDALONE
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	CMD_PREPARSING_EMBEDDED
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	CMD_PREPARSING_BROADCAST
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	RSP_PREPARSING_EMBEDDED
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	Embedded: Appli Parser
Trace	CUS	1	WM_OS_TIMER received
Trace	CUS	1	RSP_PREPARSING_BROADCAST
Trace	CUS	1	Embedded: Appli Parser



**wavecom**   
*Make it wireless*

---

WAVECOM S.A. - 3 esplanade du Foncet - 92442 Issy-les-Moulineaux Cedex - France - Tel: +33(0)1 46 29 08 00 - Fax: +33(0)1 46 29 08 08  
Wavecom, Inc. - 4810 Eastgate Mall - Second Floor - San Diego, CA 92121 - USA - Tel: +1 858 362 0101 - Fax: +1 858 558 5485  
WAVECOM Asia Pacific Ltd. - 4/F, Shui On Centre - 6/8 Harbour Road - Hong Kong - Tel: +852 2824 0254 - Fax: +852 2824 025

[www.wavecom.com](http://www.wavecom.com)