

Basic Development Guide for Open AT® V4.01

Revision: 003
Date: August 2006

OPEN AT

wavecom
Make it wireless

Operating Systems | Integrated Development Environments | Plug-Ins | Wireless CPUs | Services

Basic Development Guide for Open AT® V4.01

Reference : **WM_DEV_OAT_UGD_005**
Revision : **003**
Date : **August 17, 2006**

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Trademarks

®, WAVECOM®, WISMO®, Open AT® 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Overview

This User's Guide describes the Open AT® facility and provides guidelines for developing an Embedded Application. It applies to V4.01 and higher (until next version of this document).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Document History

Rev	Date	History	
001	January 13, 2006	Creation	
002	March 2006	Update	
003	August 17, 2006	Update for Open AT® V4.01	

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Table of Contents

1 Introduction 10

1.1 References 10
1.2 Glossary 10
1.3 Abbreviations 11
1.4 Open AT® general introduction 11

2 API 12

2.1 Data Types 12
2.2 Mandatory Functions 12
2.2.1 Required Header 12
2.2.2 Task identifiers 12
2.2.3 Task table 12
2.2.4 Stack Initialization 13
2.2.5 The Init Functions 14
2.2.6 The Parser Functions 15
2.3 AT Command API 22
2.3.1 Required Header 22
2.3.2 The wm_atSendCommand Function 22
2.3.3 The wm_atSendCommandExt Function 22
2.3.4 The wm_atUnsolicitedSubscription Function 26
2.3.5 The wm_atIntermediateSubscription Function 28
2.3.6 The wm_atCmdPreParserSubscribe Function 30
2.3.7 The wm_atRspPreParserSubscribe Function 32
2.3.8 The wm_atSendRspExternalApp Function 34
2.3.9 The wm_atSendRspExternalAppExt Function 35
2.3.10 The wm_atSendUnsolicitedExternalApp Function 36
2.3.11 The wm_atSendUnsolicitedExternalAppExt Function 36
2.3.12 The wm_atSendIntermediateExternalApp Function 37
2.3.13 The wm_atSendIntermediateExternalAppExt Function 37
2.4 Debug API 38
2.4.1 Required Header 38
2.4.2 The wm_osDebugTrace Function 38
2.4.3 The wm_osDebugFatalError Function 40
2.4.4 The wm_osDebugEraseAllBacktraces Function 40
2.4.5 The wm_osDebugInitBacktracesAnalysis Function 40
2.4.6 The wm_osDebugRetrieveBacktrace Function 41
2.5 OS API 42
2.5.1 Required Header 42
2.5.2 The wm_osStartTimer Function 42
2.5.3 The wm_osStopTimer Function 43
2.5.4 The wm_osStartTickTimer Function 43
2.5.5 The wm_osStopTickTimer Function 44
2.5.6 Important Note on Data Flash Management 45

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.7 The wm_osWriteFlashData Function 45
2.5.8 The wm_osReadFlashData Function 46
2.5.9 The wm_osGetLenFlashData Function 46
2.5.10 The wm_osDeleteFlashData Function 47
2.5.11 The wm_osGetAllowedMemoryFlashData Function 47
2.5.12 The wm_osGetFreeMemoryFlashData Function 47
2.5.13 The wm_osGetUsedMemoryFlashData Function 48
2.5.14 The wm_osDeleteAllFlashData Function 48
2.5.15 The wm_osDeleteRangeFlashData Function 48
2.5.16 The wm_osGetHeapMemory Function 49
2.5.17 The wm_osReleaseHeapMemory Function 49
2.5.18 The wm_osGetRamInfo Function 49
2.5.19 The wm_osSuspend function 50
2.5.20 The wm_osGetTask Function 50
2.5.21 The wm_osSendMsg Function 50
2.5.22 Example: Managing Data Flash Objects 52
2.5.23 Example: RAM management 52

2.6 Flow Control Manager API 53
2.6.1 Required Header 53
2.6.2 The wm_fcmFlow_e type 53
2.6.3 The wm_fcmIsAvailable Function 53
2.6.4 The wm_fcmOpen Function 54
2.6.5 The wm_fcmClose Function 55
2.6.6 The wm_fcmSubmitData Function 56
2.6.7 Receive Data Blocks 58
2.6.8 The wm_fcmCreditToRelease Function 59
2.6.9 The wm_fcmQuery Function 59

2.7 Input Output API 61
2.7.1 Required Header 61
2.7.2 AT/FCM Ports related functions 61
2.7.3 GPIO API 65

2.8 GPRS API 75
2.8.1 GPRS Overview 75
2.8.2 The wm_gprsAuthentification function 77
2.8.3 The wm_gprsIPCPInformations function 78
2.8.4 The wm_gprsOpen function 79
2.8.5 The wm_gprsClose function 79

2.9 BUS API 80
2.9.1 Required Header 80
2.9.2 Returned values definition 80
2.9.3 The wm_busOpen Function 81
2.9.4 The wm_busClose Function 85
2.9.5 The wm_busWrite Function 85
2.9.6 The wm_busRead Function 86

2.10 Lists management API 89
2.10.1 Required Header 89
2.10.2 Types definition 89
2.10.3 The wm_lstCreate Function 90
2.10.4 The wm_lstDestroy Function 90
2.10.5 The wm_lstClear Function 91
2.10.6 The wm_lstGetCount Function 91
2.10.7 The wm_lstAddItem Function 92
2.10.8 The wm_lstInsertItem Function 92

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.9 The wm_lstGetItem Function 93
2.10.10 The wm_lstDeleteItem Function....93
2.10.11 The wm_lstFindItem Function....94
2.10.12 The wm_lstFindAllItem Function....94
2.10.13 The wm_lstFindNextItem Function 95
2.10.14 The wm_lstResetItem Function....95

2.11 Sound API 96
2.11.1 Required header 96
2.11.2 The wm_sndTonePlay Function 96
2.11.3 The wm_sndTonePlayExt Function 97
2.11.4 The wm_sndToneStop Function 99
2.11.5 The wm_sndDtmfPlay Function 100
2.11.6 The wm_sndDtmfStop Function 100
2.11.7 The wm_sndMelodyPlay Function 101
2.11.8 The wm_sndMelodyStop Function 102

2.12 Standard Library....104
2.12.1 Required Header 104
2.12.2 Standard C function set 104
2.12.3 String processing function set 105

2.13 Application & Data storage API 106
2.13.1 Required Header 106
2.13.2 Returned values definition 106
2.13.3 The wm_adAllocate Function....106
2.13.4 The wm_adRetrieve Function....107
2.13.5 The wm_adFindInit Function 107
2.13.6 The wm_adFindNext Function 108
2.13.7 The wm_adWrite Function....108
2.13.8 The wm_adFinalise Function 109
2.13.9 The wm_adResume Function....109
2.13.10 The wm_adInfo Function 110
2.13.11 The wm_adDelete Function 110
2.13.12 The wm_adStats Function 111
2.13.13 The wm_adSpaceState Function....111
2.13.14 The wm_adFormat Function 112
2.13.15 The wm_adRecompactInit Function....112
2.13.16 The wm_adRecompact Function....113
2.13.17 The wm_adInstall Function....113

2.14 RTC API 114
2.14.1 Required Header 114
2.14.2 RTC related types 114
2.14.3 The wm_rtcGetTime Function....114
2.14.4 The wm_rtcConvertTime function....115

3 FUNCTIONING.... 116

3.1 Standalone External Application....116

3.2 Embedded Application in Standalone Mode 118

3.3 Cooperative Mode 121
3.3.1 Command Pre-Parsing Subscription Mechanism: WM_AT_CMD_PRE_EMBEDDED_TREATMENT 122
3.3.2 Command Pre-Parsing Subscription Process: WM_AT_CMD_PRE_BROADCAST 126

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

3.3.3 Response Pre-Parsing Subscription Process: WM_AT_RSP_PRE_EMBEDDED_TREATMENT.....129
3.3.4 Response Pre-Parsing Subscription Process: WM_AT_RSP_PRE_BROADCAST.....133
3.3.5 Example: Embedded Application Using the Different Functioning Modes 136

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

List of Figures

Figure 1: Standalone external application function 116
Figure 2: Embedded Application in standalone mode function 118
Figure 3: WM_AT_CMD_PRE_EMBEDDED_TREATMENT 122
Figure 4: WM_AT_CMD_PRE_BROADCAST 126
Figure 5: WM_AT_RSP_PRE_EMBEDDED_TREATMENT 129
Figure 6: WM_AT_RSP_PRE_BROADCAST 133

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

1 Introduction

1.1 References

I. Tools Manual (ref WM_DEV_OAT_UGD_006 revision 003)
II. AT Command Interface Guide (for AT OS6.60: ref WM_DEV_OAT_UGD_003 revision 003)
III. Open AT® ADL User guide (ref WM_DEV_OAT_UGD_007 revision 003)

1.2 Glossary

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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

1.4 Open AT® general introduction

For more information about Open AT®, please refer to the ADL User Guide.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2 API

2.1 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.

2.2 Mandatory Functions

The API described below includes a set of mandatory functions and constants that Embedded software has to supply.
This API is located in the wm_apm.h file.

2.2.1 Required Header

An Open AT® application has to include the wm_apm.h header file.
This file includes all other APIs' header files.

2.2.2 Task identifiers

The 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 OS
} wm_osTask_e;
```

2.2.3 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)   */
    s32 (*Init) ( wm_apmInitType_e );   /* Initialisation function */
    s32 (*Parser) ( wm_apmMsg_t * );    /* Parser function         */
} wm_apmTask_t;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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

```
const wm_apmTask_t wm_apmTask [] =
{
     { StackSize1, Init1, Parser1 },
     { StackSize2, Init2, Parser2 },
     { StackSize3, Init3, Parser3 },
     { 0, NULL, NULL }
};
```

The last line of the **wm_apmTask** table has to be filled with 0 values, in order to let the Wavecom OS know how many tasks are required to be started.
Up to three tasks can be declared & started by the Wavecom OS (any additional task declaration is ignored).

2.2.4 Stack Initialization

The following constants 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
```

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.2.5 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 );
```

2.2.5.1 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 wireless CPU 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.

2.2.5.2 Return Value

The returned value is not relevant.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.2.6 The Parser Functions

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

Their prototype is:

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

2.2.6.1 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;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

MsgTyp may have the following values:

MsgTyp value	Description
WM_AT_RESPONSE	the message includes an AT command response sent by the Embedded Application.
WM_AT_UNSOLICITED	the message includes an unsolicited AT response.
WM_AT_INTERMEDIATE	the message includes an intermediate AT response.
WM_AT_CMD_PRE_PARSER	the message includes an AT command sent by the External Application.
WM_AT_RSP_PRE_PARSER	the message includes a response processed by a Wavecom OS AT function.
WM_OS_TIMER	the message is sent when the timer expires.
WM_OS_RELEASE_MEMORY	the message includes the address of a released pointer.
WM_FCM_RECEIVE_BLOCK	the message includes data received by the Embedded Application.
WM_FCM_OPEN_FLOW	the requested flow opening operation is successful.
WM_FCM_CLOSE_FLOW	the requested flow closing operation is successful.
WM_IO_PORT_UPDATE_INFO	informs the Open AT® application that either a new port has been opened, or an existing port has been closed.
WM_FCM_RESUME_DATA_FLOW	the Embedded Application may resume its data sending operations.
WM_IO_SERIAL_SWITCH_STATE_RSP	includes the response to the serial link mode switching request.
WM_IO_UPDATE_INFO	The message informs the Open AT® application that a feature multiplexed with some GPIO has been enabled or disabled.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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            */
     u32 FCMCloseFlow

     /* WM_FCM_RESUME_DATA_FLOW */
     u32 FCMResumeFlow

     /* WM_IO_SERIAL_SWITCH_STATE_RSP */
     wm_ioSerialSwitchStateRsp_t     IOSerialSwitchStateRsp
     /* WM_IO_PORT_UPDATE_INFO */
     wm_ioPortUpdateInfo_t           IOPortUpdateInfo
     /* WM_IO_UPDATE_INFO   */
     wm_ioUpdateInfo_t               IOUpdateInfo;
} wm_apmBody_t;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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

***Body for** `WM_AT_RESPONSE:`*

```
typedef struct
{
  wm_atSendRspType_e Type;
  wm_ioPort_e        Dest;
  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 §2.3.3 for wm_atSendRspType_e description).

***Body for** `WM_AT_UNSOLICITED:`*

```
typedef struct    {
  wm_atUnsolicited_e  Type;
  u16             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 §2.3.4 for wm_atUnsolicited_e description).

***Body for** `WM_AT_INTERMEDIATE:`*

```
typedef struct
{
  wm_atIntermediate_e Type;
  wm_ioPort_e         Dest;
  u16                 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 §2.3.5 for wm_atIntermediate_e description).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

***Body for** `WM_AT_CMD_PRE_PARSER:`*

```
typedef struct    {
  wm_atCmdPreSubscribe_e   Type;
  wm_ioPort_e     Source;
  u16             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 §2.3.6 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 §2.3.7 for wm_atRspPreSubscribe_e description).

***Body for** `WM_OS_TIMER:`*

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

(See § 2.5.2 for timer identifier description).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

***Body for** `WM_OS_RELEASE_MEMORY:`*

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

(See §2.2.6 for this message description).

***Body for** `WM_FCM_RECEIVE_BLOCK:`*

```
typedef struct    {
u32     Reserved3;
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 §2.6.7 for `wm_fcmReceiveBlock_t` description and §2.6.2 for `wm_fcmFlow_e` description).

***Body for** `WM_FCM_OPEN_FLOW:`*

```
typedef struct    {
u32  FlowId;      /* opened IO flow ID */
u16  DataMaxToSend; /* max length of sent data */
} wm_fcmOpenFlow_t;
```

(See §2.6.4 for `wm_fcmOpenFlow_t` description and §2.6.2 for `wm_fcmFlow_e` description).

***Body for** `WM_FCM_CLOSE_FLOW:`*

(See §2.6.2 for `wm_fcmFlow_e` description).

***Body for** `WM_FCM_RESUME_DATA_FLOW:`*

(See §2.6.2 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 §2.7.2.2 for wm_ioSerialSwitchStateRsp_t description).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

***Body for** `WM_IO_SERIAL_SWITCH_STATE_RSP:`*

```
typedef struct
{
  wm_ioSerialSwitchState_e  SerialMode;    // mode requested
  s8                        RequestReturn; // <0 means error
  wm_ioPort_e               Port;          // required port
} wm_ioSerialSwitchStateRsp_t;
```

(See § 2.7.2.2 for wm_ioSerialSwitchStateRsp_t description).

***Body for** `WM_IO_PORT_UPDATE_INFO:`*

```
typedef struct
{
  wm_ioPort_e           Port;   // Port identifier
  wm_ioPortUpdateType_e Update; // Update type (Opened/Closed)
} wm_ioPortUpdateInfo_t;
```

(See §2.7.2.1 for more information about wm_ioPort_e description).
The wm_ioPortUpdateType_e type is described below :

```
typedef enum
{
    WM_IO_PORT_UPDATE_OPENED,   // New opened port
    WM_IO_PORT_UPDATE_CLOSED    // The port is now closed
} wm_ioPortUpdateType_e;
```

***Body for** `WM_IO_UPDATE_INFO:`*

```
typedef struct
{
    wm_ioUpdateType_e   UpdateType;
    u8                  FeaturesNb;
    wm_ioFeature_e      UpdatedFtrList[1];
} wm_ioUpdateInfo_t;
```

UpdateType field informs the application if the feature has been allocated or released, using the type below.

```
typedef enum
{
    WM_IO_UPDATE_TYPE_ALLOCATION,   // The feature has been
                                    allocated (linked GPIO are
                                    no more available)
    WM_IO_UPDATE_TYPE_RELEASE       // The feature has been
                                    released (linked GPIO are
                                    now available again)
} wm_ioUpdateType_e;
```

FeaturesNb field is the ***UpdatedFtrList*** table size.

UpdatedFtrList field is a table of feature identifiers which have just been updated (please refer to GPIO API description for more information).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.2.6.2 Return Values

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

2.2.6.3 Notes

- any S*trData[]* 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.

2.3 AT Command API

2.3.1 Required Header

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

2.3.2 The wm_atSendCommand Function

This function is just a shortcut to the wm_atSendCommandExt one, with the Dest parameter always set to the WM_IO_OPEN_AT_VIRTUAL_BASE value. Please refer to this function description for more information.

2.3.3 The wm_atSendCommandExt Function

The wm_atSendCommand function allows to send AT commands on a required port.

Its prototype is:

```
void  wm_atSendCommandExt (   u16                     AtStringSize,
                              wm_atSendRspType_e      ResponseType,
                              ascii                   *AtString,
                              wm_ioPort_e             Dest );
```

2.3.3.1 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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).

Dest:

Specifies the port on which the AT command has to be executed. Available ports may be opened and closed dynamically by any application (an external or an Open AT® one).
Each time a port is opened or closed, the Open AT® application receives a WM_IO_PORT_UPDATE message. The Open AT® application may also use the wm_ioIsPortAvailable function to know if a specific port is currently available.

Available values for this parameter are listed in the wm_ioPort_e type. All opened ports (ie. the ones on which the wm_ioIsPortAvailable returns TRUE) may be used to send an AT command, except the GSM & GPRS based ones (ie. the ones which have their four MSB set to WM_IO_ GSM_VIRTUAL_BASE or WM_IO_GPRS_VIRTUAL_BASE).

Used with the product physical outputs based ports (ie. all ports except the WM_IO_OPEN_AT_VIRTUAL_BASE based ones), this parameter will also :

- indicate on which port the responses have to be sent, in **WM_AT_SEND_RSP_TO_EXTERNAL** or **WM_AT_SEND_RSP_BROADCAST** mode.
- select the current port in order to set-up specific parameters (speed with AT+IPR, character framing with AT+ICF, etc...).

In **WM_AT_SEND_RSP_TO_EMBEDDED** or **WM_AT_SEND_RSP_BROADCAST** modes, this Dest parameter will be copied in the Dest field of the WM_AT_RESPONSE & WM_AT_INTERMEDIATE messages body structures, for each answer received in response of the sent AT command.

2.3.3.2 Notes

- As described in the "AT Commands Interface" document, AT commands sent by **wm_atSendCommandExt()** begin with the "AT" string, and end with a "\r" character (carriage return), except in some cases ("A/" command, SMS writing commands ("test\x1A"), ...)

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

- 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 .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.

2.3.3.3 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 OS using **the wm_atSendCommand** (see § 2.3.2) and **the wm_atSendRspExternalApp** (see § 2.3.8) functions.

- Example of sending an AT command:

```
wm_atSendCommand( 16, WM_AT_SEND_RSP_TO_EMBEDDED,
“ATD0146290800\r” );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

- Example of receiving an AT response:

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

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

         strBuffer    = &(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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.4 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);
```

2.3.4.1 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).

2.3.4.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.4.3 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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.5 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 );
```

2.3.5.1 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).

2.3.5.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.5.3 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;
   u16    nLenBuffer;

   switch (Message->MsgTyp)
   {
      ….
      case WM_AT_INTERMEDIATE:
         strBuffer    = &(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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.6 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);
```

2.3.6.1 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 OS ; 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.6.2 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 OS.
- 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 responsability to maintain an interface for other applications download and Open AT® start/stop mode. For exemple, it should filter +WDWL or +WOPEN command and require a password for download or application stop.

2.3.6.3 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);
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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_BRAODCAST)
           {
            /* Spying Embedded processing */
           …
           }
      …
   }
   return OK;
}
```

2.3.7 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 OS 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 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.7.1 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.

2.3.7.2 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 .
- 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 .

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.7.3 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;
   u16     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_BRAODCAST)     {
            /* Spying Embedded processing */
            …
         }
      …
   }
   return OK;
}
```

2.3.8 The wm_atSendRspExternalApp Function

This function is just a shortcut to the wm_atSendRspExternalAppExt one, with the Dest parameter always set to the WM_IO_UART1 value. Please refer to this function description for more information.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

The response is sent to the UART 1 physical port (if this port is not opened, the response will not be displayed anywhere).
It is strongly not recommended not to use this function ; the wm_atSendRspExternalAppExt one have to be used instead.

2.3.9 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.

2.3.9.1 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.
Available ports may be opened and closed dynamically by any application (an external or an Open AT® one).
Each time a port is opened or closed, the Open AT® application receives a WM_IO_PORT_UPDATE message. The Open AT® application may also use the wm_ioIsPortAvailable function to know if a specific port is currently available.
Available values for this parameter are listed in the wm_ioPort_e type.
All opened ports (ie. the ones on which the wm_ioIsPortAvailable returns TRUE) may be used to send an AT response, except the GSM, GPRS & Open AT® based ones (ie. the ones which have their four MSB set to WM_IO_ GSM_VIRTUAL_BASE, WM_IO_GPRS_VIRTUAL_BASE or WM_IO_OPEN_AT_VIRTUAL_BASE ; in this case the response will not be displayed anywhere).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

If the function is used to forward a response received from the WavecomOS to any external application, and if the WM_AT_SEND_RSP_HIGH_PRIORITY flag is set in the received response, then this flag has to be provided (through a bitwise OR combination) in the Dest parameter, in order to respect the response priority level. This flag is mainly used for the "CONNECT" response, issued just when a port is switched in data mode during a GSM data call (if this port is not used by the Open AT® application with the FCM API).

2.3.10 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 );
```

2.3.10.1 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.

2.3.10.2 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.

2.3.11 The wm_atSendUnsolicitedExternalAppExt Function

This function behaves exactly as the wm_atSendUnsolicitedExternalApp one, but provides an additional parameter in order just to send the unsolicited response on one specific port, instead of broadcasting it on all ports.

Its prototype is:

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


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Please note that if the Dest parameter is set to **WM_IO_NO_PORT** value, the unsolicited response will be broadcasted on all ports.

2.3.12 The wm_atSendIntermediateExternalApp Function

This function is just a shortcut to the wm_atSendIntermediateExternalAppExt one, with the Dest parameter always set to the WM_IO_UART1 value. Please refer to this function description for more information.

The wm_atSendIntermediateExternalApp function sends an AT intermediate response to the External Application. The response is sent to the UART 1 physical port (if this port is not opened, the response will not be displayed anywhere).
It is strongly not recommended not to use this function ; the wm_atSendIntermediateExternalAppExt one have to be used instead.

2.3.13 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 );
```

2.3.13.1 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.
Available ports may be opened and closed dynamically by any application (an external or an Open AT® one).
Each time a port is opened or closed, the Open AT® application receives a WM_IO_PORT_UPDATE message. The Open AT® application may also use the wm_ioIsPortAvailable function to know if a specific port is currently available.
Available values for this parameter are listed in the wm_ioPort_e type.
All opened ports (ie. the ones on which the wm_ioIsPortAvailable returns TRUE) may be used to send an AT intermediate response, except the GSM, GPRS & Open AT® based ones (ie. the ones which have their four MSB set to WM_IO_ GSM_VIRTUAL_BASE, WM_IO_GPRS_VIRTUAL_BASEor WM_IO_OPEN_AT_VIRTUAL_BASE; in this case the intermediate response will not be displayed anywhere).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.3.13.2 Notes

- An intermediate response string sent by the **wm_atSendIntermediateExternalAppExt** 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.

2.4 Debug API

2.4.1 Required Header

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

2.4.2 The wm_osDebugTrace Function

The **wm_osDebugTrace** function allows the application to send a debug trace to the Target Monitoring Tool.

Its prototype is:

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

2.4.2.1 Parameters

Level:
Used to differentiate the traces. The Target Monitoring Tool 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.

2.4.2.2 Returned values

This function always returns OK

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.4.2.3 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” */
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.4.3 The wm_osDebugFatalError Function

The **wm_osDebugFatalError** function allows the application to store a "backtrace" in the product memory, and to reset it. A backtrace is composed of the provided message, and a call stack "footprint" taken at the function call time. It is readable by the Target Monitoring Tool (Please refer to the Tools Manual for more information).

Its prototype is:

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

2.4.3.1 Parameters

Message:

String to be displayed whenever an error occurs, and to be stored with the backtrace in the product memory.
Please note that only the string address is stored in the backtrace, so this parameter has not to be a pointer on a RAM buffer, but a constant string pointer. Moreover, the string will only be correctly displayed if the current application is still present in the wireless CPU's flash memory. If the application is erased or modified, the string will not be correctly displayed when retrieving the backtraces.

2.4.3.2 Returned Value

None : this function will reset the product.

2.4.3.3 Note

The reboot is performed on 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 Open AT® application software startup is done after a 20 seconds delay.
Therefore, in order not to miss any event, any application has to handle the case of a startup delay of 20 seconds.
Moreover, in the product reset is due to a fatal error (from Open AT® application, or from Wavecom OS), the Init function's InitType parameter will be set to the **WM_APM_REBOOT_FROM_EXCEPTION** value.

2.4.4 The wm_osDebugEraseAllBacktraces Function

The **wm_osDebugEraseAllBacktraces** function allows the application to re-initialize the product backtraces storage place. All the currently stored backtraces will be erased.

Its prototype is:

```
void   wm_osDebugEraseAllBacktraces ( void );
```

2.4.5 The wm_osDebugInitBacktracesAnalysis Function

In order to start backtraces analysis, the **wm_osDebugInitBacktracesAnalysis** function have to be called before the first **wm_osDebugRetrieveBacktrace** function call. Each time a full backtraces analysis is required to be started, this function has to be called first.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Its prototype is:

```
s32    wm_osDebugInitBacktracesAnalysis ( void );
```

2.4.5.1 Returned Value

OK on success.
A negative internal error value if an unexpected error occurred.

2.4.6 The wm_osDebugRetrieveBacktrace Function

This function retrieves the next stored backtrace in the product's memory. Before the first call to this function, the **wm_osDebugInitBacktracesAnalysis** function has to be called in order to initialize the analysis. Successive calls to the **wm_osDebugRetrieveBacktrace** function will allow to retrieve all the backtraces, until the function returns a negative value.

Its prototype is:

```
s32    wm_osDebugRetrieveBacktrace ( u8 * BacktraceBuffer, u16 Size );
```

2.4.6.1 Parameters

BacktraceBuffer:
Buffer where the backtrace content will be copied. The buffer content will not be modified if it is not large enough. This parameter may be set to **NULL** in order to know the next backtrace buffer required size.

Size:
Backtrace buffer size. The buffer content will not be modified if it is not large enough. A backtrace buffer size will be at least about 550 bytes, + the stored message size.

2.4.6.2 Returned Value

OK on success.
A negative internal error value if there is no more backtrace stored in the product's memory, or if the **wm_osDebugInitBacktracesAnalysis** function was not called yet.

A positive size value, if the BacktraceBuffer argument was set to **NULL** : the next backtrace buffer required size is returned (at least 550 bytes, + the stored message size).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5 OS API

2.5.1 Required Header

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

2.5.2 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  );
```

2.5.2.1 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).

2.5.2.2 Return Values

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

2.5.2.3 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).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.2.4 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;
}
```

2.5.3 The wm_osStopTimer Function

The wm_osStopTimer function stops the timer identified by TimerId.

Its prototype is:

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

2.5.3.1 Parameter

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

2.5.3.2 Return Values

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

2.5.4 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  );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.4.1 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).

2.5.4.2 Return Values

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

2.5.4.3 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.

2.5.4.4 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 );
```

2.5.5 The wm_osStopTickTimer Function

The wm_osStopTickTimer function stops the timer identified by TimerId.

Its prototype is:

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

2.5.5.1 Parameter

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

2.5.5.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.6 Important Note on Data Flash Management

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.

2.5.7 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 );
```

2.5.7.1 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.

2.5.7.2 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)

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.8 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 );
```

2.5.8.1 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.

2.5.8.2 Return Values

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

2.5.9 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 );
```

2.5.9.1 Parameter

Id:
Identifier assigned to the stored data.

2.5.9.2 Return Values

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.10 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 );
```

2.5.10.1 Parameter

Id:

Identifier assigned to the stored data.

2.5.10.2 Return Values

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

2.5.11 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).

2.5.12 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.13 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 );
```

2.5.13.1 Parameters

StartId:
Range to browse first Id

EndId:
Range to browse last Id

2.5.13.2 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 or equal than Enid .

2.5.14 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).

2.5.15 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 );
```

2.5.15.1 Parameters

StartId:
Range to browse first Id

EndId:
Range to browse last Id

2.5.15.2 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 or equal than Enid.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.16 The wm_osGetHeapMemory Function

The wm_osGetHeapMemory function gets memory from the Embedded Application heap.

Its prototype is:

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

2.5.16.1 Parameter

MemorySize:
Requested size.

2.5.16.2 Return Values

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

2.5.17 The wm_osReleaseHeapMemory Function

The wm_osReleaseHeapMemory function releases a previously reserved memory buffer.

Its prototype is:

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

2.5.17.1 Parameter

PtrData:
Points to the reserved memory.

2.5.17.2 Return Values

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

2.5.18 The wm_osGetRamInfo Function

The **wm_osGetRamInfo** function retrieves information about the Open AT® RAM areas sizes.

Its prototype is:

```
s32 wm_osGetRamInfo ( wm_osRamInfo_t * Info );
```

2.5.18.1 Parameter

Info:
Open AT® RAM information structure, using the following type:

```
typedef struct
{
    u32 TotalSize;
    u32 StackSize;
    u32 HeapSize;
    u32 GlobalSize;
} wm_osRamInfo_t;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

- **TotalSize**
  Total RAM size for the Open AT® application (in bytes).
- **StackSize**
  Open AT® application call stack area size (in bytes).
- **HeapSize**
  Open AT® application total heap memory area size (in bytes).
- **GlobalSize**
  Open AT® application global variables area size (in bytes).

2.5.18.2 Return Values

OK on success.
ERROR on parameter error.

2.5.19 The wm_osSuspend function

The wm_osSuspend suspend the execution of the Open AT® 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® application running in Remote Task Environment can not be suspended (the function has no effect).

2.5.20 The wm_osGetTask Function

The wm_osGetTask function returns the current task ID.

Its prototype is:

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

2.5.20.1 Return Values

The return parameter is the current embedded application task ID.

2.5.21 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 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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.

2.5.21.1 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.

2.5.21.2 Return Values

The return parameter is the current embedded application task ID (Range of values is [0 ;(WM_APM_MAX_TASK – 1)].

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.5.22 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);
}
```

2.5.23 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);
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6 Flow Control Manager API

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

- Serial link based flows (UART 1, UART 2 physical outputs, or associated logical 27.010 protocol channels if any) ;
- Data Communication (through the GSM or GPRS network) ;
- Connected Bluetooth peripherals data access (Serial Port Profile only).

By default, these flows are closed to transmit all data directly between the V24 serial links and GSM, GPRS or Bluetooth flows. The Embedded Application can use the functions **wm_fcmOpen()** (see §2.6.4) and **wm_fcmClose()** (see §2.6.5) to open or close these flows.

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

2.6.1 Required Header

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

2.6.2 The wm_fcmFlow_e type

This type is the same as the **wm_ioPort_e** one. Please refer to the IO Port API for more information about the available ports. Previous versions defined flow identifiers have been kept for backward compatibility, but the new **wm_ioPort_e** ones should be used instead.

```
#define WM_FCM_DATA        WM_IO_GSM_BASE     // GSM CSD FCM flow
#define WM_FCM_GPRS        WM_IO_GPRS_BASE    // GPRS FCM flow
#define WM_FCM_V24         WM_IO_UART1        // UART 1 FCM flow
#define WM_FCM_V24_UART1   WM_IO_UART1        // UART 1 FCM flow
#define WM_FCM_V24_UART2   WM_IO_UART2        // UART 2 FCM flow
#define WM_FCM_USB         WM_IO_USB          // USB flow (reserved)
```

2.6.3 The wm_fcmIsAvailable Function

This function allows to check if the required port is available and ready to handle the FCM service.
Its prototype is :

```
bool wm_fcmIsAvailable ( wm_fcmFlow_e  FlowID );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6.3.1 Parameters

Flow

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

2.6.3.2 Return value

- TRUE if the flow is ready for the FCM service.
- FALSE if it is not ready.

2.6.3.3 Notes

All ports should be available for the FCM service, except :

- The Open AT® virtual one, which is only usable for AT commands,
- The Bluetooth virtual ones with enabled profiles other than the SPP one,
- If the port is already used to handle a feature required by an external application through the AT commands (+WLDB command, or a CSD/GPRS data session is already running)

2.6.4 The wm_fcmOpen Function

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

Its prototype is :

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

2.6.4.1 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 depends on the required flow :

- maximum **120 bytes** for a serial link based flow ;
- maximum **270 bytes** for the GSM CSD data flow ;
- not used for the GPRS flow ;
- maximum **120 bytes** for a Bluetooth based flow.

2.6.4.2 Return value

- WM_FCM_OK if successful.
- WM_FCM_ERR_NO_LINK if the requested flow can not be opened (the GSM and GPRS flows can not be opened together).
- WM_FCM_KO if the required flow is unknown or not available

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6.4.3 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.

2.6.5 The wm_fcmClose Function

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

Its prototype is:

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

2.6.5.1 Parameters

Flow

The flow to be closed, using the `wm_fcmFlow_e` type (see §2.6.2 for `wm_fcmFlow_e` description).

2.6.5.2 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.

2.6.5.3 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6.6 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 );
```

2.6.6.1 Parameters

Flow

Specifies the IO flow where the data are sent; See §2.6.2 for `wm_fcmFlow_e` description.

fcmDataBlock:

Pointer on a wm_fcmSendBlock_t structure, allocated (see § **2.5.16**: "**The wm_osGetHeapMemory** ") and filled by the Embedded Application before sending.
For example, to send 10 data bytes, the buffer 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];
    u32  Reserved3;
    u16  DataLength;       /* number of byte of data to send */
    u16  Reserved2[5];
    u8   Data[1];/* data buffer to send */
} wm_fcmSendBlock_t;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6.6.2 Returned Values

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

2.6.6.3 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6.7 Receive Data Blocks

The Embedded Application may receive data blocks from an opened IO flow, through the WM_FCM_RECEIVE_BLOCK message. This message is available as a parameter of the **wm_apmAppliParser()** function.

2.6.7.1 Message Parameters

This is the WM_FCM_RECEIVE_BLOCK message structure:

```
typedef struct   {
    u32          Reserved3;
    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 §2.6.4).

FlowID
: Specifies the opened IO flow from where the data are received. See §2.6.2 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 § 2.2.6).

2.6.7.2 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 § 2.6.8).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.6.8 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 );
```

2.6.8.1 Parameters

Flow:

Specifies the IO flow on which the Flow Control Manager may release credits. See §2.6.2 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.

2.6.8.2 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).

2.6.9 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 );
```

2.6.9.1 Parameters

Flow:

Specifies the IO flow from which the buffer status is requested. See §2.6.2for wm_fcmFlow_e description.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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;
```

2.6.9.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7 Input Output API

This API manages Serial Link State and GPIO operations.

2.7.1 Required Header

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

2.7.2 AT/FCM Ports related functions

2.7.2.1 The wm_ioPort_e type

This type is used to identify the product IO ports, usable to send AT commands and/or with the FCM service in order to exchange data with an external peripheral. The type is detailed below.

```
typedef enum
{
    WM_IO_NO_PORT,
    WM_IO_UART1,
    WM_IO_UART2,
    WM_IO_USB,

    WM_IO_UART1_VIRTUAL_BASE = 0x10,
    WM_IO_UART2_VIRTUAL_BASE = 0x20,
    WM_IO_USB_VIRTUAL_BASE   = 0x30,
    WM_IO_BLUETOOTH_VIRTUAL_BASE = 0x40,
    WM_IO_GSM_BASE = 0x50,
    WM_IO_GPRS_BASE = 0x60,
    WM_IO_OPEN_AT_VIRTUAL_BASE = 0x80
} wm_ioPort_e;
```

The available ports are described hereafter :

- WM_IO_NO_PORT
  Not usable
- WM_IO_UART1
  Product physical UART 1
- WM_IO_UART2
  Product physical UART 2
- WM_IO_USB
  Product physical USB port (reserved for future products)
- WM_IO_UART1_VIRTUAL_BASE
  Base ID for 27.010 protocol logical channels on UART 1
- WM_IO_UART2_VIRTUAL _BASE
  Base ID for 27.010 protocol logical channels on UART 2
- WM_IO_USB_VIRTUAL _BASE
  Base ID for 27.010 protocol logical channels on USB link (reserved for future products)
- WM_IO_BLUETOOTH_VIRTUAL_BASE
  Base ID for connected Bluetooth peripheral virtual port.
  ONLY USABLE WITH THE FCM SERVICE

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

- WM_IO_GSM_BASE
  GSM CSD data flow identifier for FCM API
  This flow is always considered as available (no update messages)
  ONLY USABLE WITH THE FCM SERVICE
- WM_IO_GPRS_BASE
  GPRS data flow identifier for FCM API
  This flow is always considered as available if the GPRS feature is enabled, or unavailable if this feature is disabled (no update messages)
  ONLY USABLE WITH THE FCM SERVICE
- WM_IO_OPEN_AT_VIRTUAL_BASE
  Base ID for AT commands contexts dedicated to Open AT® applications
  ONLY USABLE WITH AT COMMANDS

2.7.2.2 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);
```

2.7.2.2.1 Parameters

Port

Specifies the IO port to switch the state, using the wm_ioPort_e type.

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_IO_SERIAL_AT_OFFLINE     // Incoming message only
} 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")
WM_IO_SERIAL_AT_OFFLINE is only received in the WM_IO_SERIAL_SWITCH_STATE_RSP message, when the external application used the "+++" sequence to switch back the serial link in AT mode.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7.2.2.2 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 § 2.2.6). 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. In this case, the WM_IO_SERIAL_SWITCH_STATE_RSP message will always return a negative **RequestReturn** parameter.
- In Figure 2 (see § 2.6), 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.
  In this case, the Open AT® application will receive a WM_IO_SERIAL_SWITCH_STATE_RSP message with the SerialMode field set to the WM_IO_SERIAL_AT_OFFLINE state.

2.7.2.3 The wm_ioSerialGetSignal Function

The wm_ioSerialGetSignal function allows to get the current values of the CTS and DSR signals of the required port.

Its prototype is :

```
s32 wm_ioSerialGetSignal ( wm_ioPort_e             Port,
                           wm_ioSerialGetSignal_e  SerialSignal )
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7.2.3.1 Parameters

Port:
Required port from which to query the signal state, based on the wm_ioPort_e type. Only physical output related ports (UARTX & USB ones, used as physical ports, or with the 27.010 protocol) may be used with this function.

SerialSignal:
Value designating the signal to get, using following type :

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

2.7.2.3.2 Returned Values

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

2.7.2.4 The wm_ioIsPortAvailable Function

The wm_ioIsPortAvailable function allows to query the required port state (opened or closed).

Its prototype is :

```
bool wm_ioIsPortAvailable ( wm_ioPort_e Port )
```

2.7.2.4.1 Parameters

Port :
Port from which to query the state.

2.7.2.4.2 Returned Values

TRUE if the port is currently opened,
FALSE otherwise.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7.3 GPIO API

2.7.3.1 Types

2.7.3.1.1 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.

2.7.3.1.2 The wm_ioLabel_u union

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

```
typedef union
{
} wm_ioLabel_u;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Wismo QUIK Q2686 Gpio Labels

The following table lists the Gpio labels for Wismo Quik Q2686 products. Linked features are indicated when applicable.

GPIO identifier	Linked feature (if any)
WM_IO_Q2686_GPIO_1	
WM_IO_Q2686_GPIO_2	
WM_IO_Q2686_GPIO_3	WM_IO_FEATURE_INT0
WM_IO_Q2686_GPIO_4	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_5	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_6	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_7	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_8	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_9	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_10	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_11	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_12	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_13	WM_IO_FEATURE_KBD
WM_IO_Q2686_GPIO_14	WM_IO_FEATURE_UART2
WM_IO_Q2686_GPIO_15	WM_IO_FEATURE_UART2
WM_IO_Q2686_GPIO_16	WM_IO_FEATURE_UART2
WM_IO_Q2686_GPIO_17	WM_IO_FEATURE_UART2
WM_IO_Q2686_GPIO_18	WM_IO_FEATURE_SIMPRES
WM_IO_Q2686_GPIO_19	
WM_IO_Q2686_GPIO_20	
WM_IO_Q2686_GPIO_21	
WM_IO_Q2686_GPIO_22	WM_IO_FEATURE_BT_RST
WM_IO_Q2686_GPIO_23	
WM_IO_Q2686_GPIO_24	
WM_IO_Q2686_GPIO_25	WM_IO_FEATURE_INT1
WM_IO_Q2686_GPIO_26	WM_IO_FEATURE_BUS_I2C
WM_IO_Q2686_GPIO_27	WM_IO_FEATURE_BUS_I2C
WM_IO_Q2686_GPIO_28	WM_IO_FEATURE_BUS_SPI1_CLK
WM_IO_Q2686_GPIO_29	WM_IO_FEATURE_BUS_SPI1_IO
WM_IO_Q2686_GPIO_30	WM_IO_FEATURE_BUS_SPI1_I
WM_IO_Q2686_GPIO_31	WM_IO_FEATURE_BUS_SPI1_CS
WM_IO_Q2686_GPIO_32	WM_IO_FEATURE_BUS_SPI2_CLK
WM_IO_Q2686_GPIO_33	WM_IO_FEATURE_BUS_SPI2_IO
WM_IO_Q2686_GPIO_34	WM_IO_FEATURE_BUS_SPI2_I
WM_IO_Q2686_GPIO_35	WM_IO_FEATURE_BUS_SPI2_CS
WM_IO_Q2686_GPIO_36	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_37	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_38	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_39	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_40	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_41	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_42	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_43	WM_IO_FEATURE_UART1
WM_IO_Q2686_GPIO_44	

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

As soon as a multiplexed feature is used (or stopped), a **WM_IO_UPDATE_INFO** message is notified to the Open AT® application in order to inform it that the related GPIO is available or is not available).

```
typedef enum
{
     WM_IO_Q2686_GPIO_1 = 3,
     WM_IO_Q2686_GPIO_2,
     WM_IO_Q2686_GPIO_3,
     …
     WM_IO_Q2686_GPIO_43,
     WM_IO_Q2686_GPIO_44,
     WM_IO_Q2686_PAD    = 0x7FFFFFFF
} wm_ioLabel_Q2686_e;
```

2.7.3.1.3 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  // for GPO only, deprecated
} 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.

2.7.3.1.4 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.

2.7.3.1.5 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.

2.7.3.1.6 The wm_ioRead_t structure

This type is used by the wm_ioRead function to read one or several GPIO value.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
typedef struct
{
      wm_ioLabel_u  eLabel;
      wm_ioState_e  eState;
} wm_ioRead_t;
```

The eLabel member represents the GPIO label.
The eState member will be set by the function to the GPIO read value.

2.7.3.1.7 The wm_ioWrite_t structure

This type is used by the wm_ioWrite function to write one or several GPIO value.

```
typedef struct
{
      wm_ioLabel_u  eLabel;
      wm_ioState_e  eState;
} wm_ioWrite_t;
```

The eLabel member represents the GPIO label.
The eState member represents the GPIO value to be set on the output.

2.7.3.1.8 The wm_ioFeature_e type

This type enumerates the wireless CPU features which are multiplexed with one or several GPIO.

```
typedef enum
{
    WM_IO_FEATURE_NONE,
    WM_IO_FEATURE_BUS_SPI1_CLK,  // SPI 1 bus clock pin
    WM_IO_FEATURE_BUS_SPI1_IO,   // SPI 1 bus bi-directional data pin
    WM_IO_FEATURE_BUS_SPI1_I,    // SPI 1 bus uni-directional input pin
    WM_IO_FEATURE_BUS_SPI1_CS,   // SPI 1 bus hardware chip select pin
    WM_IO_FEATURE_BUS_SPI2_CLK,  // SPI 2 bus clock pin
    WM_IO_FEATURE_BUS_SPI2_IO,   // SPI 2 bus bi-directional data pin
    WM_IO_FEATURE_BUS_SPI2_I,    // SPI 2 bus uni-directional input pin
    WM_IO_FEATURE_BUS_SPI2_CS,   // SPI 2 bus hardware ship select pin
    WM_IO_FEATURE_BUS_I2C,       // I2C bus
    WM_IO_FEATURE_BUS_PARALLEL,  // Reserved for future use
    WM_IO_FEATURE_KBD,           // Keypad
    WM_IO_FEATURE_SIMPRES,       // SIM presence signal
    WM_IO_FEATURE_UART1,         // UART 1 port
    WM_IO_FEATURE_UART2,         // UART 2 port
    WM_IO_FEATURE_INT0,          // External interrupt 0
    WM_IO_FEATURE_INT1,          // External interrupt 1
    WM_IO_FEATURE_BT_RST,        // Bluetooth reset signal
    WM_IO_FEATURE_LAST
} wm_ioFeature_e;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7.3.1.9 Return values definition

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

2.7.3.2 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 );
```

2.7.3.2.1 Parameters

NbGpioToAllocate :

Size of the GpioCustomerConfig array.

GpioCustomerConfig :

Array of values, defined by the wm_ioConfig_t structure.
For each member of this array :

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

2.7.3.2.2 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 one of the reserved GPIO. Otherwise, a negative returned value represents an error.

2.7.3.2.3 Notes

- 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

- After a successful allocation, any GPIO allocated by the Embedded Application is no longer available for AT commands (AT+WIOR, AT+WIOW, AT+WIOM) or linked features (cf. **wm_ioFeature_e** type definition).

2.7.3.3 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 );
```

2.7.3.3.1 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.
Each member of this array represents the label of one GPIO to release.

2.7.3.3.2 Returned Values

OK: successful completion
Otherwise, a negative returned value represents an error.

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), and GPIO is available again for any linked feature.

2.7.3.4 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 );
```

2.7.3.4.1 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

GpioDirection :
Array of values, defined by the wm_ioSetDirection_t structure.

For each member of this array :
- eLabel represents the label of the GPIO to change direction, depending on the used product.
- eDirection represents the new direction to use for this GPIO.

2.7.3.4.2 Returned Values

OK: successful completion
Otherwise, a negative returned value represents an error.

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

2.7.3.5 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 NbGpioToRead,
                  wm_ioRead_t * GpioToRead );
```

2.7.3.5.1 Parameters

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

NbGpioToRead :
Size of the GpioToRead array.

GpioToRead :
Array of values, defined by the wm_ioRead_t structure.
For each member of this array:
- eLabel represents the label of the GPIO to read.
- Read value will be copied in the eState field.

2.7.3.5.2 Returned Values

OK: successful completion
Otherwise, a negative returned value represents an error.

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

2.7.3.6 The wm_ioSingleRead Function

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

Its prototype is :

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

2.7.3.6.1 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.

2.7.3.6.2 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.

Notes

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

2.7.3.7 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 NbGpioToWrite,
                  wm_ioWrite_t*  GpioToWrite );
```

2.7.3.7.1 Parameters

Handle :

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

NbGpioToWrite :

Size of the GpioToWrite array.

GpioToWrite :

Array of values, defined by the wm_ioWrite_t structure.

For each member of this array:

- eLabel represents the label of the GPIO to write.
- eState represents the value to set on the output.

2.7.3.7.2 Returned Values

OK: successful completion
Otherwise, a negative returned value represents an error.

Notes

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7.3.8 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 );
```

2.7.3.8.1 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).

2.7.3.8.2 Returned Values

OK: successful completion
Otherwise, a negative returned value represents an error.

Notes

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

2.7.3.9 The wm_ioGetGPIOListForFeature Function

This function allows to dynamically get a GPIO list associated to a feature.

Its prototype is :

```
s32 wm_ioGetGPIOListForFeature ( wm_ioFeature_e Feature,
                                 u8 * GpioTab,
                                 u8 GpioNumber );
```

2.7.3.9.1 Parameters

Feature:

Feature from which to get the multiplexed GPIO list.

GpioTab:

Returned list of GPIO identifier associated to the required feature.
If set to NULL, the GPIO number will just be returned by the function.

GpioNumber:

Size of the GpioTab array.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.7.3.9.2 Returned Values

ERROR if the feature is unknown.
WM_IO_EXCEED_MAX_NUMBER if the provided GpioNumber parameter does not match with the required array size.
GPIO number associated to the feature on success (if GpioTab is not NULL, it will be updated with the GPIO list associated to the required feature).

2.7.3.10 The wm_ioIsFeatureEnabled Function

This function allows to check if a feature is currently enabled or not.

Its prototype is :

```
bool wm_ioIsFeatureEnabled ( wm_ioFeature_e Feature );
```

2.7.3.10.1 Parameters

Feature:
Feature to be checked.

2.7.3.10.2 Returned Values

FALSE: if the feature is disabled, or if it is unknown all the associated GPIOs are all available for allocation.
TRUE: if the feature is enabled: the associated GPIOs are not available for allocation.

2.7.3.10.3 Note

Each time a feature is enabled or disabled, a WM_IO_UPDATE_INFO message is sent to notify to the Open AT® application.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.8 GPRS API

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

2.8.1 GPRS Overview

2.8.1.1 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.8.1.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.8.2 The wm_gprsAuthentification function

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

Its prototype is :

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

2.8.2.1 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:

Authentification parameters, which allow the application to supply the PDP context login and password strings.
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.

2.8.2.2 Required Header

```
Wm_gprs.h
```

2.8.2.3 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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.8.3 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)
```

2.8.3.1 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.

2.8.3.2 Required Header

```
Wm_gprs.h
```

2.8.3.3 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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.8.4 The wm_gprsOpen function

This command sets Open AT® 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**.

2.8.4.1 Parameters

Cid:

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

2.8.4.2 Required Header

```
Wm_gprs.h
```

2.8.4.3 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

2.8.5 The wm_gprsClose function

This command unsets Open AT® 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**.

2.8.5.1 Parameters

Cid:

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

2.8.5.2 Required Header

```
Wm_gprs.h
```

2.8.5.3 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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.9 BUS API

This API manages the I2C and SPI bus operations.

2.9.1 Required Header

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

2.9.2 Returned values definition

Returned Value	Description
WM_BUS_PROC_DONE (0)	the function processing is successfully done.
WM_BUS_MODE_UNKNOWN_TYPE (-1)	unknown open mode type.
WM_BUS_UNKNOWN_TYPE (-11)	unknown bus type.
WM_BUS_BAD_PARAMETER (-12)	a parameter has an incorrect value.
WM_BUS_SPI_ALREADY_USED (-13)	the SPI bus is already opened.
WM_BUS_I2C_HARD_ALREADY_USED (-14)	the I2C bus is already opened.
WM_BUS_UNKNOWN_HANDLE (-21)	the handle used has an incorrect value.
WM_BUS_HANDLE_NOT_OPENED (-22)	no existing handle for this bus.
WM_BUS_NO_MORE_HANDLE_FREE (-23)	no more handles available for this bus.
WM_BUS_NOT_CONNECTED_ON_I2C (-31)	no peripheral connected on I2C bus.
WM_BUS_NOT_ALLOWED_ADDRESS (-32)	unknown address.
WM_BUS_SPI_GPIO_CS_NOT_GPIO (-36)	the considered chip select pin is not a GPIO or a GPO.
WM_BUS_SPI_GPIO_CS_NOT_FREE (-37)	the considered chip select GPIO is not free.
WM_BUS_SPI1_CS_HARD_NOT_FREE (-48)	Hardware chip select is already used for SPI1 bus
WM_BUS_SPI2_CS_HARD_NOT_FREE (-49)	Hardware chip select is already used for SPI2 bus

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.9.3 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 );
```

2.9.3.1 Parameters

BusType:

Type of the bus to open. Defined values are :

- WM_BUS_SPI1 for SPI 1 bus;
- WM_BUS_SPI2 for SPI 2 bus;
- WM_BUS_HARD_I2C for I2C bus.

Mode :

Bus mode: the only defined value is WM_BUS_MODE_STANDARD.

Settings :

Pointer on settings union, defined as below.

```
typedef union
{
    wm_busSPISettings_t          SPI;
    wm_busI2CHardSettings_t I2C_Hard;
} wm_busSettings_u;
```

2.9.3.1.1 SPI bus settings

To open the SPI bus the application has to 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;
    u32      DataLinesConf;

} wm_busSPI1Settings_t;
```

- The "*Clk_Speed*" parameter is the SPI clock speed. Allowed values are in the [0 – 127] range. The output data rate is reckoned using the formula:

  ```
  Pclk / ( ( 2 * Clk_Speed ) + 2 )
  ```

  Where **Pclk** is the Wireless CPU Peripheral Clock speed.
  On the Q2686 wireless CPU, **Pclk** rate is 26 MHz.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

- 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_CS_GPIO	Use a GPIO as ChipSelect, the *GpioChipSelect* parameter must be used
WM_BUS_SPI_ADDRESS_CS_HARD	Use the reserved hardware chip select pin for the required bus.
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 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 has to be a member of the wm_ioLabel_u union (see the GPIO API description).

- The "*WriteByteHandling*" parameter defines the chip select signal behavior; defined values are :
  - WM_BUS_SPI_WORD_HANDLING (chip select signal state changes on each written or read word; word size is defined on read or write process request, in the AccessMode configuration structure);
  - WM_BUS_SPI_FRAME_HANDLING (Chip select signal is enabled at the beginning of the read/write process, and is disabled at the end of this process).
  - Note:
    This mode is not available with the WM_BUS_SPI_ADDRESS_CS_HARD Chip Select configuration.

- The "*DataLinesConf*" parameter defines if the SPI bus uses one single pin to handle both input and output data signals, or two pins to handle them separately. Defined values are:
  - WM_BUS_SPI_DATA_BIDIR (One bi-directional pin is used to handle both input & output data signals);
  - WM_BUS_SPI_DATA_UNIDIR (Two pins are used to handle separately input & output data signals).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.9.3.1.2 I2C bus

To open the I2C bus the application has to use the "*I2C_Hard*" parameter of the union, defined below:

```
typedef struct
{
     u32       ChipAddress;
     u32       Clk_Speed;

} wm_busI2CHardSettings_t;
```

- The *ChipAddress* parameter is the remote chip address on the bus. Only the b1 to b7 bits are used to define this address.

- The *Clk_Speed* parameter sets the required I2C bus speed. Defined values are:
  - WM_BUS_I2CHARD_CLK_STD (standard I2C bus speed, 100 Kbit/s)
  - WM_BUS_I2CHARD_CLK_FAST (fast I2C bus speed, 400 Kbit/s)

2.9.3.2 Returned Values

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.9.3.3 Notes

A bus is available only if the GPIO multiplexed with the corresponding feature (according to the required configuration) are not yet allocated by an Open AT® application. Concerned features depends on the bus type and configuration:

Bus type & configuration	Multiplexed features
WM_BUS_HARD_I2C bus	**WM_IO_FEATURE_BUS_I2C**
WM_BUS_SPI1 bus, not using the hardware chip select pin, using one bi-directional data pin	**WM_IO_FEATURE_BUS_SPI1_CLK,** **WM_IO_FEATURE_BUS_SPI1_IO**
WM_BUS_SPI1 bus, using the hardware chip select pin, using one bi-directional data pin	**WM_IO_FEATURE_BUS_SPI1_CLK,** **WM_IO_FEATURE_BUS_SPI1_IO,** **WM_IO_FEATURE_BUS_SPI1_CS**
WM_BUS_SPI1 bus, not using the hardware chip select pin, using two data pin	**WM_IO_FEATURE_BUS_SPI1_CLK,** **WM_IO_FEATURE_BUS_SPI1_IO,** **WM_IO_FEATURE_BUS_SPI1_I**
WM_BUS_SPI1 bus, using the hardware chip select pin, using two data pin	**WM_IO_FEATURE_BUS_SPI1_CLK,** **WM_IO_FEATURE_BUS_SPI1_IO,** **WM_IO_FEATURE_BUS_SPI1_I,** **WM_IO_FEATURE_BUS_SPI1_CS**
WM_BUS_SPI2 bus, not using the hardware chip select pin, using one bi-directional data pin	**WM_IO_FEATURE_BUS_SPI2_CLK,** **WM_IO_FEATURE_BUS_SPI2_IO**
WM_BUS_SPI2 bus, using the hardware chip select pin, using one bi-directional data pin	**WM_IO_FEATURE_BUS_SPI2_CLK,** **WM_IO_FEATURE_BUS_SPI2_IO,** **WM_IO_FEATURE_BUS_SPI2_CS**
WM_BUS_SPI2 bus, not using the hardware chip select pin, using two data pin	**WM_IO_FEATURE_BUS_SPI2_CLK,** **WM_IO_FEATURE_BUS_SPI2_IO,** **WM_IO_FEATURE_BUS_SPI2_I**
WM_BUS_SPI2 bus, using the hardware chip select pin, using two data pin	**WM_IO_FEATURE_BUS_SPI2_CLK,** **WM_IO_FEATURE_BUS_SPI2_IO,** **WM_IO_FEATURE_BUS_SPI2_I,** **WM_IO_FEATURE_BUS_SPI2_CS**

Once the bus is opened, the features corresponding to the required configuration are enabled, and the multiplexed GPIO are no more available for allocation by the Open AT® application, or through the standard AT commands.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.9.4 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 );
```

2.9.4.1 Parameters

Handle :

Handle of the bus to close, returned by wm_busOpen function.

2.9.4.2 Returned Values

On successful completion, the function returns 0.
Otherwise, the function will return a negative error value (cf Returned values definition above).

Note

Once a bus is closed, the features corresponding to the required configuration are disabled, and the multiplexed GPIO are available again for allocation by the Open AT® application, or through the standard AT commands.

2.9.5 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 Size );
```

2.9.5.1 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_busSize_e AccessSize;
} wm_busAccess_t;
```

This structure parameters usage depends on the bus type.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

	SPI bus	I2C bus
Address	Up to 32 bits to be sent on the bus before writing the data buffer. If less than 32 bits are required, only the most significant bits are sent.	Up to 32 bits to be sent on the bus before writing the data buffer. If less than 32 bits are required, only the most significant bits are sent.
AddressLength	Length of the Address parameter. Values range is [0-32].	Length of the Address parameter. Allowed values are 8, 16, 24, 32.
Opcode	Up to 32 bits to be sent on the bus before writing the data buffer. If less than 32 bits are required, only the most significant bits are sent.	Ignored.
OpcodeLength	Length of the Opcode parameter; values range is [0-32].	Ignored.
AccessSize	Data format of the provided data buffer (from 1 to 16 bits), using the **wm_busSize_e** type.	Always set to the **WM_BUS_SIZE_BYTE** value (data buffer always contains whole bytes).

pDataToWrite :

Buffer containing data to write on the requested bus.
According to AccessSize configuration, this parameter has to use the following types:

- 1 <= AccessSize <= 8 : the data buffer has to use the u8 * type
- 9 <= AccessSize <= 16 : the data buffer has to use the u16 * type

If the AccessSize parameter is not the 8 or 16 value, only the less significant bits are written.

Size:

Size of the pDataToWrite buffer (according to the AccessSize configuration).

2.9.5.2 Returned Values

On successful completion, the function returns the number of bytes written.
Otherwise, the function will return a negative error value (cf Returned values definition above).

2.9.6 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 Size );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.9.6.1 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_busSize_e AccessSize;
} wm_busAccess_t;
```

This structure parameters usage depends on the bus type.

	SPI bus	I2C bus
Address	Up to 32 bits to be sent on the bus before reading the data buffer. If less than 32 bits are required, only the most significant bits are sent.	Up to 32 bits to be sent on the bus before reading the data buffer. If less than 32 bits are required, only the most significant bits are sent.
AddressLength	Length of the Address parameter. Values range is [0-32].	Length of the Address parameter. Allowed values are 8, 16, 24, 32.
Opcode	Up to 32 bits to be sent on the bus before reading the data buffer. If less than 32 bits are required, only the most significant bits are sent.	Ignored.
OpcodeLength	Length of the Opcode parameter; values range is [0-32].	Ignored.
AccessSize	Data format of the provided data buffer (from 1 to 16 bits), using the **wm_busSize_e** type.	Always set to the **WM_BUS_SIZE_BYTE** value (data buffer always contains whole bytes).

If required, “Address” and/or “Opcode” data are written on the bus, prior to reading data from the remote chip.
For I2C bus, using the “Address” data allows to send data to the chip and then read from it without sending a STOP instruction on the bus between the read and write processes.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

pDataToRead :

Buffer containing data read from the requested bus.
According to AccessSize configuration, this parameter has to use the following types:

- 1 <= AccessSize <= 8 : the data buffer has to use the u8 * type
- 9 <= AccessSize <= 16 : the data buffer has to use the u16 * type

If the AccessSize parameter has not the 8 or 16 value, only the less significant bits are written.

Size:

Size of the pDataToRead buffer (according to the AccessSize configuration).

2.9.6.2 Returned Values

On successful completion, the function returns the number of bytes read.
Otherwise, the function will return a negative error value (cf Returned values definition above).

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10 Lists management API

2.10.1 Required Header

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

2.10.2 Types definition

2.10.2.1 The wm_lst_t type

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

```
typedef void * wm_lst_t;
```

2.10.2.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.3 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 );
```

2.10.3.1 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.

2.10.3.2 Returned Values

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

2.10.4 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.5 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.

2.10.6 The wm_lstGetCount Function

The wm_lstGetCount function returns the current item count.

Its prototype is :

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

2.10.6.1 Parameters

list :
The list from which to get the item count.

2.10.6.2 Returned Values

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.7 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 );
```

2.10.7.1 Parameters

list : The list to add an item to.

item : The item to add to the list.

2.10.7.2 Returned Values

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

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.

2.10.8 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 );
```

2.10.8.1 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.

2.10.8.2 Returned Values

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

2.10.8.3 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.9 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 );
```

2.10.9.1 Parameters

list :
The list from which to get the item.

index :
The location where to get the item.

2.10.9.2 Returned Values

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

2.10.10 The wm_lstDeleteItem Function

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

Its prototype is :

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

2.10.10.1 Parameters

list :
The list to delete an item from.

index :
The location where to delete the item.

2.10.10.2 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.11 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 );
```

2.10.11.1 Parameters

list : The list where to search.

item : The item to find.

2.10.11.2 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.

2.10.12 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 );
```

2.10.12.1 Parameters

list : The list where to search.

item : The item to find.

2.10.12.2 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 cannot be changed during the resulting buffer processing. Otherwise the `wm_lstFindNextItem` should be used.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.10.13 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 );
```

2.10.13.1 Parameters

list : The list to search in.

item : The item to find.

2.10.13.2 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.

2.10.14 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 );
```

2.10.14.1 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.11 Sound API

2.11.1 Required header

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

2.11.2 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 );
```

2.11.2.1 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`**.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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

2.11.2.2 Returned values

OK on success,or a negative error value

2.11.2.3 Example:

An example of playing tone:

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

2.11.3 The wm_sndTonePlayExt Function

This function allows a dual tone (two frequencies) to be played on the specified output. Frequencies, gains and duration can be specified.

Note : only the speaker output is able to play two frequencies tones. The second tone parameters will be ignored on the buzzer output.

Its prototype is:

```
s32  wm_sndTonePlayExt ( wm_snd_dest_e     Destination,
                         u16               Frequency,
                         u16               Frequency2,
                         u8                Duration,
                         u8                Gain,
                         u8                Gain2 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.11.3.1 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, Frequency2:

For speaker : range is from 1 Hz to 3999 Hz.
For buzzer : range is from 1 Hz to 50000 Hz.

Please remind that the Frequency2 parameter will be only processed on the speaker output.

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, Gain2 :

This parameter sets the tones gain. Gain parameter applies to Frequency value, and Gain2 applies to the Frequency2 one.

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

2.11.3.2 Returned values

OK on success, or a negative error value

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.11.3.3 Example:

An example of playing tone:

```
wm_sndTonePlayExt ( WM_SND_DEST_SPEAKER, 1000, 2000, 0, 9, 10 );
```

2.11.4 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 );
```

2.11.4.1 Parameters

Destination:

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

2.11.4.2 Returned values

OK on success, or a negative error value.

2.11.4.3 Example:

An example of stopping tone:

```
wm_sndToneStop ( WM_SND_DEST_BUZZER );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.11.5 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 );
```

2.11.5.1 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.

2.11.5.2 Returned values

OK on success, or a negative error value

2.11.5.3 Example:

An example of playing DTMF:

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

2.11.6 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 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.11.6.1 Parameters

Destination:

Destination of the current playing tone to stop, this has to be a speaker (GSM network DTMF cannot be stopped).

2.11.6.2 Returned values

OK on success, or a negative error value

2.11.6.3 Example:

An example of stopping DTMF:

```
wm_sndDtmfStop ( WM_SND_DEST_SPEAKER );
```

2.11.7 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 );
```

2.11.7.1 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  ,
};
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
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.

2.11.7.2 Returned values

OK on success, or a negative error value

2.11.7.3 Example:

An example of playing melody:

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

2.11.8 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 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.11.8.1 Parameters

Destination:

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

2.11.8.2 Returned values

OK on success, or a negative error value

2.11.8.3 Example:

An example of stopping a melody:

```
wm_sndMelodyStop ( WM_SND_DEST_SPEAKER );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.12 Standard Library

2.12.1 Required Header

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

2.12.2 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
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.12.3 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 an 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 an 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 an 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 an 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 an 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 an 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".

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13 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 OS update, an Open AT® application, or an E2P configuration file).

The total Application & Data Storage volume size is configurable with the AT+WOPEN command.

2.13.1 Required Header

This API is defined in **wm_ad.h** header file.
This file is included by **wm_apm.h**.

2.13.2 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.

2.13.3 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 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.3.1 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.

2.13.3.2 Returned Values

This function will return OK if successful, otherwise, it will return an error value (please refer to § 2.13.2 "Returned values definition").

2.13.4 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 );
```

2.13.4.1 Parameters

CellId
Unique identifier of the cell to retrieve.

Handle
Returned handle on the retrieved cell.

2.13.4.2 Returned Values

This function will return OK if successful, otherwise, it will return an error value (cf § 2.13.2 "Returned values definition").

2.13.5 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 );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.5.1 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.

2.13.5.2 Returned Values

This function will return OK if successful, otherwise, it will return an error value (cf § 2.13.2 "Returned values definition").

2.13.6 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 );
```

2.13.6.1 Parameters

BrowseInfo
Browse informations, returned by the wm_adFindInit() function.
Handle
Next found cell handle.

2.13.6.2 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 § 2.13.2 "Returned values definition").

2.13.7 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 );
```

2.13.7.1 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.7.2 Returned Values

This function will return OK if successful, otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

2.13.8 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 );
```

2.13.8.1 Parameters

Handle

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

2.13.8.2 Returned Values

This function will return OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

2.13.9 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 );
```

2.13.9.1 Parameters

Handle

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

2.13.9.2 Returned Values

This function will return OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.10 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 );
```

2.13.10.1 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;
```

2.13.10.2 Returned Values

This function will return OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

2.13.11 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 );
```

2.13.11.1 Parameters

Handle

Handle on the cell to delete.

2.13.11.2 Returned Values

This function will return OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.12 The wm_adStats Function

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

Its prototype is :

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

2.13.12.1 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;
```

2.13.12.2 Returned Values

This function will return OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

2.13.13 The wm_adSpaceState Function

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

Its prototype is :

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

2.13.13.1 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
                        // recompaction process. The application has
                        // to call wm_adRecompactInit to continue
                        // this process.
} wm_adSpaceState_e;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.14 The wm_adFormat Function

The **wm_adFormat** function destroys the whole Application & Data space stored data. The function has to be called a first time with the **WM_AD_FORMAT_INIT** mode, to initialize the format process. Then it has to be called regularly (Eg. on a cyclic timer reception) with the **WM_AD_FORMAT_CONTINUE** mode, until the format process is completed.

Its prototype is :

```
s32 wm_adFormat (wm_adFormatMode_e Mode,
                 u32 *             FormatHandle,
                 u32 *             FormatProgress );
```

2.13.14.1 Parameters

Mode

Required operation mode, using the following type:

```
typedef enum
{
    WM_AD_FORMAT_INIT,      // Initialize the format process
    WM_AD_FORMAT_CONTINUE,  // Continue running the format process
    WM_AD_FORMAT_ABORT      // Abort format process
                            // (need to be re-started later)
} wm_adFormatMode_e;
```

FormatHandle

Pointer on an u32 integer, modified by the function; the same pointer has to be used for every function call during the format process.

FormatProgress

Format process progress indicator (as a percentage) updated by the function. The process is complete when this indicator reaches the 100 value.

2.13.14.2 Returned Values

This function will return OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

2.13.15 The wm_adRecompactInit Function

The wm_adRecompactInit function starts the recompaction process. The process steps are then done by the wm_adRecompcact() 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 recompaction completion**. If the recompaction is interrupted by a product reset, **wm_adSpaceState** function will return WM_AD_REPAIR state.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.13.15.1 Returned Values

This function returns OK if successful.
Otherwise, the function will return an error value (cf § 2.13.2 "Returned values definition").

2.13.16 The wm_adRecompact Function

The **wm_adRecompact** function performs a new recompacting step. The recompacting process has to be initialised by the **wm_adRecompactInit()** function.
Its prototype is :

```
s32 wm_adRecompact ( void );
```

2.13.16.1 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 § 2.13.2 "Returned values definition").

2.13.17 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 OS update, an Open AT® application, or an E2P configuration file).

Its prototype is :

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

2.13.17.1 Parameters

Handle
Handle on the cell to install.

2.13.17.2 Returned Values

This function resets the product and installs 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 (2.13.2 "Returned values definition").

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.14 RTC API

2.14.1 Required Header

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

2.14.2 RTC related types

2.14.2.1 The wm_rtcTime_t type

This type is the used structure by the Wavecom OS in order to retrieve the current RTC time. This type is defined below :

```
typedef struct
{
    u16  Year;            // Year (Four digits)
    u8   Month;           // Month (1-12)
    u8   Day;             // Day of the month (1-31)
    u8   Hour;            // Hour (0-23)
    u8   Minute;          // Minute (0-59)
    u8   Second;          // Second (0-59)
    u8   Pad;             // Not used
    u16 SecondFracPart;   // Second fractional part
} wm_rtcTime_t;
```

Second fractional part step matches to $1/2^{15}$ (about 30.5 µs), since the most significant bit is not used.

2.14.2.2 The wm_rtcTimeStamp_t structure

This type may be used in order to perform arithmetic operations on time data ; it is defined below :

```
typedef struct
{
    u32 TimeStamp;        // Seconds elapsed since 1st January 1970
    u16 SecondFracPart;   // Second fractional part
} wm_rtcTimeStamp_t;
```

The timestamp uses the Unix format (seconds elapsed since the 1st January 1970).

Second fractional part step matches to $1/2^{15}$ (about 30.5 µs), since the most significant bit is not used.

2.14.3 The wm_rtcGetTime Function

The wm_rtcGetTime function retrieves the current RTC time structure.

Its prototype is :

```
s32  wm_rtcGetTime ( wm_rtcTime_t * TimeStructure );
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

2.14.3.1 Parameters

TimeStructure
The returned time structure.

2.14.3.2 Return value

- OK on success.
- A negative error value if the provided time structure is set to NULL.

2.14.4 The wm_rtcConvertTime function

This function is able to convert RTC time structure to timestamp structure, and timestamp structure to RTC time structure.
Its prototype is :

```
s32  wm_rtcConvertTime ( wm_rtcTime_t *       TimeStructure,
                         wm_rtcTimeStamp_t *  TimeStamp,
                         wm_rtcConvert_e      Conversion );
```

2.14.4.1 Parameters

TimeStructure:
Input / output RTC time structure

TimeStamp:
Input / output timestamp structure

Conversion:
Conversion mode, using the type below :

```
typedef enum
{
    WM_RTC_CONVERT_TO_TIMESTAMP,
    WM_RTC_CONVERT_FROM_TIMESTAMP
} wm_rtcConvert_e;
```

WM_RTC_CONVERT_TO_TIMESTAMP
This mode allows to convert TimeStructure parameter to TimeStamp one.

WM_RTC_CONVERT_FROM_TIMESTAMP
This mode allows to convert TimeStamp parameter to TimeStructure one.

2.14.4.2 Return value

- OK on success.
- ERROR if conversion failed (internal error) or if one parameter value is incorrect.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

3 FUNCTIONING

There are three different functioning modes, depending on the type of application. They are described in the following paragraphs.

3.1 Standalone External Application

This mode corresponds to the standard operation mode: no Embedded Application is active.

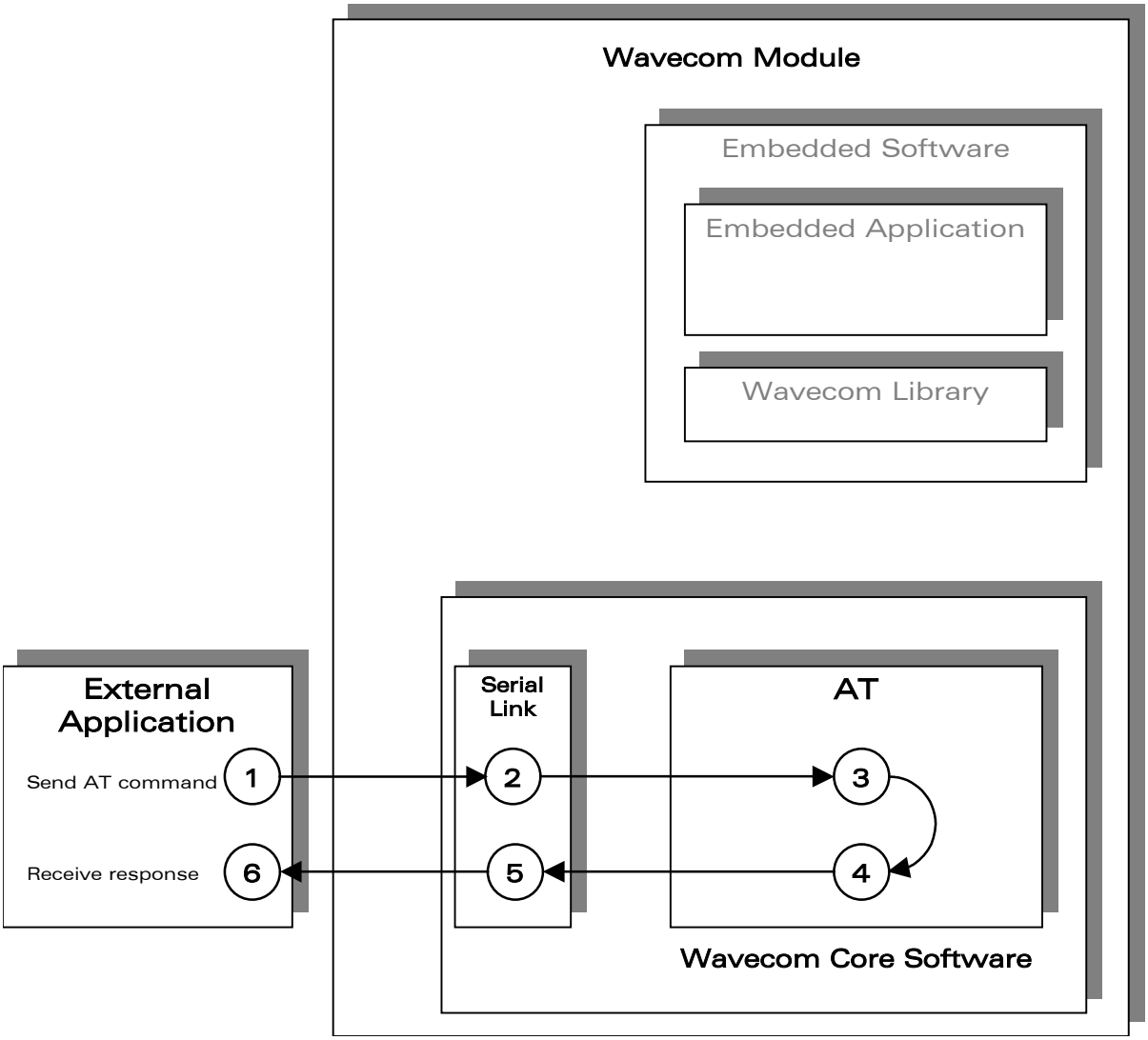


Figure 1: Standalone external application function

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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 OS,
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 § 3.2, where the AT function is in charge of dispatching the responses to the appropriate application.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

3.2 Embedded Application in Standalone Mode

This mode is based on an Embedded Application driving the GSM product independently.

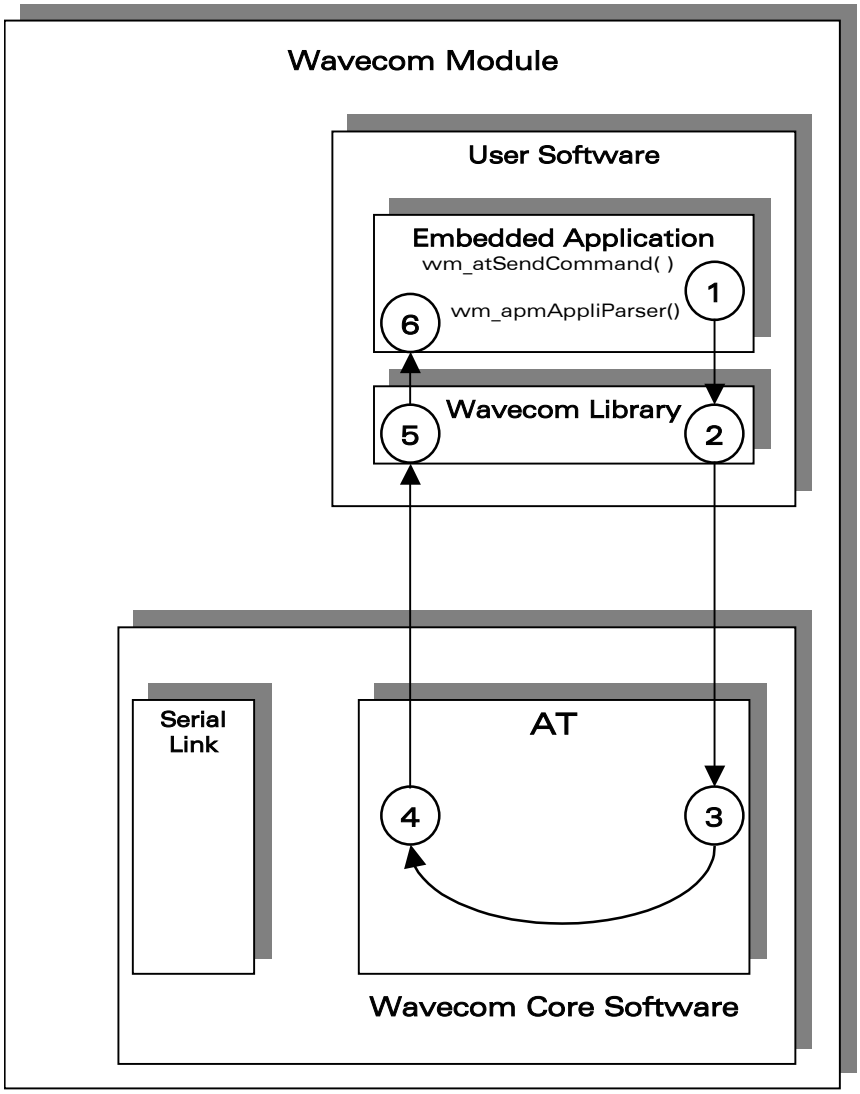


Figure 2: Embedded Application in standalone mode function

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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 OS,
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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*****************************************/
/*  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              }
};
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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>

3.3 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.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

3.3.1 Command Pre-Parsing Subscription Mechanism: WM_AT_CMD_PRE_EMBEDDED_TREATMENT

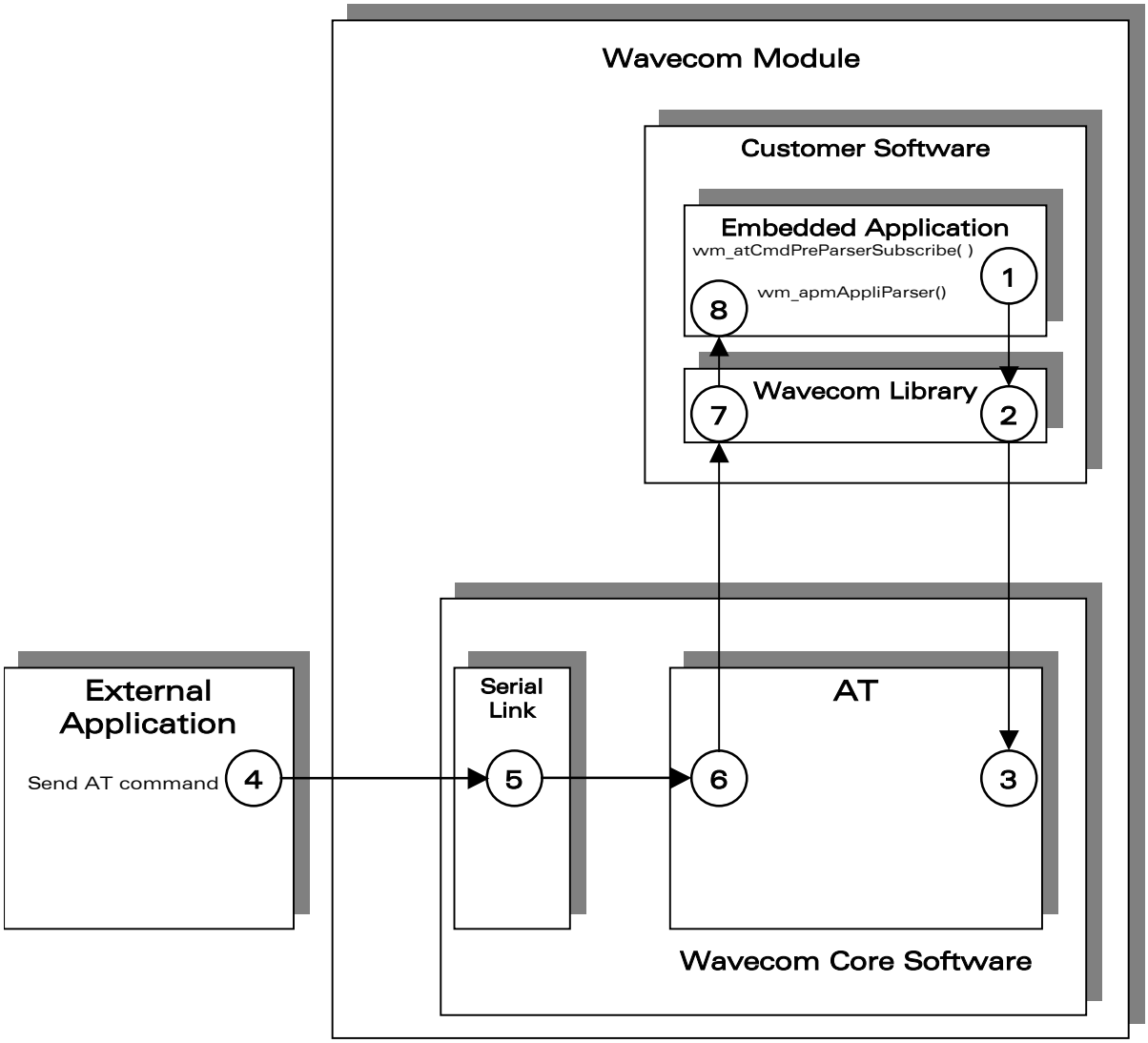


Figure 3: WM_AT_CMD_PRE_EMBEDDED_TREATMENT

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 OS,
3) The AT function sets the subscription.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

The steps in AT command processing 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 in the Wavecom OS,
3) The AT function does not process the command but transmits it to the Embedded Application,
4) The AT function does not process the command but transmits it to the Embedded Application,
5) 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),
6) 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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*************************************/
/*  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 OS command" );
                    wm_atSendCommand (
                            pMessage->Body.ATCmdPreParser.StrLength,
                           WM_AT_SEND_RSP_TO_EXTERNAL,
                           pMessage->Body.ATCmdPreParser.StrData );
                }
            }
        break;
    }

    return OK;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*************************/
/*  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 OS 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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

3.3.2 Command Pre-Parsing Subscription Process: WM_AT_CMD_PRE_BROADCAST

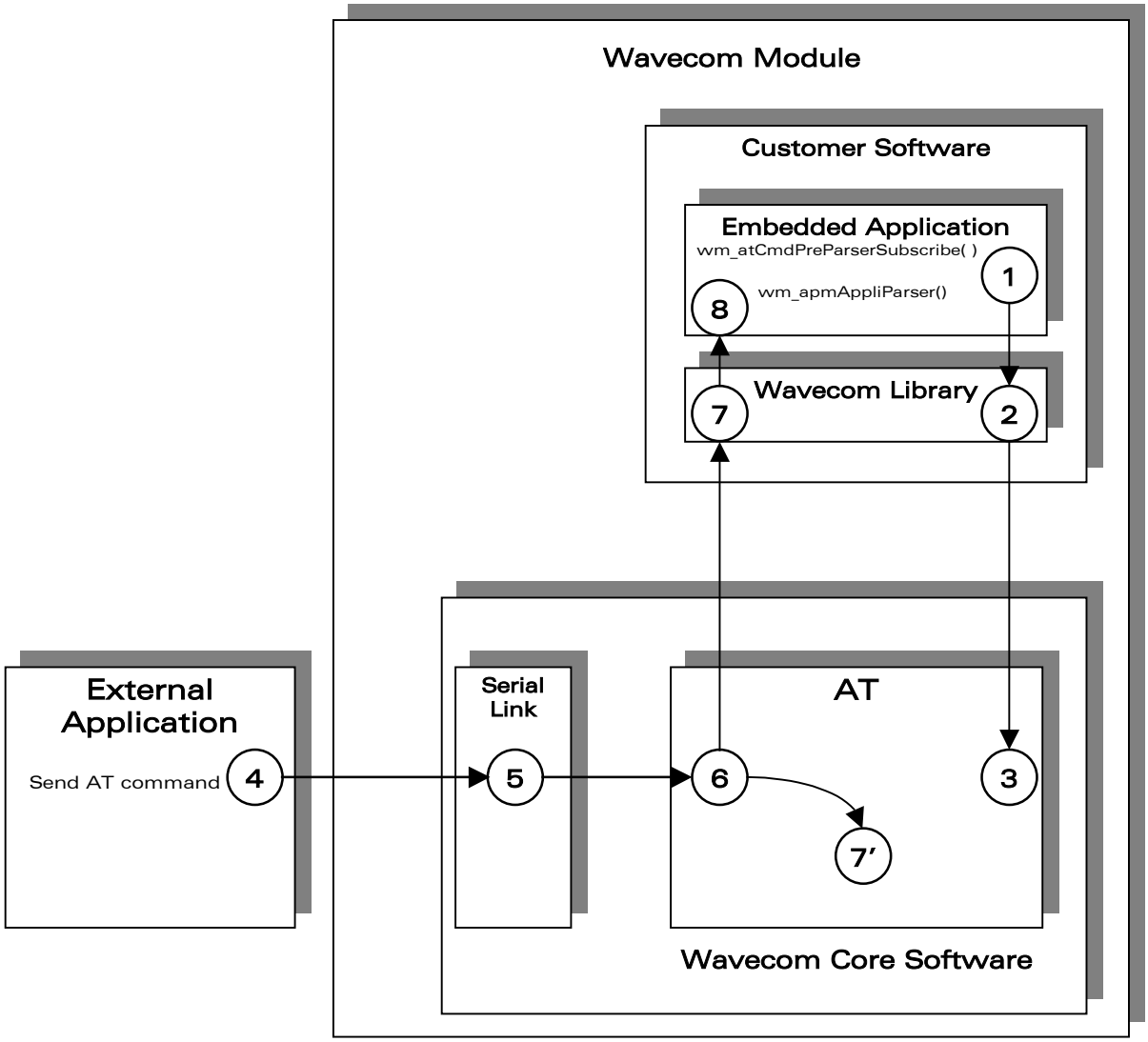


Figure 4: WM_AT_CMD_PRE_BROADCAST

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 OS, and
3) The AT function sets the subscription.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

The steps in AT command processing are performed in the following sequence:

1) The External Application sends an AT command,
2) The serial link transmits the command to the AT function of the Wavecom OS,
3) This AT function checks the subscription status of the "external" AT command,
4) 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,
5) 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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/***************************************/
/*  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
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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>

3.3.3 Response Pre-Parsing Subscription Process: WM_AT_RSP_PRE_EMBEDDED_TREATMENT

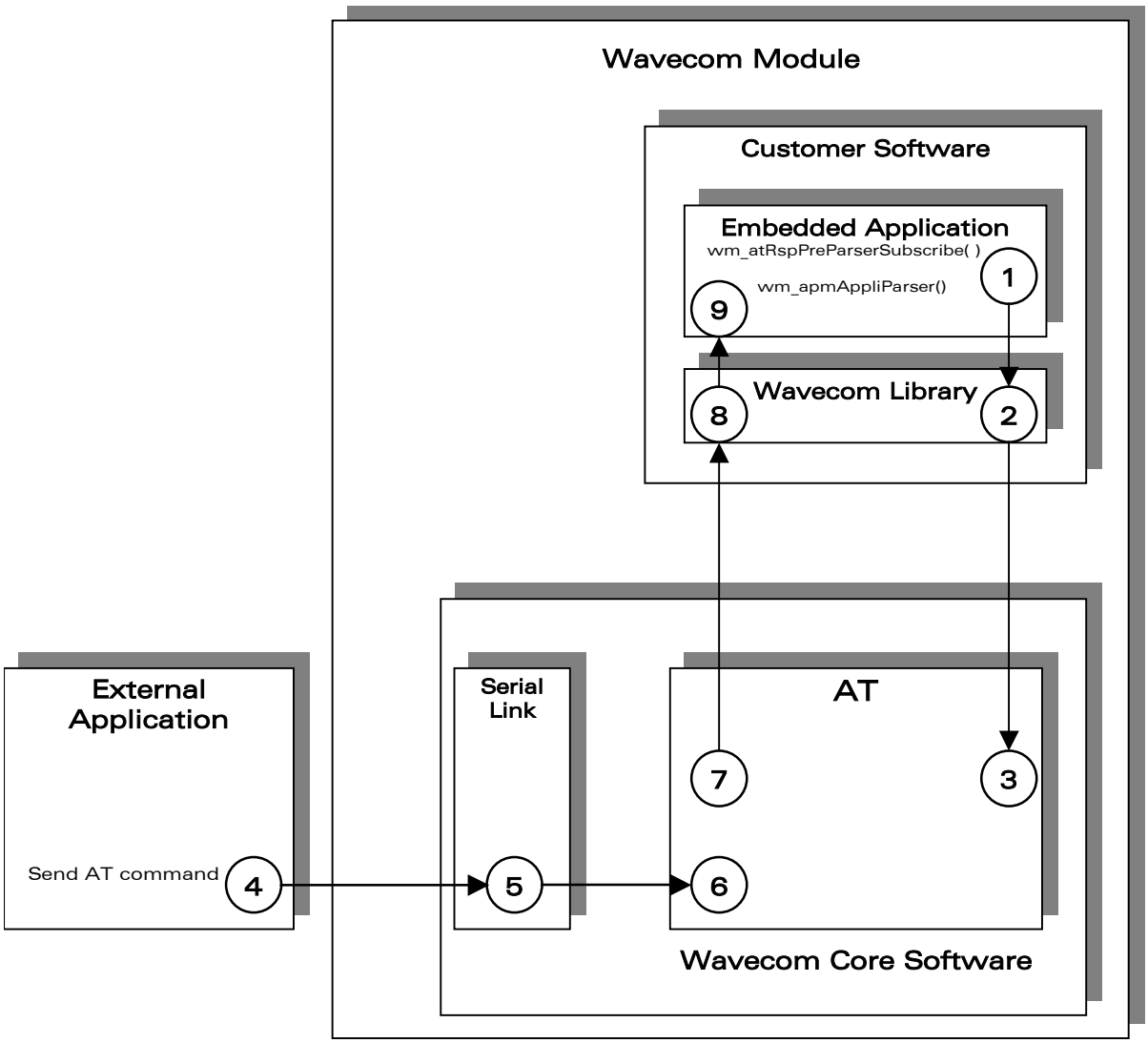


Figure 5: WM_AT_RSP_PRE_EMBEDDED_TREATMENT

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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 OS, and
3) The AT function sets the subscription.

The steps in AT command processing are performed in the following sequence:

1) The External Application sends an AT command,
2) The serial link transmits the command to the AT function of the Wavecom OS,
3) This configuration does not rely on command pre–parsing. The AT function processes the command,
4) 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,
5) 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),
6) 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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*****************************************/
/*  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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*************************/
/*  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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

3.3.4 Response Pre-Parsing Subscription Process: WM_AT_RSP_PRE_BROADCAST

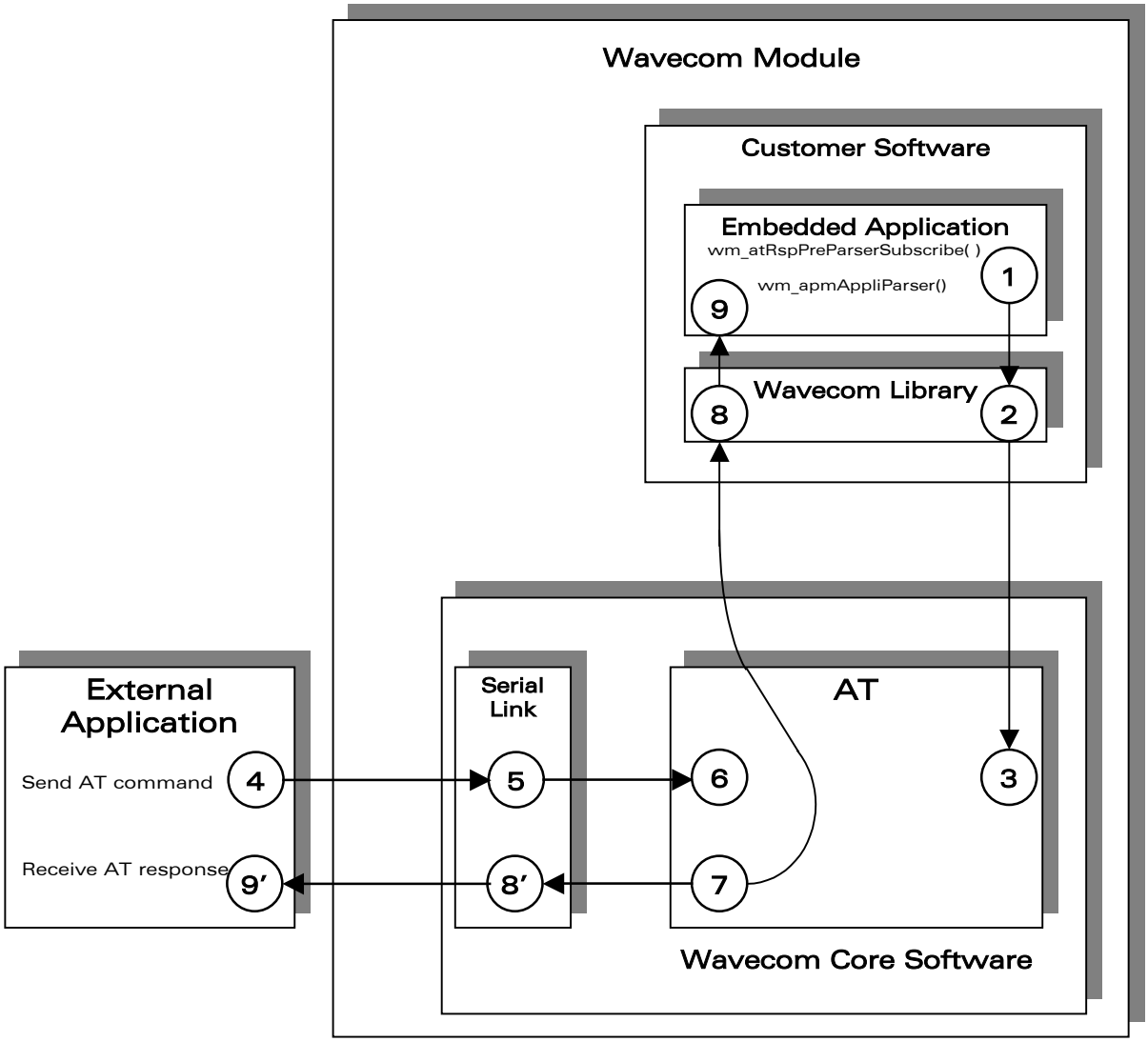


Figure 6: WM_AT_RSP_PRE_BROADCAST

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 OS, and
3) The AT function sets the subscription.

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

The steps in AT command processing are performed in the following sequence:

1) The External Application sends an AT command,
2) The serial link transmits the command to the AT function of the Wavecom OS,
3) This configuration does not rely on command pre–parsing. The AT function processes the command,
4) The AT function checks the subscription status of the response and sends it to both the External Application and the Embedded Application,
5) 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),
6) This function processes the response (the parameters of the function include a broadcast response indication),
7) This response is sent through the serial link,
8) 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;
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*****************************************/
/*  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
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

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>

3.3.5 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;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*************************/
/*  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;
  }
}
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
/*************************/
/*  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;
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

```
        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              }
};
```


This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.

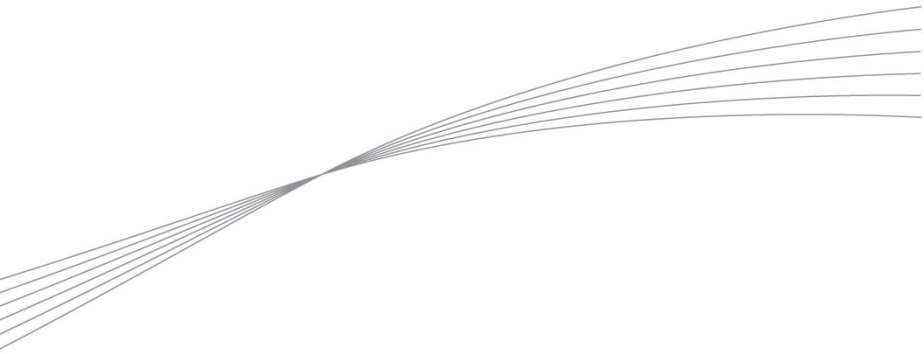
AT command log for the external application with this example:

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

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement. Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.



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. - Unit 201-207, 2nd Floor, Bio-Informatics Centre – No.2 Science Park West Avenue - Hong Kong Science Park, Shatin - New Territories, Hong Kong