Scope of the course

- **Open-AT Overview**
- Open-AT SDK Installation
- Development Tools Overview
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- Open-AT programming models
- A Snapshot on the AT Commands
- Open-AT Roadmap
- Q&A

Open-AT Overview

- What is MUSE ?
- What is Open-AT ?
- Why Open-AT ?
- Potential applications
- Open-AT SDK contents
- System requirements of Open-AT
- Hardware Resources for Open-AT applications

What is MUSE ?

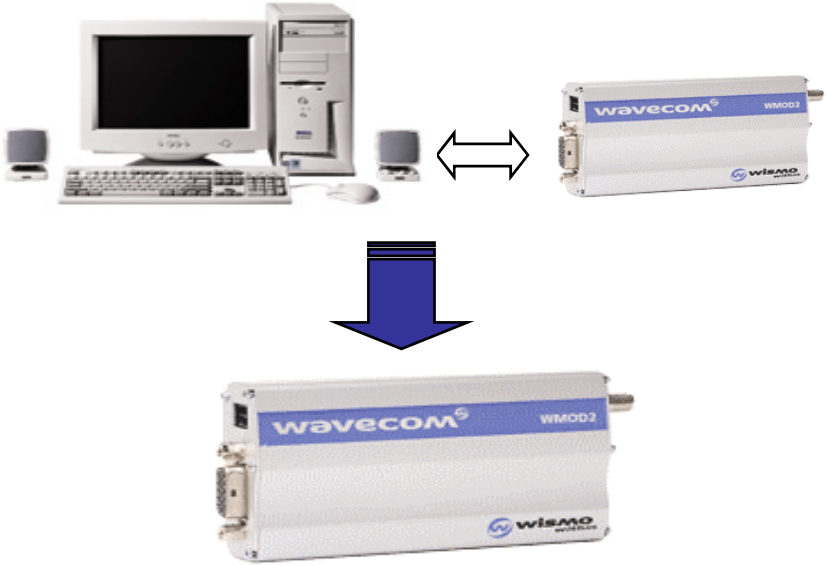
- Modular User Software Environment
  - ✓ Wavecom's software solution/platform worked together with the WISMO technology for creating unlimited new and innovative wireless applications in a rapid (time to market), cost effective (shorten development cycle) and competitive manner (lower cost, small in size)

What is MUSE ?

- Open-AT is the first product available in the MUSE platform for vertical markets (M2M)

What is Open-AT ?

- A software mechanism allows customers to embedded and run simple, standardize, AT command based applications on a WISMO module/modem



Why Open-AT ?

- Time to market
  - Reducing hardware design, software development, testing and validation process
- Saving cost
  - Reducing components and incorporating existing building blocks
- Reduce Hardware footprints
  - Increasing integration levels and avoiding redundant subsystem
- Innovative applications
  - More innovative applications become possible
- More Flexibility for developers
  - Developing your own AT commands

Potential Applications

- TELEMETRY (M2M)
  - Transmission of data/SMS on hardware event or on a regular basis for altering, monitoring or other purposes.
    - Vending machines
    - Alarm system
    - Automatic metering reporting system
    - Car safety
    - Car security
- Wireless Voice
  - Simple MMI and call management
    - Wireless Local Loop phones
    - Wireless public phones

Potential Applications

- Wireless Data/Internet
  - Providing wireless data link (GSM data/GPRS) to handheld devices for accessing Internet accessing at anytime, anywhere.
    - PDA dongle
    - Wireless Modem/fax Hub installed in a Car

Open-AT SDK contents

- Technical specifications & Documentations
- Open-AT development tools for PC:
  - target monitoring tool to provide debug information from the embedded application
  - serial link manager to establish communications with Wavecom devices via the PC serial link
  - terminal emulator to send/receive commands to/from the Wavecom device
  - remote task environment to execute an embedded application from a PC in remote mode
  - target binary file generator to generate the application binary file, ready to be downloaded onto the Wavecom device

Open-AT SDK contents

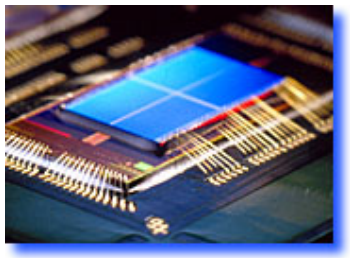
- Examples of Open-AT applications
- Technical training and support
- ARM compiler and linker version 2.51 (optional)

System requirements

- Wavecom Product
  - Open-AT Software Development Kit
  - Starter-Kit / modem including WISMO (2C2 / 2D)
- Compiler
  - ARM compiler version 2.51 (available as an option with the Open-AT SDK)
- PC
  - 300 MB Ram, 500 MB Hard Disk
  - Windows 98/2000/ME/NT
  - 1 or 2 serial ports available
- Software recommended:
  - Microsoft Visual C++ 6.0 or higher

Resources for Open-AT applications

- Memory
  - 384 KB of flash memory (for code and constant data storage)
  - 32 KB of RAM memory
  - 5 KB of configuration memory (EEPROM emulation)
- Processing Power
  - 2 MIPS when in GSM idle mode
  - 0.5 MIPS when in GSM dedicated mode (communication in progress)



Scope of the course

- Open-AT Overview
- **Open-AT SDK Installation**
- Development Tools Overview
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- Open-AT Programming models
- A Snapshot on the AT Commands
- OpenAT Roadmap
- Q&A

Before Installation

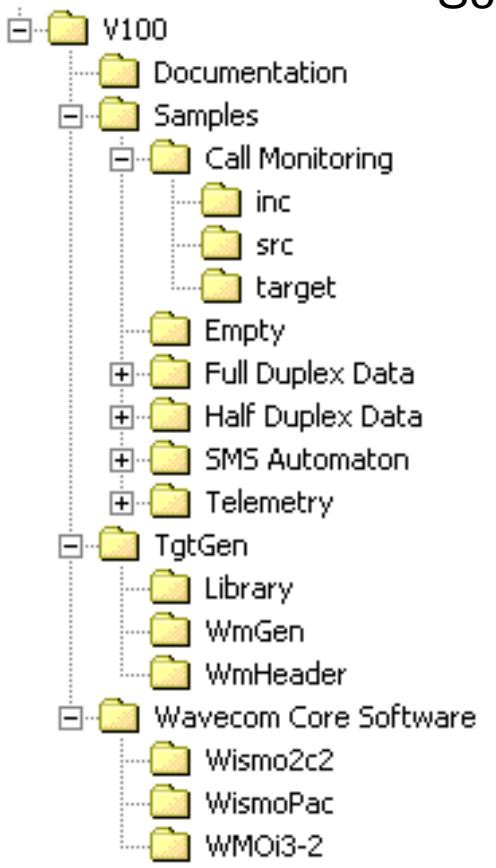
- Hardware requirement:
  - PC:
    - Pentium 300Mhz, 64MRam, 500 Mbytes free on hard disk, 2 free serial ports, CD-Rom drive
- Software Requirment:
  - OS:
    - Windows 98/2000/ME
  - Development tools:
    - Visual C++ 6.0 or above (recommended)
    - ARM compiler version V.251 (an option with the Open-AT SDK)
    - Open-AT SDK

Installation

- The details of the installation process is described in the document:
  - 'Getting Started with Open AT.pdf'
- Now, let's go through the installation process together
  - Insert the Open-AT SDK CD into the CD-Rom drive.
  - Or, launches the setup\disk1\setup.exe if auto-start is not enabled

After Installation of Open-AT

Directory structure of the Open-AT
Software installed in: c:\OpenAT

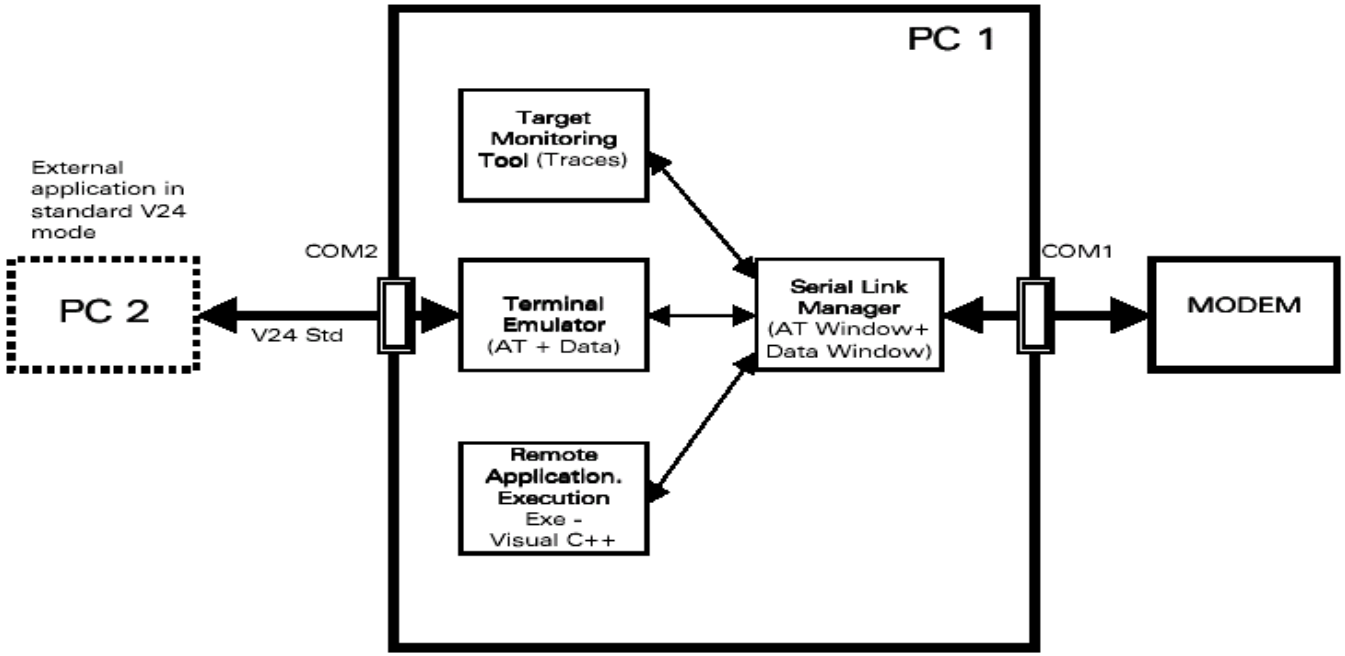


Scope of the course

- Open-AT Overview
- Open-AT SDK Installation
- **Development Tools Overview**
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- Open-AT Programming models
- A Snapshot on the AT Commands
- OpenAT Roadmap
- Q&A

Developments Tools overview

A set of 4 tools running on a PC which connects with the target by the serial link

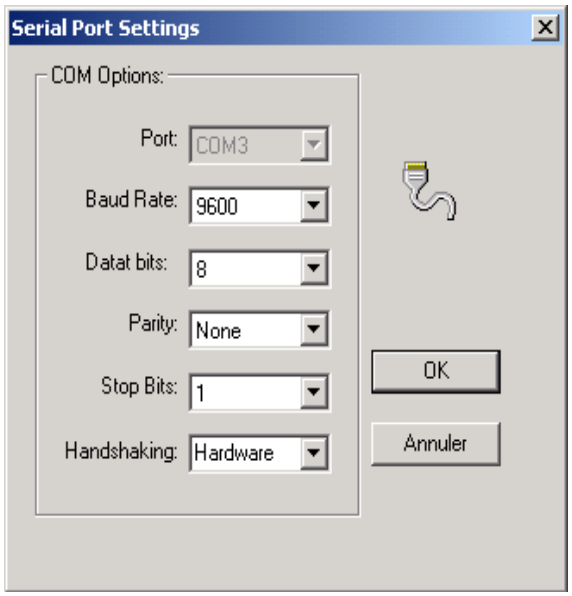


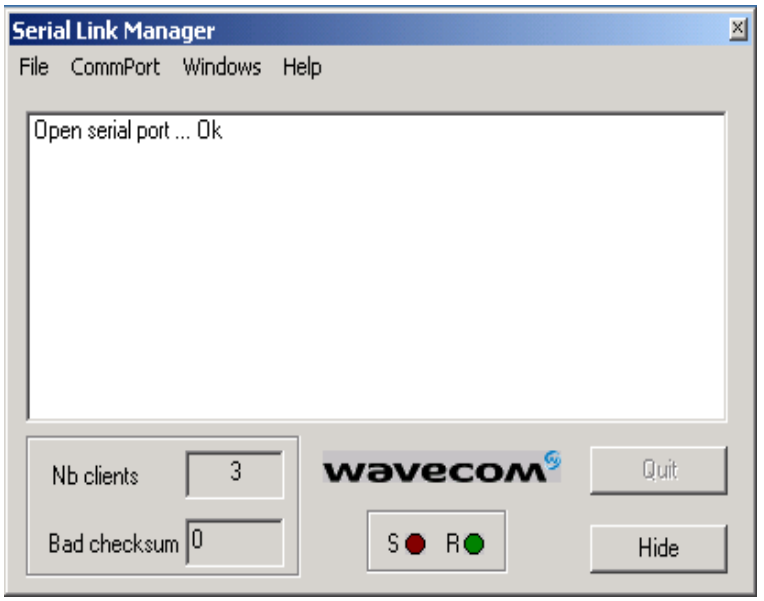
Serial Link Manager

- It acts like a data link server to other development tools.
- It manages the serial link to be shared among the development tools.
- It helps you monitor the status & configure the serial port
- It is automatically launched as long as any one of the development tools starts

Serial Link Manager

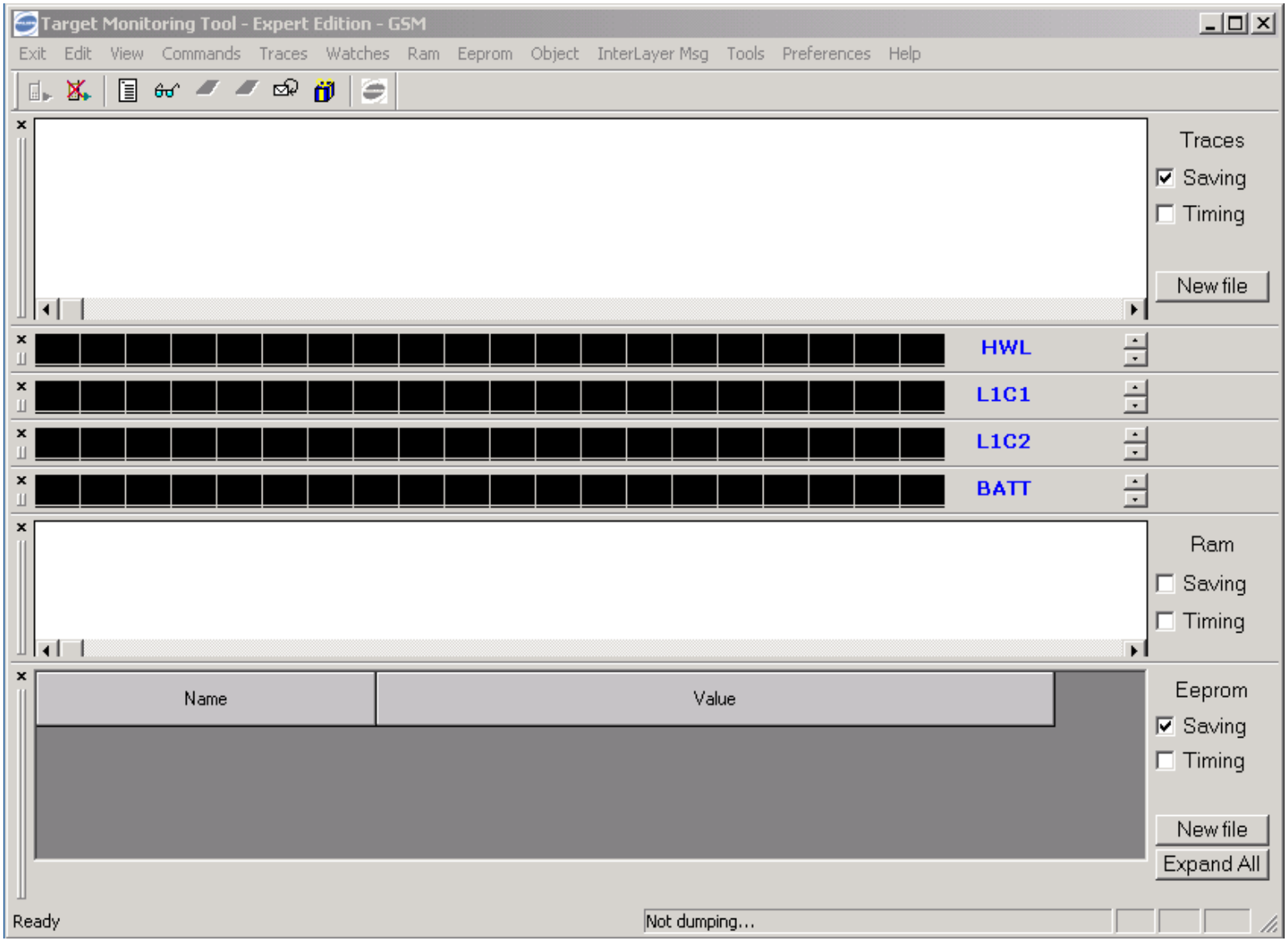
- Once it is activated, an icon is displayed on the task bar. Double-clicking on the icon, the SLM dialog will show up





Target Monitoring Tool (Moka)

- A powerful application development tool for wavecom WISMO products
- Monitor & logging down the software behaviors at run time
- Software behaviors:
  - Messages/Events between difference layers
  - Internal variables/objects in a specific task
  - User-defined debugging messages
  - Contents of the Ram and EEPROM

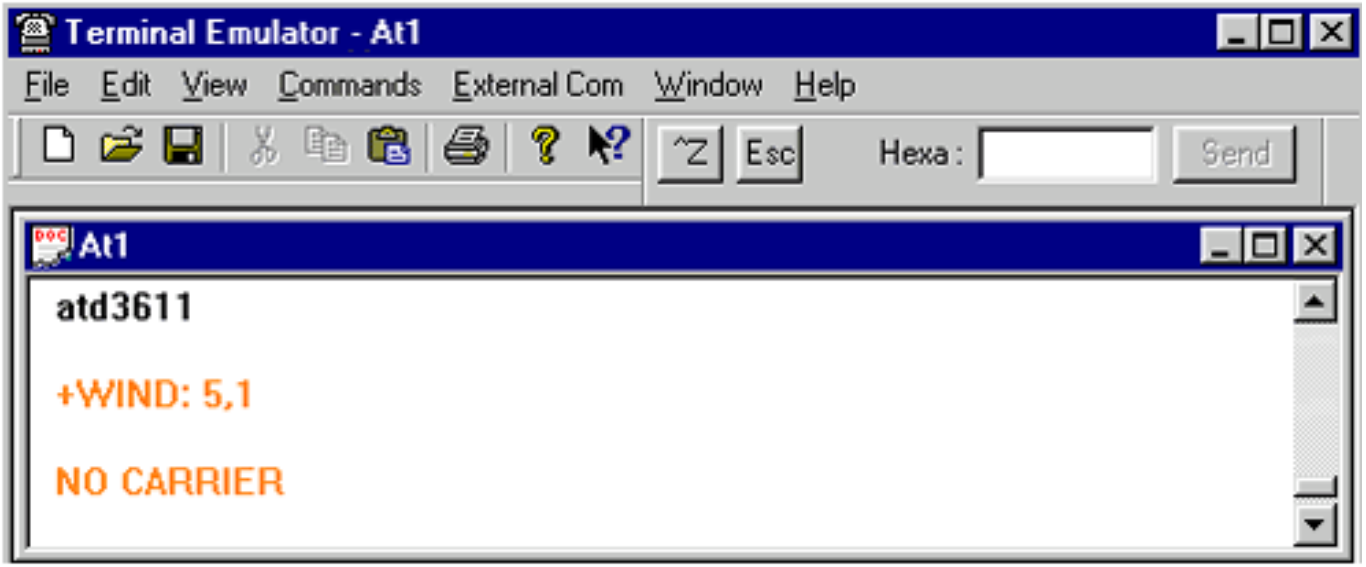

Target Monitoring Tool - Expert Edition - GSM
Exit Edit View Commands Traces Watches Ram Eeprom Object InterLayer Msg Tools Preferences Help
Traces
Saving
Timing
New file
HWL
L1C1
L1C2
BATT
Ram
Saving
Timing
Name
Value
Eeprom
Saving
Timing
New file
Expand All
Ready
Not dumping...

Terminal Emulator

- A tool for users to interactively talk to the Wavecom products via the serial link.
  - Two working modes
    - (Nominal & Wavecom Debug)
- A tool to bridge the external application to the target board
  - Converts the data flow from external application (in nominal mode to the Wavecom debug mode and vice versa
  - All the debugging information can be assessed in PC1

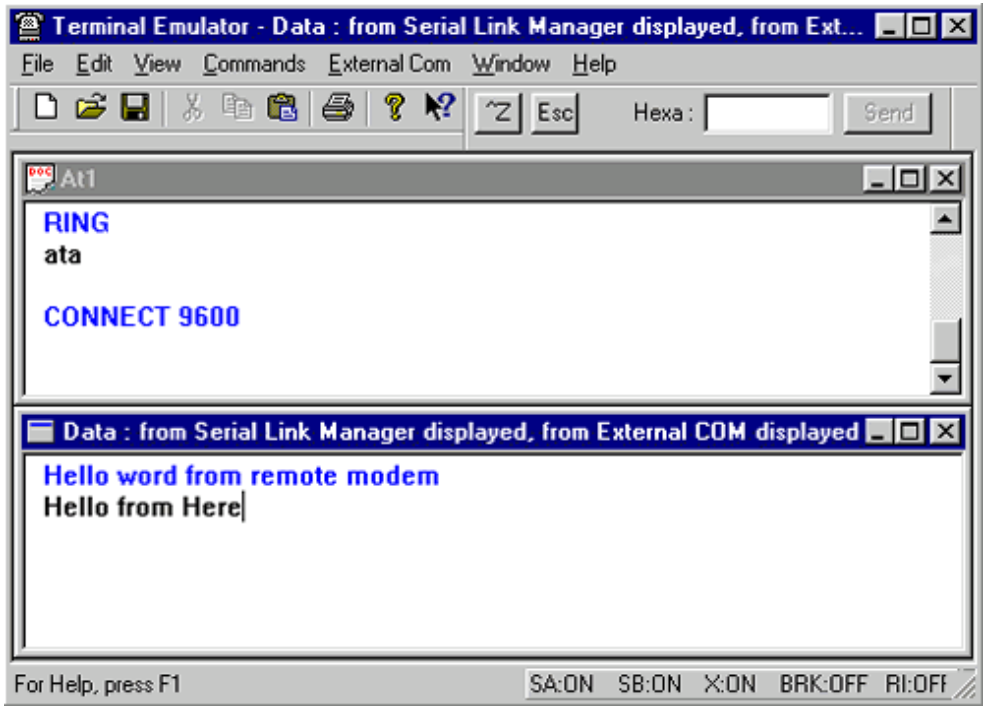
Nominal Mode

- Only AT commands and responses are available. The modem is Offline --- Disconnected
- The responses are in Orange color

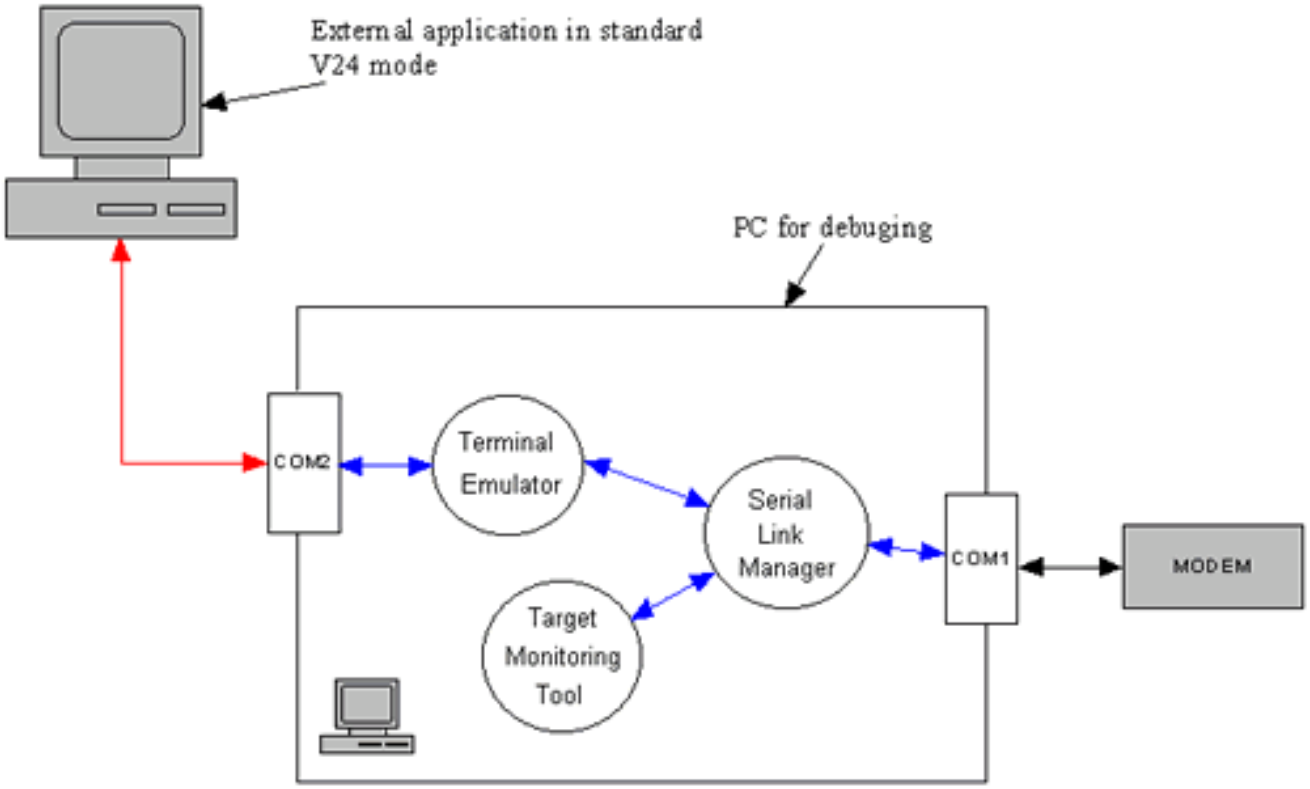


Wavecom Data Mode

- The modem is Online --- Connected
- AT commands/Responses & datas are shown in two windows



Connecting an external application



Remote Application Execution

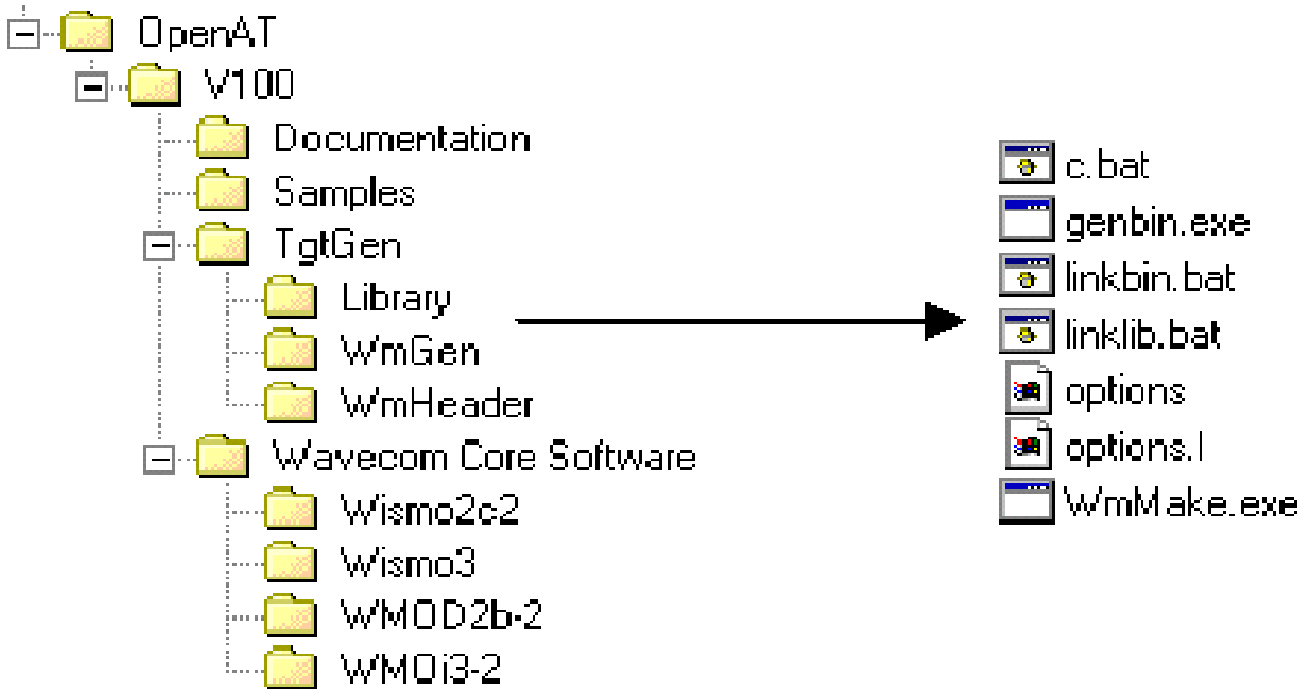
- To run the Embedded application on a PC as it is running in the target
- A special Visual C++ DevStudio's project template (wizard)

Using Remote Application Execution

- Create a project with the wizard provided by the Open-AT SDK
- Build the project as usual in Visual Studio (F7)
- Run it in the Debug mode (F5)
- The application Controller will show up. Very Simple!

Building the Target software

After installation of the Open-AT, the tools for building the target software were installed inside the ‘WmGen’ directory



Tools for Building the Target software

- **C.BAT**, used to compile C files.

  Syntax : c <name of the file without the ".c" extension>.

- **LINKLIB.BAT**, used to generate libraries.

  Syntax : linklib <name of the descriptor file without the ".lnk" extension>.

- **LINKBIN.BAT**, used to generate a binary file and convert it to an Xmodem format file, ready to be downloaded.

  Syntax : linkbin <name of the descriptor file without the ".lnk" extension>.

- **GENBIN.EXE**, required for linkbin.bat.
- **OPTIONS**, for compiler option file required for c.bat.
- **OPTIONS.L**, for linker option file required for linkbin.bat.
- **WMMAKE.EXE**, used to compile and link a library or the application binary.

  Syntax : WmMake <name of the makefile>.

Note: For the detail usage of these tools, please refer to the 'Tools Manual.pdf'

Downloader

- A tool to download the embedded software to the target board.
- Using Hyper Terminal with special AT commands and 1KXmodem / Xmodem

Downloading with the Hyper Terminal

- Launch the Hyper Terminal and connect it with the target by using the right configuration
- Use the 'at+wdwl' at command to start the downloading mode in the target
- From the 'transfer' menu, select 'send a file'
- Choose the file to be downloaded and the '1K XModem' protocol
- Press the 'send' button to start downloading...
- Reset the target by using 'at+cfun=1' command
- Run the software by using 'at+open=1' command

Note: The detail procedure is described in the section 3.3 of the 'Tutorial.pdf'

Scope of the course

- Open-AT Overview
- Open-AT SDK Installation
- Development Tools Overview
- **Application Development process**
- Open-AT technical overview
- A closer look on Open-AT APIs
- Open-AT Programming models
- A Snapshot on the AT Commands
- OpenAT Roadmap
- Q&A

Application Development process

1. Develop your embedded application software on PC (Visual Studio) with the Remote Application Execution Environment
2. Run you application on PC with the tools provided by Open-AT
3. Test (Terminal Emulator) and Debug (Moka) your software on the PC linked with the target by serial link
4. Continue the above process until the software is in good quality
5. Build the software with the ARM compiler & linker
6. Download the software to the target board for testing

Before start downloading

- Make sure that you are using the right module
- Make sure the right Wavecom Core software (version 4.32a) is being used in the module
- Make sure that the Open-AT feature is activated

Checking the right software version

- To check the software version
  - Use the AT command 'AT+CGMR'
  - Check if the result shows an software version of 4.31(for 1.0), 4.32a (for 1.1)
  - Otherwise you need to upgrade your firmware.
- 'AT+CGMR' Usage

Command syntax : AT+CGMR

Command	Possible responses
AT+CGMR *Note : Get software version*	310_G250.51 806216 032199 17:04 OK *Note : Software release 3.10, revision 51 generated on the 21st of March 1999*

Checking the activation of Open-AT option

- To check if Open-AT is activated in the module
  - Use the AT command 'AT+WSSW'
  - If the fouth digit of the last number shows a '1' as the following example, it indicates Open-AT is activated
    - Eg. A01_12gm.2C2 000100088F5DC6EA
  - Otherwise you need to contact Wavecom for the activation arrangment
- AT+WSSW Usage

Command syntax : AT+WSSW

Command	Possible responses
AT+WSSW *Note : Get Software version*	A00_00gm.2c 000000008F5DC6EA OK *Note : internal software information*

Application Development process (Practice)

- Let's go through a complete process of an Open-AT application development process.
  - Follow the section 2 of the Tutorial.pdf to generate an application with the Open-AT Wizard
  - Run the application with the remote application execution environment
  - Follow the section 3 of the Tutorial.pdf to build an embedded version without Open-AT Wizard
  - Download and execute the software on the target

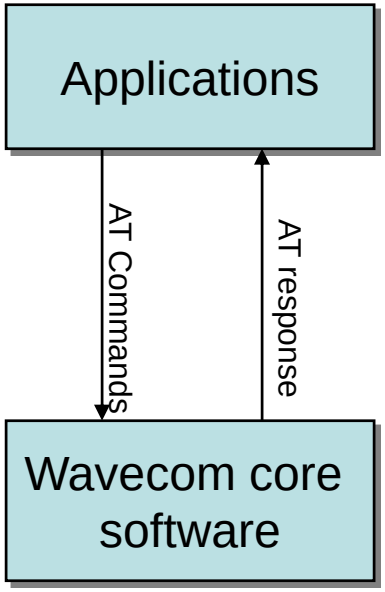
Scope of the course

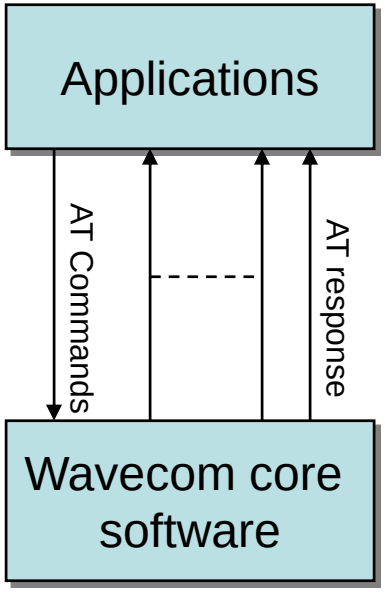
- Open-AT Overview
- Open-AT SDK Installation
- Development Tools Overview
- Application Development process
- **Open-AT technical overview**
- A closer look on Open-AT APIs
- Open-AT Programming models
- A Snapshot on the AT Commands
- OpenAT Roadmap
- Q&A

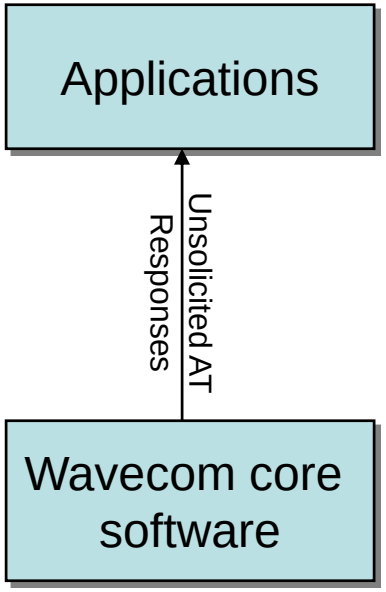
Open-AT technical overview

- Interfacing with Wavecom Modules
- Application Models
- Embedded Software Architecture
- Functional view of the Open-AT Library
- Interfacing with Open-AT
- Application skeleton

Interfacing with Wavecom modules

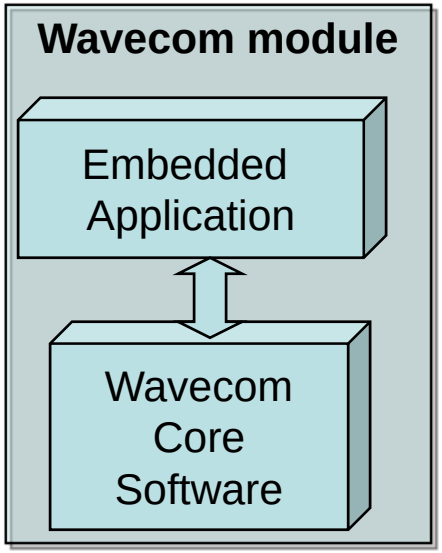




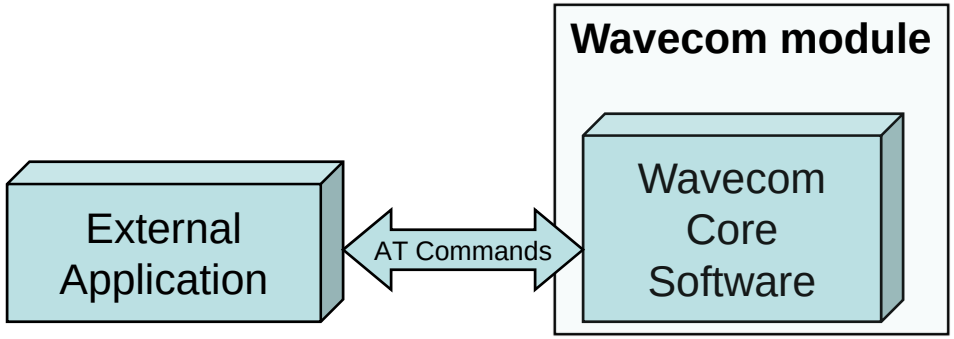


Applications models

- Embedded Standalone mode

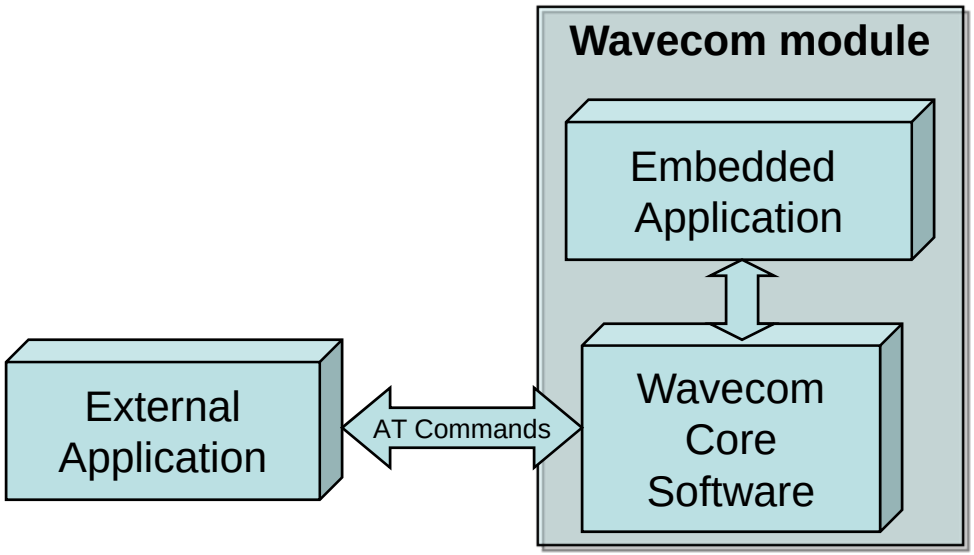


- External Standalone mode

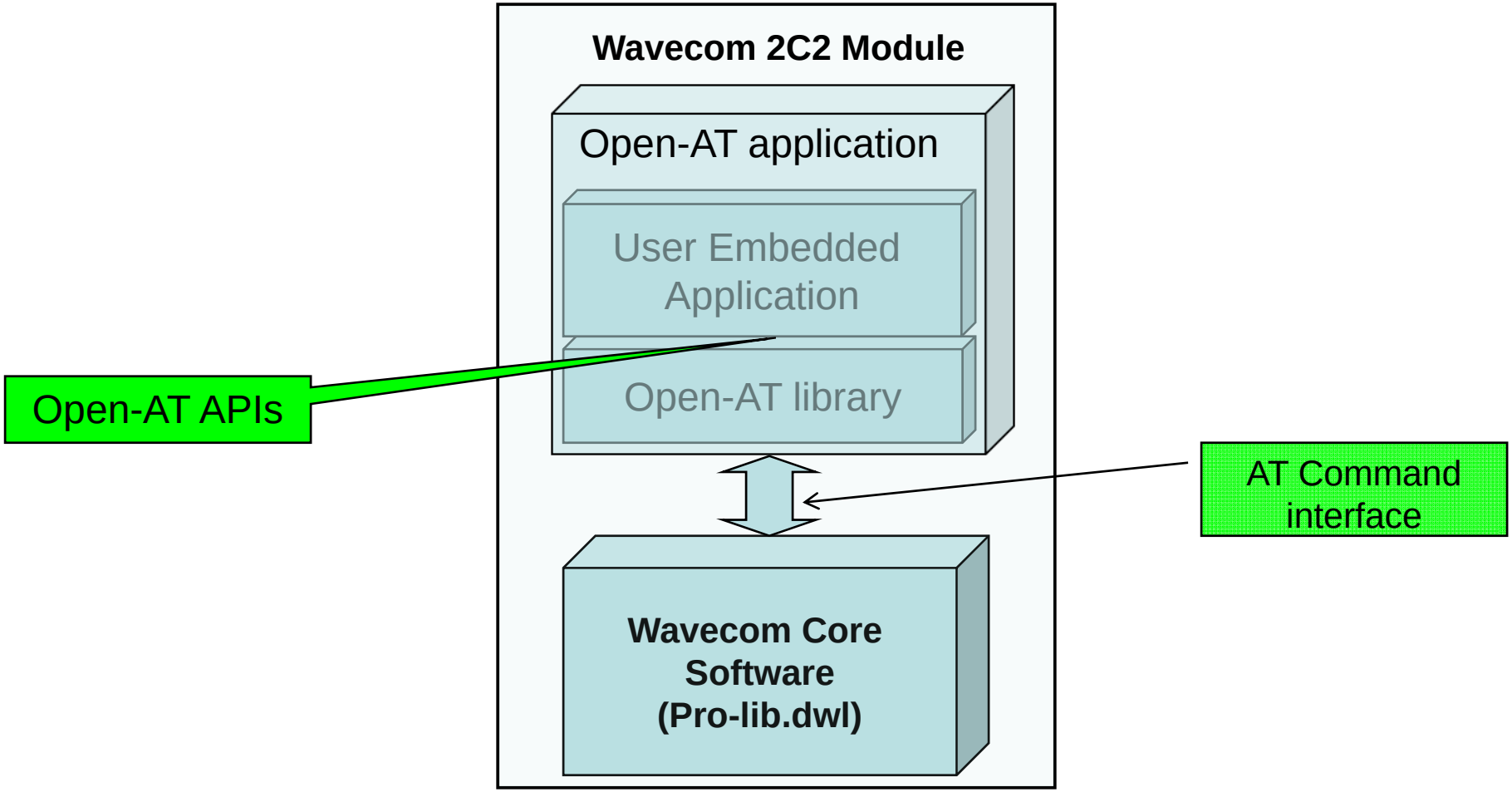


Applications models

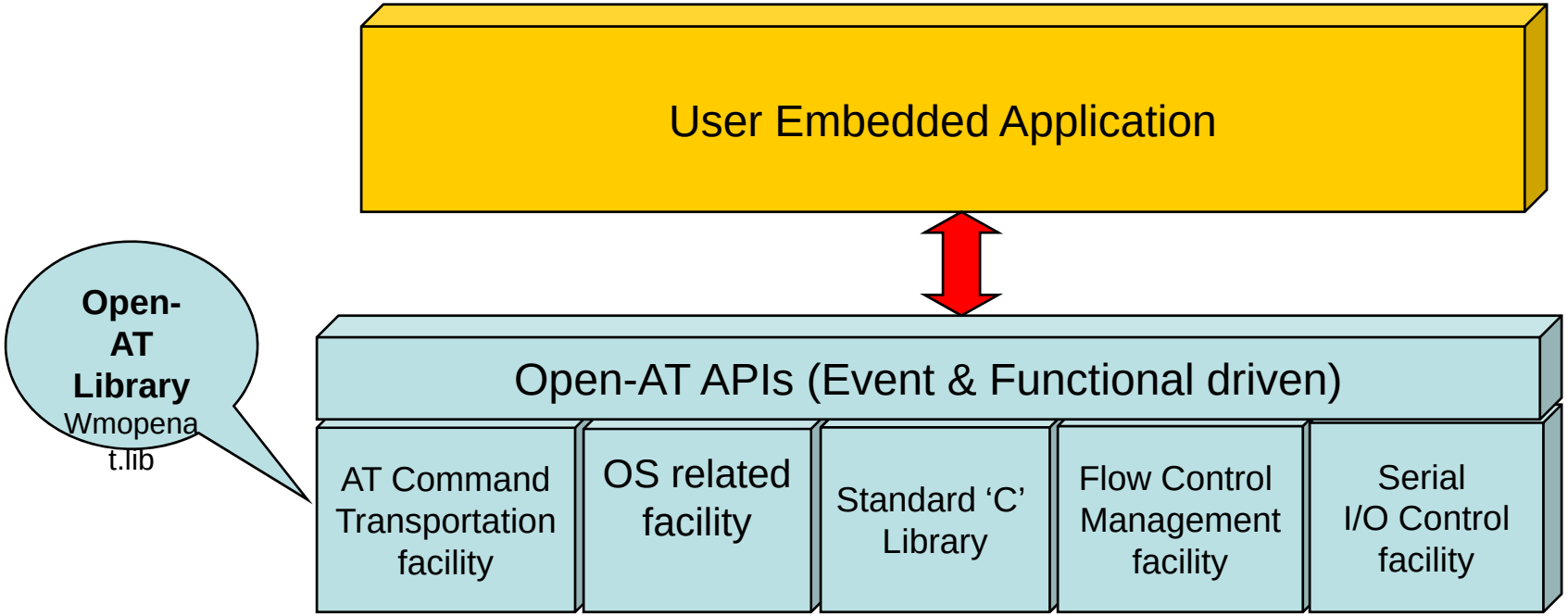
- Cooperative mode



Embedded Software Architecture



Functional view of the Open-AT



AT Command transportation facilities

- Send AT commands to Wavecom core software
- Subscribe/Filter out the AT responses from Wavecom core software
- Subscribe/Filter out the Intermediate AT responses from Wavecom core software
- Intercept the Commands/Responses from/to external applications
- Send AT responses to the External Application

AT Command transportation API

wm_atSendCommand
wm_atUnsolicitedSubscription
wm_atIntermediateSubscription
wm_atCmdPreParserSubscribe
wm_atRspPreparserSubscribe
wm_atSendRspExternalApp

OS related facilities

- Timer management
  - Start, stop, etc…
- Trace/Debugging facilities
- Non-volatile memory management
  - Read/Write Flash, etc…
- Dynamic memory allocation

OS related API Summary

Timer management	
wm_osStartTimer	wm_osStopTimer
Trace/Debugging facilities	
wm_osDebugTrace	wm_osDebugFatalError
Non-volatile memory management	
wm_osWriteFlashData	wm_osReadFlashData
wm_osGetLenFlashData	wm_osDeleteFlashData
wm_osGetAllocatedMemoryFlashData	wm_osGetFreeMemoryFlashData
Dynamic memory allocation	
wm_osGetHeapMemory	wm_osReleaseHeapMemory

Standard ‘C’ library

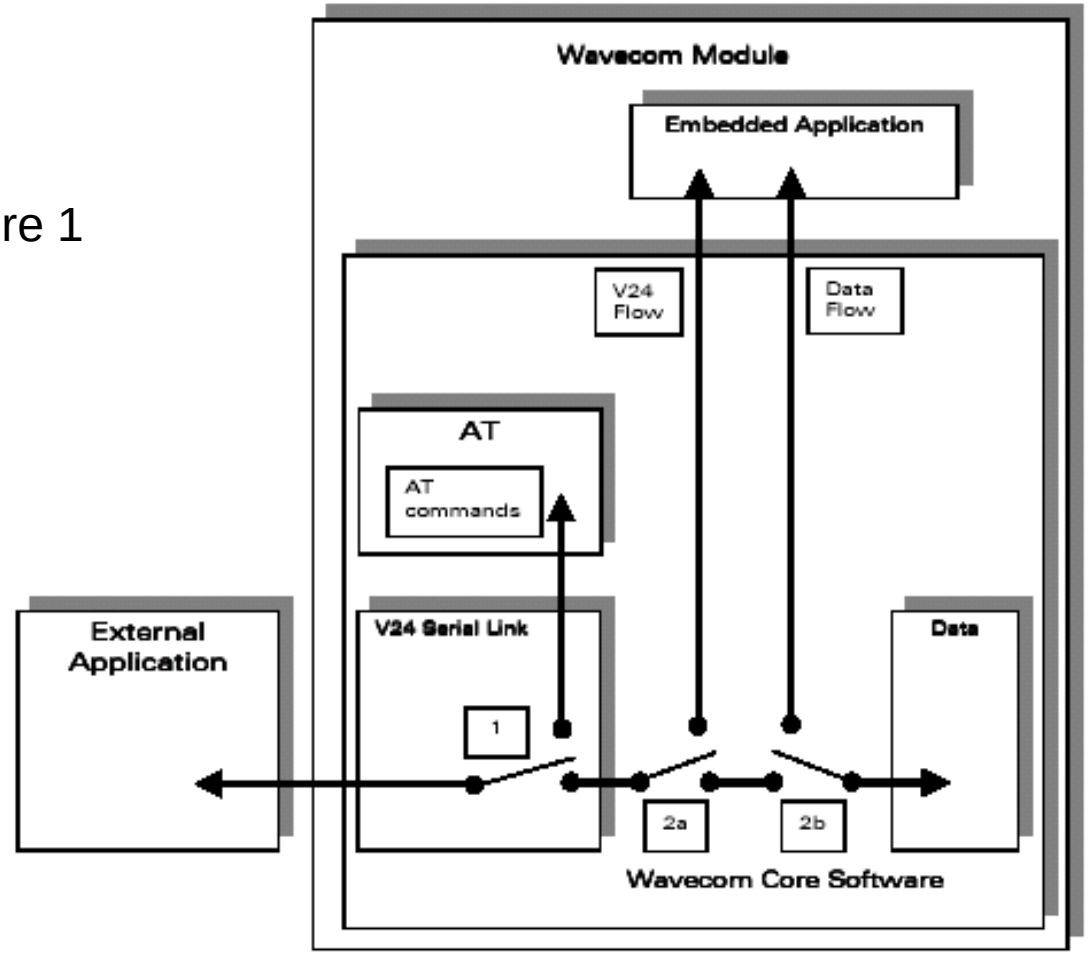
- A subset of standard ‘C’ library mainly for String manipulation
  - wm_strlen, wm_strcpy, wm_strcmp, etc…

Standard ‘C’ library API Summary

wm_strcpy	wm_strncpy
wm_strcat	wm_strncat
wm_strlen	wm_strcmp
wm_strncmp	wm_stricmp
wm_strnicmp	wm_memset
wm_memcpy	wm_memcmp
wm_itoa	wm_atoi
wm_strcmpi	
wm_isascii	wm_isdigit

Data flow switching (software control)

Figure 1



Serial I/O Control facility

- Switching the V.24 line between Data and Command
  - corresponding to the Switch 1 shown in Figure 1
  - API
    - wm_ioSerialSwitchState( )

Data Flow Control Management facility

- It provides functions for the embedded app. to
  - Alter the data path between external app. and the GSM data channel
  - Control the Switch 2a & 2b shown in Figure 1
  - Send data to the external application / GSM data channel
  - Receive data from the external application / GSM data channel
  - Control the data flow between embedded application and external app / GSM data channel

FCM API Summary

wm_fcmOpenDataAndV24
wm_fcmCloseDataAndV24
wm_fcmSubmitData
wm_apmAppliParser
wm_fcmCreditToRelease

Interfacing with Open-AT

How can we interface with the Open-AT?

- Two types of Functional APIs
  - Blocking versus Non-Blocking
- Blocking
  - Result returns immediately
- Non-Blocking
  - Event-driven (by using callback function)

Open-AT application Skeleton

```
char wm_apmCustomStack[1024];
 /* the value 1024 is an example */

const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

void wm_apmAppliInit ( wm_apminitType_e initType)
{     }

bool wm_apmAppliParse ( wm_apmMsg_t *Message)
 {
   return TRUE;
     }
```

Scope of the course

- Open-AT Overview
- Open-AT SDK Installation (Interactive)
- Development Tools Overview
- Application Development process (demo)
- Open-AT technical overview
- **A closer look on Open-AT APIs**
- Open-AT Programming models
- A Snapshot on the AT Commands
- OpenAT Roadmap
- Q&A

A closer look on Open-AT API

- Details of the Open-AT application skeleton
- AT Command transportation facilities
- OS related facilities
- Standard ‘C’ library
- Serial I/O Control facility
- Data Flow Control Management facility

Details of the Open-AT application skeleton

The mandatory part of an Open-AT application

```
char wm_apmCustomStack[1024];
 /* the value 1024 is an example */

const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

void wm_apmAppliInit ( wm_apminitType_e initType)
{     }

bool wm_apmAppliParse ( wm_apmMsg_t *Message)
 {
   return TRUE;
     }
```

The main( ) function of your application

void wm_apmAppliInit(wm_apmInitType_e initType)

```
typedef enum
{
    WM_APM_POWER_ON,
    WM_APM_REBOOT_FROM_EXCEPTION
}   wm_apmInitType_e;
```

WM_APM_POWER_ON means that normal Power On has occurred.

WM_APM_REBOOT_FROM_EXCEPTION means the module has restarted after an exception.

The following events may cause an exception:

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

The message/event handler

bool Wm_apmAppliParse(wm_apmMsg_t *Message)

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

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

MsgTyp may have the following values:

WM_AT_RESPONSE means the message includes an AT command response sent by the Embedded Application.

WM_AT_UNSOLICITED means the message includes an unsolicited AT response.

The message/event handler

WM_AT_INTERMEDIATE means the message includes an intermediate AT response.

WM_AT_CMD_PRE_PARSER means the message includes an AT command sent by the External Application.

WM_AT_RSP_PRE_PARSER means the message includes a response processed by a Wavecom Core Software AT function.

WM_OS_TIMER means the message is sent when the timer expires.

WM_OS_RELEASE_MEMORY means the message includes the address of a released pointer.

WM_FCM_RECEIVE_BLOCK means the message includes data received by the embedded application.

WM_FCM_OPEN_FLOW means the requested flow opening operation is successful.

WM_FCM_CLOSE_FLOW means the requested flow closing operation is successful.

WM_FCM_RESUME_DATA_FLOW means 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.

```
typedef union
{
    /* Includes herein the different specific structures associated to
    MsgTyp */
    /* WM_AT_RESPONSE   */
    wm_atResponse_t                 ATResponse;

    /* WM_AT_UNSOLICITED            */
    wm_atUnsolicited_t              ATUnsolicited;

    /* WM_AT_INTERMEDIATE           */
    wm_atIntermediate_t             ATIntermediate;

    /* WM_AT_CMD_PRE_PARSER         */
    wm_atCmdPreParser_t             ATCmdPreParser;

    /* WM_AT_RSP_PRE_PARSER         */
    wm_atRspPreParser_t             ATRspPreParser

    /* WM_OS_TIMER                  */
    wm_osTimer_t                    OSTimer;

    /* WM_OS_RELEASE_MEMORY         */
    wm_osRelease_t                  OSRelease;

    /* WM_FCM_RECEIVE_BLOCK          */
    wm_fcmReceiveBlock_t            FCMReceiveBlock;

    /* WM_FCM_OPEN_FLOW              */
    wm_fcmOpenFlow_t                FCMOpenFLow

    /* WM_FCM_CLOSE_FLOW             */
    wm_fcmFlow_e                    FCMCloseFlow

    /* WM_FCM_RESUME_DATA_FLOW */
    wm_fcmFlow_e                    FCMResumeFlow

    /* WM_IO_SERIAL_SWITCH_STATE_RSP */
    wm_ioSerialSwitchStateRsp_t     IOSerialSwitchStateRsp
} wm_apmBody_t;
```

The sub-structures of the message body

Body for WM_AT_RESPONSE:

```
typedef struct      {
   wm_atSendRspType_e     Type;
   u16                    StrLength;      /* Length of StrData[] */
   char                   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;
```

Body for WM_AT_UNSOLICITED:

```
typedef struct      {
   wm_atUnsolicited_e   Type;
   u16                  StrLength;
   char                 StrData[1];
} wm_atUnsolicited_t;

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

Body for WM_AT_INTERMEDIATE:

```
typedef struct      {
   wm_atIntermediate_e     Type;
   u16                     StrLength;
   char                    StrData[1];
} wm_atIntermediate_t;

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

Body for WM_AT_CMD_PRE_PARSER:

```
typedef struct      {
   wm_atCmdPreSubscribe_eType;
   u16                 StrLength;
   char                StrData[1];
} wm_atCmdPreParser_t;

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

The sub-structures of the message body

Body for WM_AT_RSP_PRE_PARSER:

```
typedef struct      {
   wm_atRspPreSubscribe_e  Type;
   u16                     StrLength;
   char                    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;
```

Body for WM_OS_TIMER:

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

Body for WM_FCM_RECEIVE_BLOCK:

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

typedef enum        {
   WM_FCM_DATA,
   WM_FCM_V24
} wm_fcmFlow_e;
```

Body for WM_OS_RELEASE_MEMORY:

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

The sub-structures of the message body

Body for WM_FCM_OPEN_FLOW:

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

typedef enum        {
   WM_FCM_DATA,
   WM_FCM_V24
} wm_fcmFlow_e;
```

Body for WM_FCM_RESUME_DATA_FLOW:

```
typedef enum        {
   WM_FCM_DATA,
   WM_FCM_V24
} wm_fcmFlow_e;
```

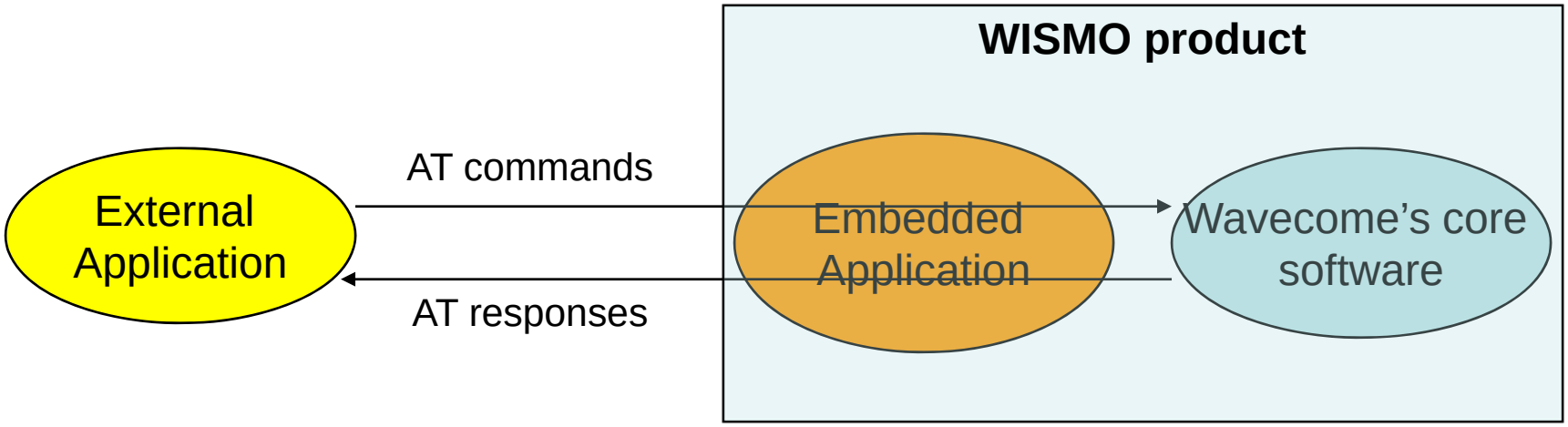
Body for WM_FCM_CLOSE_FLOW:

```
typedef enum        {
   WM_FCM_DATA,
   WM_FCM_V24
} wm_fcmFlow_e;
```

Body for WM_IO_SERIAL_SWITCH_STATE_RSP:

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

AT Command transportation facilities



Notes: The external & Embedded application are optional

AT Command transportation facilities

- Send AT commands to Wavecom core software
- Subscribe/Filter out the Unsolicited AT responses from Wavecom core software
- Subscribe/Filter out the Intermediate AT responses from Wavecom core software
- Intercept the Commands/Responses from/to external applications
- Send AT responses to the External Application
- Header: wm_apm.h

wm_atSendCommand
wm_atUnsolicitedSubscription
wm_atIntermediateSubscription
wm_atCmdPreParserSubscribe
wm_atRspPreparserSubscribe
wm_atSendRspExternalApp

Sending AT commands

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

AtString:

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

AtStringSize:

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

ResponseType:

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

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

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

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

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

Receiving Unsolicited AT Responses

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

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).

An example of a filter subscription is given below:

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

Receiving unsolicited AT responses:

```
bool wm_apmAppliParser (wm_apmMsg_t * Message)
{
   char * strBuffer;
   int    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 (TRUE);
}
```

Receiving Intermediate AT Responses

```
void   wm_atIntermediateSubscription (
                              wm_atIntermediate_e  Intermediate );

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).

An example of a filter subscription is given below:

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

```
bool wm_apmAppliParser (wm_apmMsg_t * Message)
{
   char * strBuffer;
   int    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 (TRUE);
}
```

Accessing the external AT commands

```
void   wm_atCmdPreParserSubscribe  (
                        wm_atCmdPreSubscribe_e   SubscribeType);

typedef enum            {
    WM_AT_CMD_PRE_WAVECOM_TREATMENT, /* Default value */
    WM_AT_CMD_PRE_EMBEDDED_TREATMENT,
    WM_AT_CMD_PRE_BROADCAST
} 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.

An example of a spying subscription is given below:

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

```
bool wm_apmAppliParser (wm_apmMsg_t * Message)
{
   char * strBuffer;
   int    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 (TRUE);
}
```

Accessing the external AT responses

```
void   wm_atRspPreParserSubscribe (
                    wm_atRspPreSubscribe_e   SubscribeType );

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.

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

```
bool wm_apmAppliParser (wm_apmMsg_t * Message)
{
    char *    strBuffer;
    int     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 (TRUE);
}
```

Sending Response to External Applications

void wm_atSendRspExternalApp (u16 *AtStringSize,*
char **AtString);*

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 include "\r\n" characters at the end and/or the beginnig of the string.

AtStringSize:

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

Recap on AT transportation facility

- Provides a flexibility for dividing an AT command based application software into two portions (external and embedded)
- Filtering vs. Spying
- By default, all the commands & responses (between external App. and Wavecom software) are transparent to the embedded application

OS related facilities

- Timer management
  - Start, stop, etc…
- Trace/Debugging facilities
- Non-volatile memory management
  - Read/Write Flash, etc…
- Dynamic memory allocation
- Header: wm_os.h

Timer management	
wm_osStartTimer	wm_osStopTimer
Trace/Debugging facilities	
wm_osDebugTrace	wm_osDebugFatalError
Non-volatile memory management	
wm_osWriteFlashData	wm_osReadFlashData
wm_osGetLenFlashData	wm_osDeleteFlashData
wm_osGetAllocatedMemoryFlashData	wm_osGetFreeMemoryFlashData
Dynamic memory allocation	
wm_osGetHeapMemory	wm_osReleaseHeapMemory

Timer management

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

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**: in case the timer has only one cycle.

TimerValue:

Timer unity:100 ms.

The return parameter is TRUE if the timer is set up and FALSE if not.

Timer management

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

```
bool wm_apmAppliParser (wm_apmMsg_t * Message)
{
   char *   strBuffer;
   int    nLenBuffer;

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

      ...
   }
   return (TRUE);
}
```

bool wm_osStopTimer (u8 *TimerId*);

TimerId:

Timer identifier: the range 0 to WM_OS_MAX_TIMER_ID is accepted.

The return parameter is TRUE if the timer was still running and FALSE otherwise.

Debugging facilities

void wm_osDebugTrace (u8 *Level*, **char** **Format*, ... **);**

Level:

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

Format:

Used to specify a string and the corresponding formats (like the printf function), as far as the data to trace is concerned. The supported formats are 'c', 'x', 'X', 'u', 'd'.
Up to 6 parameters may be included in the *Format* string.
As the 's' format is not supported, the way to display a char * string is to replace the *Format* string by this char, without any parameters.

...:

Represents the list of data to be traced.

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" */
```

An example of tracing an informational message using a decimal parameter is given below:

```
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" */
```

An example of tracing a string is given below:

```
char 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" */
```

Debugging facilities

The wm_osDebugFatalError function is the fatal error function: it stores the error code and then performs a reboot.

Its prototype is:

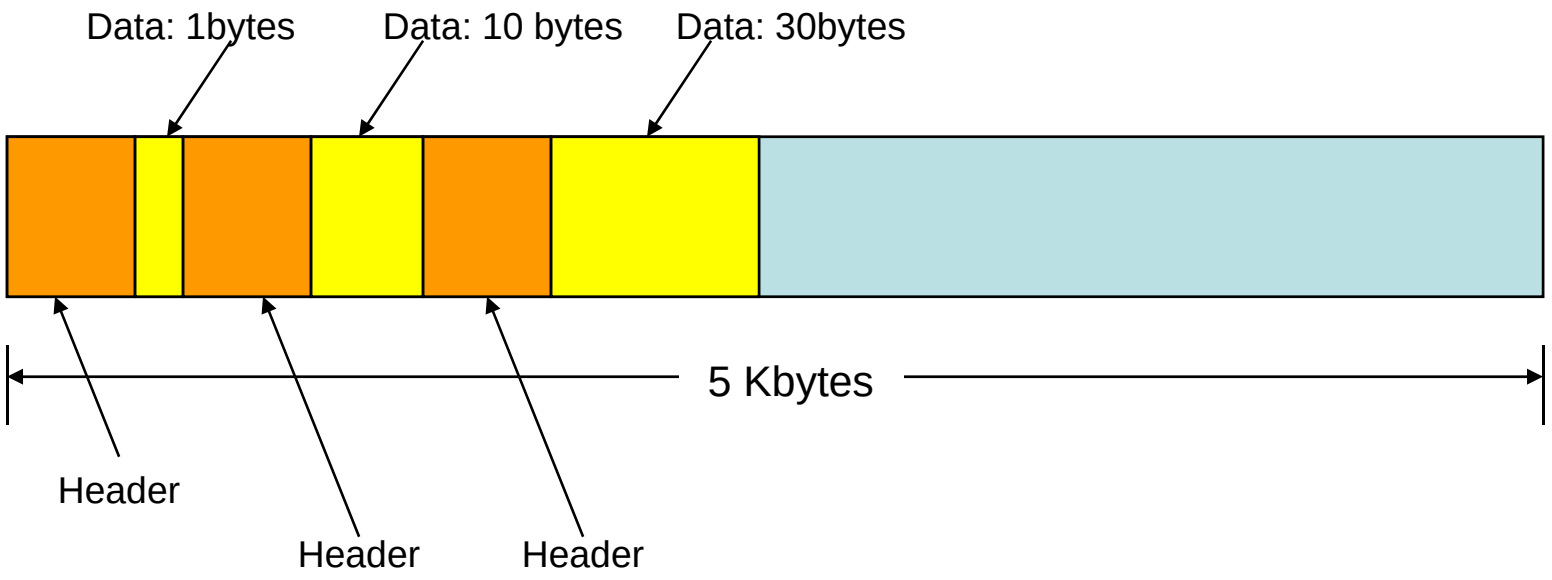
void Osw_DebugFatalError (char **Message*);

Message:

String to be displayed whenever an error occurs.

Non-volatile memory management

- Total size : 5 KB
- Each memory block contains a 10 bytes header
- Each block is identified by an unique 16bitys ‘id’
- ‘id’ can be any value except 0xFFFF



Non-volatile memory management

bool wm_osWriteFlashData (u16 *Id*, **u16** *DataLen*, **u8** **Data* **);**

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.

The return parameter is TRUE if data has been written and FALSE if not.

Non-volatile memory management

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

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.

The return parameter is the length to be read and copied to **Data*.

s32 wm_osGetLenFlashData (u16 *Id*);

Id:

Identifier assigned to the stored data.

The return parameter is the byte length of the data identified by Id. If it is negative, an error has occurred.

Non-volatile memory management

bool wm_osDeleteFlashData (u16 *Id*);

Id:

Identifier assigned to the stored data.

The return parameter is TRUE if the data have been deleted and FALSE if not.

u16 wm_osGetAllocatedMemoryFlashData (void);

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

u16 wm_osGetFreeMemoryFlashData (void);

The return parameter is the quantity of free memory in Flash ROM.

Non-volatile memory management

```
u16 LengthRead;
s32 Length;
u8* ptr;
u16 Id;
bool Writen;

FlashId = 112;

/* Get the len */
Length = wm_osGetLenFlashData (FlashId);
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);
```

Dynamic memory allocation

void *wm_osGetHeapMemory (u16 *MemorySize*);

MemorySize:

Requested size.

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

bool wm_osReleaseHeapMemory (void * *ptrData*);

PtrData:

Points to the reserved memory.

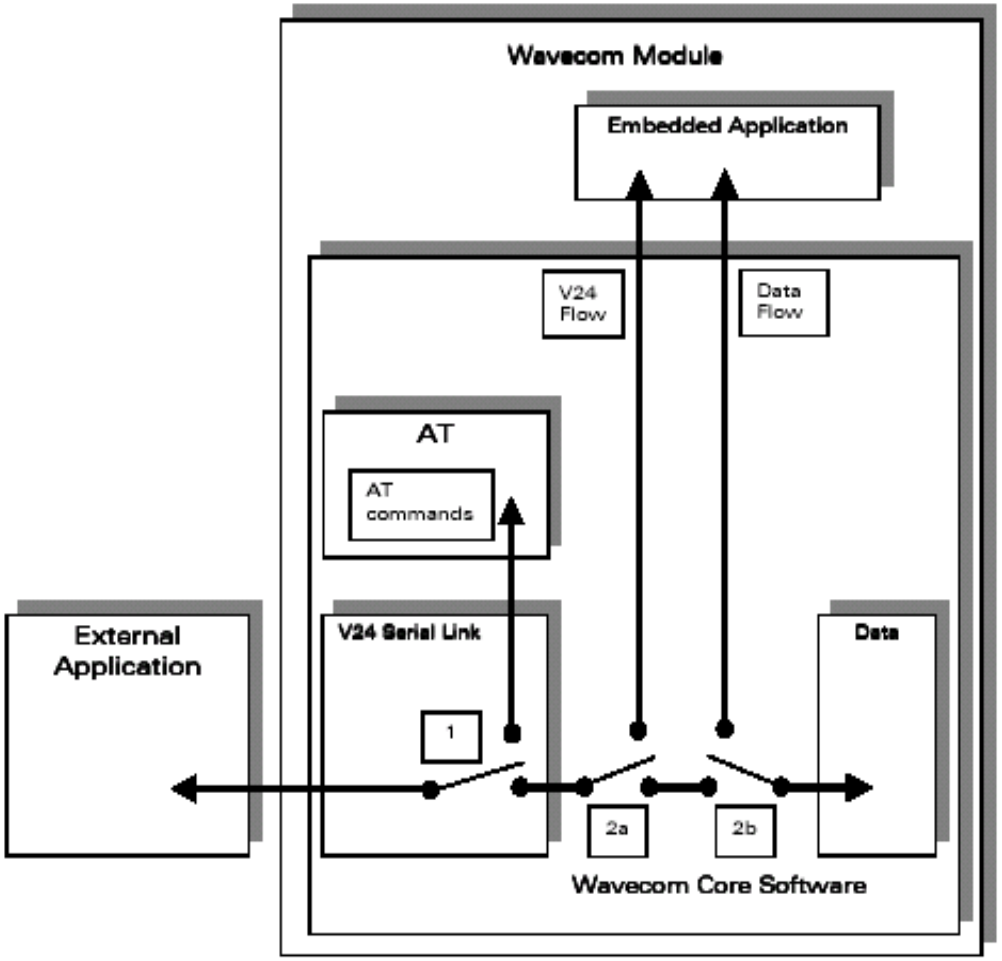
The return parameter is TRUE if the reserved memory has been released and FALSE if not.

Standard ‘C’ library

```
char *    wm_strcpy       ( char * dst, char * src );
char *    wm_strncpy      ( char * dst, char * src, u32 n );
char *    wm_strcat       ( char * dst, char * src );
char *    wm_strncat      ( char * dst, char * src, u32 n );
u32       wm_strlen       ( char * str );
s32       wm_strcmp       ( char * s1, char * s2 );
s32       wm_strncmp      ( char * s1, char * s2, u32 n );
s32       wm_stricmp      ( char * s1, char * s2 );
s32       wm_strnicmp     ( char * s1, char * s2, u32 n );
char *    wm_memset       ( char * dst, char c, u32 n );
char *    wm_memcpy       ( char * dst, char * src, u32 n );
s32       wm_memcmp       ( char * dst, char * src, u32 n );
char *    wm_itoa         ( s32 a, char * szBuffer );
s32       wm_atoi         ( char * p );
s32       wm_strcmpi      ( char * dst,   char * src );
s32       wm_strnicmp     ( char * first, char * last, u32 count );
char      wm_isascii      ( char c );
char      wm_isdigit      ( char c );
```

Serial I/O Control facility

- Switching the V.24 serial link between data and command
- Corresponding to the switch 1 shown next



Serial I/O Control facility

void wm_ioSerialSwitchState (wm_ioSerialSwitchState_e *SerialState* **);**

SerialState:

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

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

WM_IO_SERIAL_AT_MODE represents the AT commands computing mode. In this mode, data received from V24 serial link are 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 are 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

Data Flow Control Management facility

- It provides functions to alter the data path % external app. and GSM data channel
  - Control the Switch 2a & 2b shown in Figure 1
  - Send data to the external application / GSM data channel
  - Receive data from the external application / GSM data channel
  - Control the data flow between embedded application and external app / GSM data channel

APIs (wm_fcm.h)

wm_fcmOpenDataAndV24
wm_fcmCloseDataAndV24
wm_fcmSubmitData
wm_apmAppliParser
wm_fcmCreditToRelease

Flow Control Management facility

```
void  wm_fcmOpenDataAndV24 ( u16   DataMaxToReceiveFromData,
                             u16   DataMaxToReceiveFromV24 );
```

DataMaxToReceiveFromData:

Maximum block size to be sent to the embedded application from a Data communication. This size can not exceed **270 bytes**.

DataMaxToReceiveFromV24:

Maximum block size to be sent to the embedded application from the V24 serial link. This size can not exceed **120 bytes**.

```
void  wm_fcmCloseDataAndV24 ( void );
```

Flow Control Management facility

s8 wm_fcmSubmitData (**wm_fcmFlow_e** *Flow,*
wm_fcmSendBlock_t * *fcmDataBlock* **);**

Flow:

Specifies the IO flow where the data are sent; the possible values are:

```
typedef enum        {
    WM_FCM_DATA.
    WM_FCM_V24
} wm_fcmFlow_e;
```

WM_FCM_DATA represents the data flow of a Data Communication.

WM_FCM_V24 represents the data flow of the V24 serial link.

fcmDataBlock:

Pointer on a wm_fcmSendBlock_t structure, allocated (see § 3.4.13: "The wm_osGetHeapMemory ") and filled by the embedded application before sending. The definition of this structure is as follows:

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

Data Flow Control Management facility

Return of wm_fcmSubmitData()

WM_FCM_OK means 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 means 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 (see § 3.2.3: "The wm_apmAppliParser").

WM_FCM_ERR_NO_CREDIT means 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 (see § 3.2.3: "The wm_apmAppliParser").

WM_FCM_ERR_NO_LINK means 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 means the embedded application used an incorrect flow ID. The data block is not sent and the memory allocated for fcmDataBlock is not released.

On Receiving Data

wavecom

This is the WM_FCM_RECEIVE_BLOCK message structure:

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

DataLength:

Number of data bytes received in Data parameter from this flow. This size will not exceed DataMaxToReceiveFromData or DataMaxToReceiveFromV24 parameters (depending on the flow type) of the **wm_fcmOpenDataAndV24()** function (see § 3.5.1: "The wm_fcmOpenDataAndV24").

FlowID:

Specifies the opened IO flow from where the data are received. The possible values are:

```
typedef enum          {
    WM_FCM_DATA,
    WM_FCM_V24
} wm_fcmFlow_e;
```

WM_FCM_DATA represents the data flow of a Data Communication.

WM_FCM_V24 represents the data flow of the V24 serial link.

Data:

Data block received from the IO flow. The memory allocated for Data parameter will be released at the end of the **wm_apmAppliParser()** function (see § 3.2.3: "The wm apmAppliParser").

Data flow control

wavecom

```
s8  wm_fcmCreditToRelease ( wm_fcmFlow_e    Flow,
                            u8              Credits );
```

Flow:

Specifies the IO flow on which the Flow Control Manager may release credits. The possible values are:

```
typedef enum        {
    WM_FCM_DATA,
    WM_FCM_V24
} wm_fcmFlow_e;
```

WM_FCM_DATA represents the data flow of a data communication.

WM_FCM_V24 represents the data flow of the V24 serial link.

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 the **wm_fcmCreditToRelease()** function with the Credits parameter set to 3.

The returned value is ≥ 0 if the credits are released, otherwise it is negative (an error occurred and the credits are not released).

Scope of the course

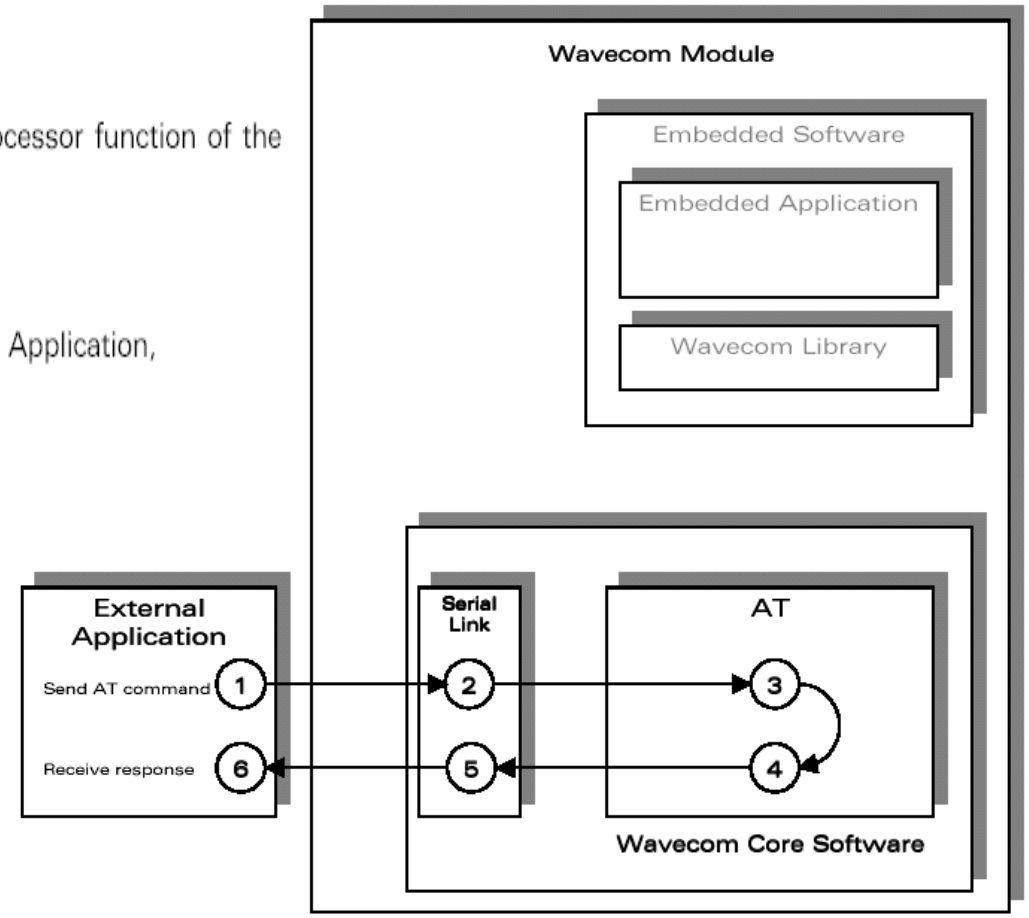
- Open-AT Overview
- Open-AT SDK Installation
- Development Tools Overview
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- **Open-AT Programming models**
- A Snapshot on the AT Commands
- Programming example and practice
- Q&A

Open-AT Programming models

- Standalone external application
- Standalone embedded application
- Cooperative mode
  - Interactive between the embedded & external applications
    - Command filtering process
    - Command spying process
    - Response filtering process
    - Response spying process

Standalone external application

1) The External Application sends an AT command,
2) The serial link transmits the command to the AT processor function of the Wavecom Core Software,
3) The AT function processes the command,
4) The AT function sends an AT response to the External Application,
5) This response is sent through the serial link, and
6) The External Application receives the response.



Standalone embedded application

1) The Embedded Application calls the "wm_atSendCommand" function to send an AT command. The response parameter is then WM_AT_SEND_RSP_TO_EMBEDDED,

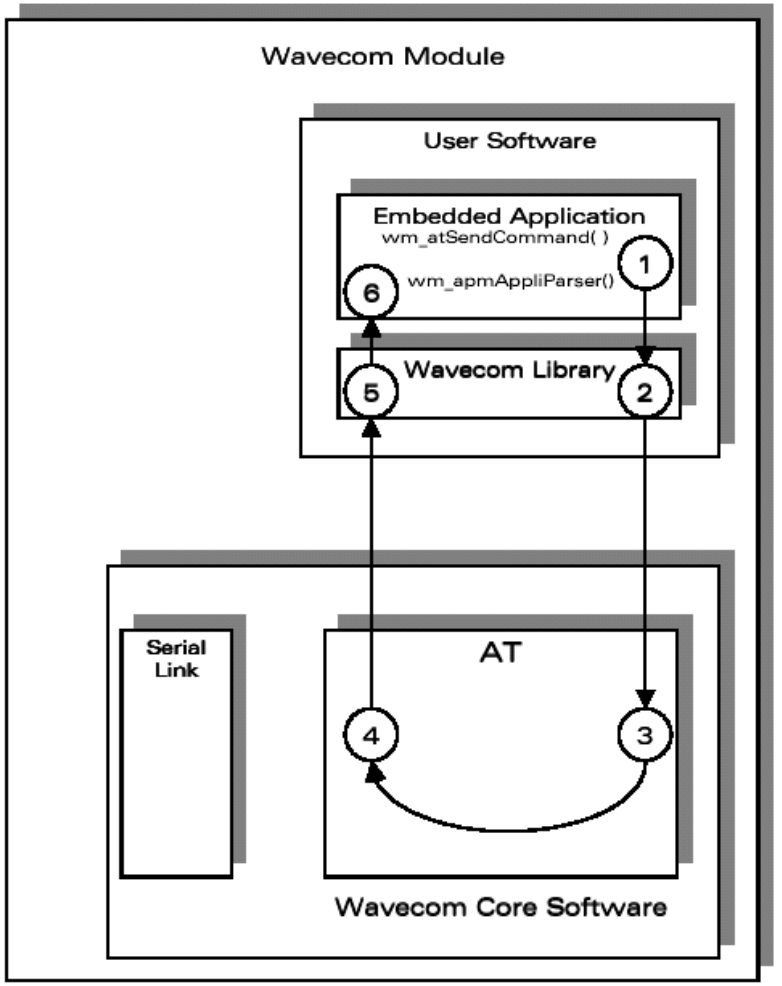
2) The Wavecom library calls the appropriate AT function from the Wavecom Core Software,

3) The AT function processes the command,

4) The AT function sends the AT response to the Embedded Application,

5) This response is dispatched by the Wavecom library which calls the "wm_apmAppliParser" function of the Embedded Application,

6) The "wm_apmAppliParser" function processes the response (the AT response is a parameter of the function). The Message type is WM_AT_RESPONSE.



Example of Standalone embedded application

```
/*************************************************/
/*  Appli.c   -  Copyright Wavecom S.A. (c) 2001    */
/*************************************************/

#include "wm_types.h"
#include "wm_apm.h"

#define TIMER 01

/*************************/
/*  Mandatory Variables     */
/*************************/

char wm_apmCustomStack[1024];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );


/*************************/
/*  Mandatory Functions    */
/*************************/

/************************************/
/*  wm_apmApplIInit                 */
/* Embedded Application initialisation */
/************************************/
void wm_apmApplIInit ( wm_apmInitType_e InitType )
{
   wm_osDebugTrace(1, "Embedded: Appli Init" );

   wm_osStartTimer ( TIMER, FALSE, WM_S_TO_TICK ( 2 ) );
}
/***************************************/
```

```
/*  wm_apmAppliParser                  */
/* Embedded Application message parser */
/***************************************/
bool 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 TRUE;
}
```

Cooperative mode (cmd filtering)

1) The Embedded Application subscribes to the command pre-parsing service, by calling the wm_atCmdPreParserSubscribe() function,

2) The Wavecom library calls the appropriate function from the Wavecom Core Software, and

3) The AT function sets the subscription.

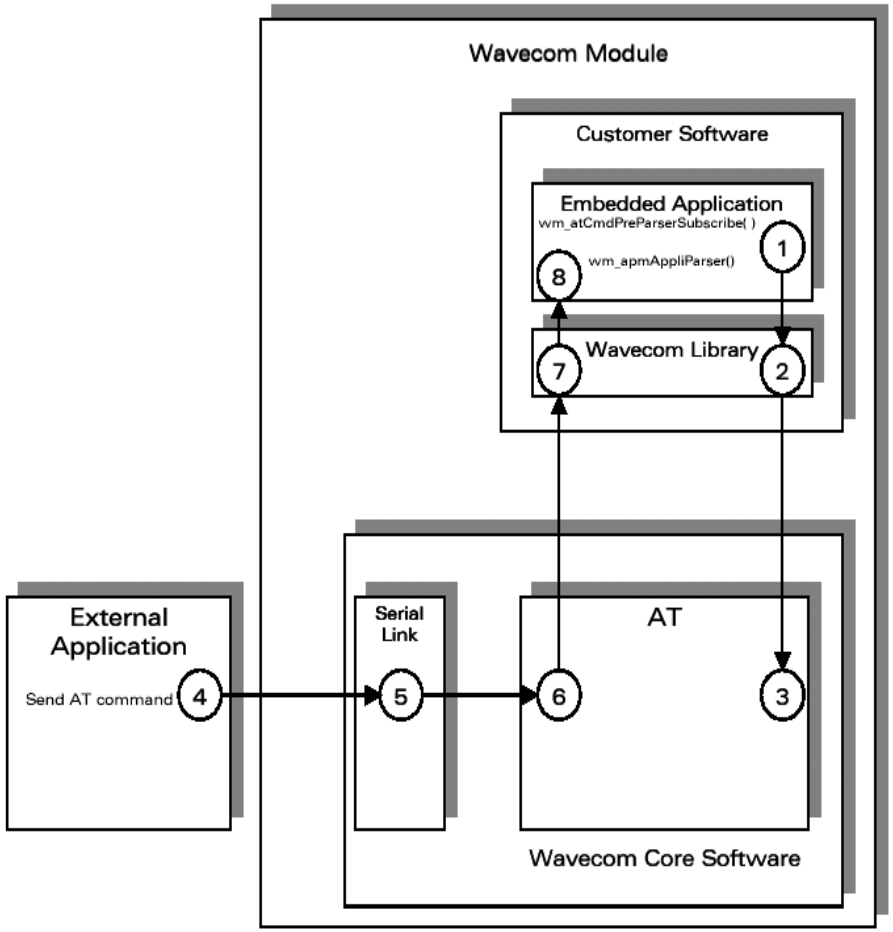
4) The External Application sends an AT command,

5) The serial link transmits the command to the AT processor function in the Wavecom Core Software,

6) The AT function does not process the command but transmits it to the Embedded Application,

7) The command is routed by the Wavecom library which calls the "wm_apmAppliParser" function of the Embedded Application (the Message type is WM_AT_CMD_PRE_PARSER),

8) This function processes the command: the parameters of the function include the AT command and an indication that the command comes from an External Application.



Example of Cooperative mode (cmd filtering)

```
/*************************************************/
/*  Appli.c   -  Copyright Wavecom S.A. (c) 2001    */
/*************************************************/

#include "wm_types.h"
#include "wm_apm.h"

#define TIMER 01

/*************************/
/*  Mandatory Variables     */
/*************************/

char wm_apmCustomStack[1024];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );


/*************************/
/*  Mandatory Functions    */
/*************************/

/************************************/
/*  wm_apmAppliInit                 */
/* Embedded Application initialisation */
/************************************/
void 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 ) );
}
```

```
bool wm_apmAppliParser ( wm_apmMsg_t * pMessage )
{
   wm_osDebugTrace ( 1, "Embedded: Appli Parser" );

   switch ( pMessage->MsgTyp )
   {
      case WM_OS_TIMER:
         wm_osDebugTrace ( 1, "WM_OS_TIMER received" );
      break;

      case WM_AT_CMD_PRE_PARSER:
         wm_osDebugTrace ( 1, "WM_AT_CMD_PRE_PARSER received" );
         if ( pMessage->Body.ATCmdPreParser.Type ==
                        WM_AT_CMD_PRE_EMBEDDED_TREATMENT )
         {
            wm_osDebugTrace ( 1, "command received:" );
            wm_osDebugTrace ( 1, pMessage->Body.ATCmdPreParser.StrData );

            if ( !wm_strncmp ( pMessage->Body.ATCmdPreParser.StrData,
                              "AT-W", 4 ) )
            {
               /* filter Specific embedded application command */
               wm_osDebugTrace ( 1, "Specific embedded application command" );

               /* send response to external application */
               wm_atSendRspExternalApp ( 10, "\r\n->WOK\r\n" );
            }
            else
            {
               /* command must be treated by AT Software */
               wm_osDebugTrace ( 1, "Wavecom Core Software command" );
               wm_atSendCommand (
                     pMessage->Body.ATCmdPreParser.StrLength,
                     WM_AT_SEND_RSP_TO_EXTERNAL,
                     pMessage->Body.ATCmdPreParser.StrData );
            }
         }
      break;
   }

   return TRUE;
}
```

Cooperative mode (cmd spying)

1) The Embedded Application subscribes to the command pre-parsing service, by calling the wm_atCmdPreParserSubscribe() function,

2) The Wavecom library calls the appropriate function in the Wavecom Core Software, and

3) The AT function sets the subscription.

4) The External Application sends an AT command,

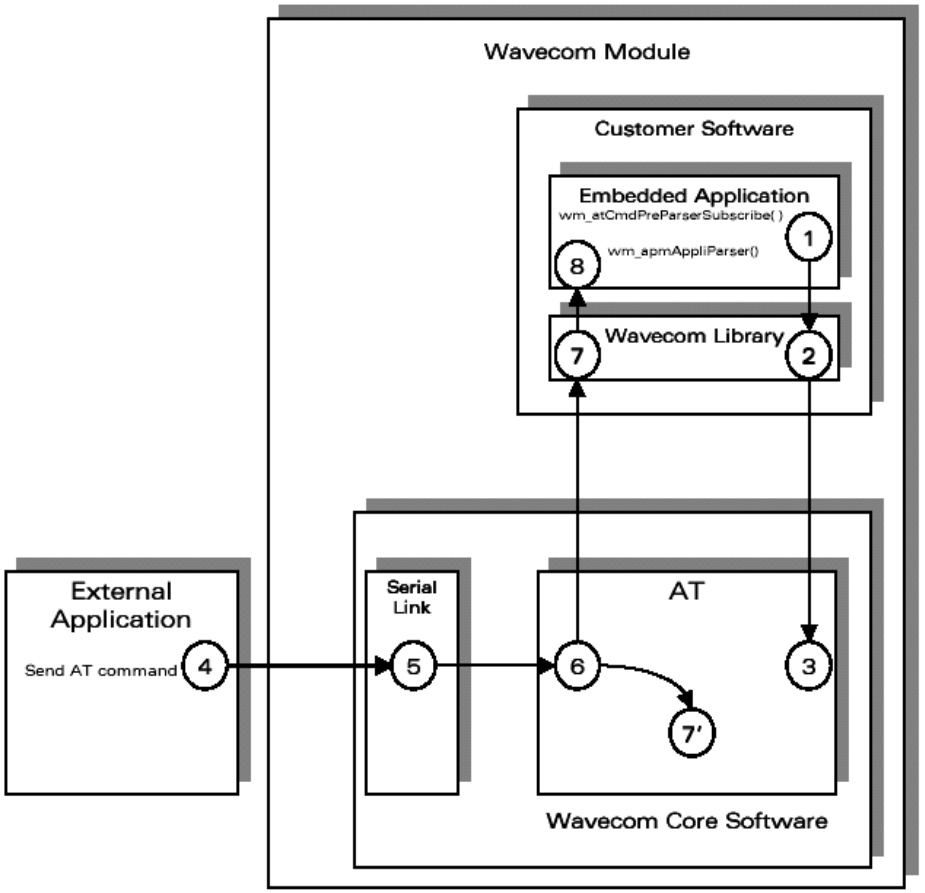
5) The serial link transmits the command to the AT function of the Wavecom Core Software,

6) This AT function checks the subscription status of the "external" AT command,

7) This external AT command is dispatched by the Wavecom library which calls the "wm_apmAppliParser" function of the Embedded Application,

7') Meanwhile, the AT function processes the command,

8) The "wm_apmAppliParser" function spies the command: the parameters include the AT command and the indication of whether or not the command is a copy (the Message type is WM_AT_CMD_PRE_PARSER).



Example of Cooperative mode (cmd spying)

```
/*************************************************/
/*  Appli.c   -  Copyright Wavecom S.A. (c) 2001    */
/*************************************************/

#include "wm_types.h"
#include "wm_apm.h"

#define TIMER 01

/***************************/
/*  Mandatory Variables     */
/***************************/

char wm_apmCustomStack[1024];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

/***************************/
/*  Mandatory Functions    */
/***************************/

/*************************************/
/*  wm_apmAppliInit                  */
/* Embedded Application initialisation */
/*************************************/
void 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 ) );
}
/*****************************************/
```

```
/*  wm_apmAppliParser                    */
/* Embedded Application message parser */
/*****************************************/
bool 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 TRUE;
}
```

Cooperative mode (Rsp filtering)

1) The Embedded Application subscribes to the response pre-parsing facility, by calling the wm_atRspPreParserSubscribe() function,

2) The Wavecom library calls the appropriate function from the Wavecom Core Software, and

3) The AT function sets the subscription.

4) The External Application sends an AT command,

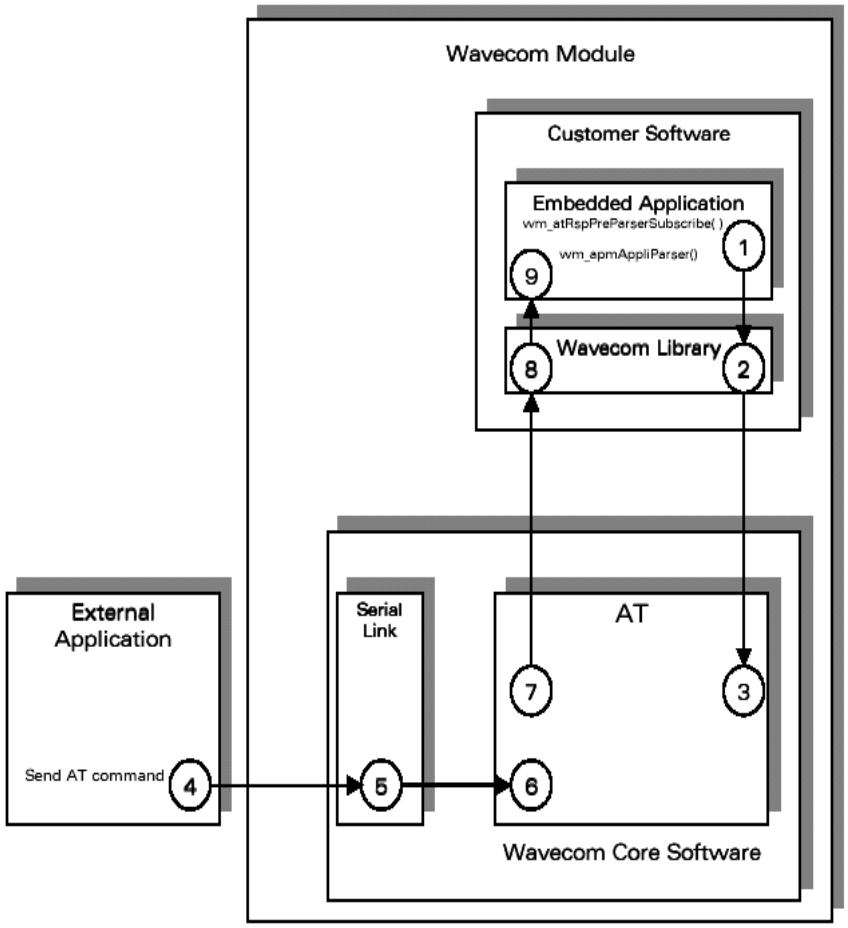
5) The serial link transmits the command to the AT function of the Wavecom Core Software,

6) This configuration does not rely on command pre-parsing. The AT function processes the command,

7) The AT function checks the subscription status of the response and does not send the response to the External Application. Instead, it sends the response to the Embedded Application,

8) The response is dispatched by the Wavecom library which calls the "wm_apmAppliParser" function of the Embedded Application (the Message type is WM_AT_RSP_PRE_PARSER),

9) This function processes the response (the parameters of the function include an indication of the response filtering).



Example of Cooperative mode (Rsp filtering)

```
/*************************************************/
/*  Appli.c   -  Copyright Wavecom S.A. (c) 2001    */
/*************************************************/

#include "wm_types.h"
#include "wm_apm.h"
#define TIMER 01

/***************************/
/*  Mandatory Variables     */
/***************************/

char wm_apmCustomStack[1024];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

/***************************/
/*  Mandatory Functions     */
/***************************/

/************************************/
/*  wm_apmAppliInit                 */
/* Embedded Application initialisation */
/************************************/
void 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 ) );
}
```

```
bool 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 TRUE;
```

Cooperative mode (Rsp spying)

1) The Embedded Application subscribes to the response pre-parsing facility, by calling the wm_atRspPreParserSubscribe() function,

2) The Wavecom library calls the appropriate function in the Wavecom Core Software, and

3) The AT function sets the subscription.

4) The External Application sends an AT command,

5) The serial link transmits the command to the AT function of the Wavecom Core Software,

6) This configuration does not rely on command pre-parsing. The AT function processes the command,

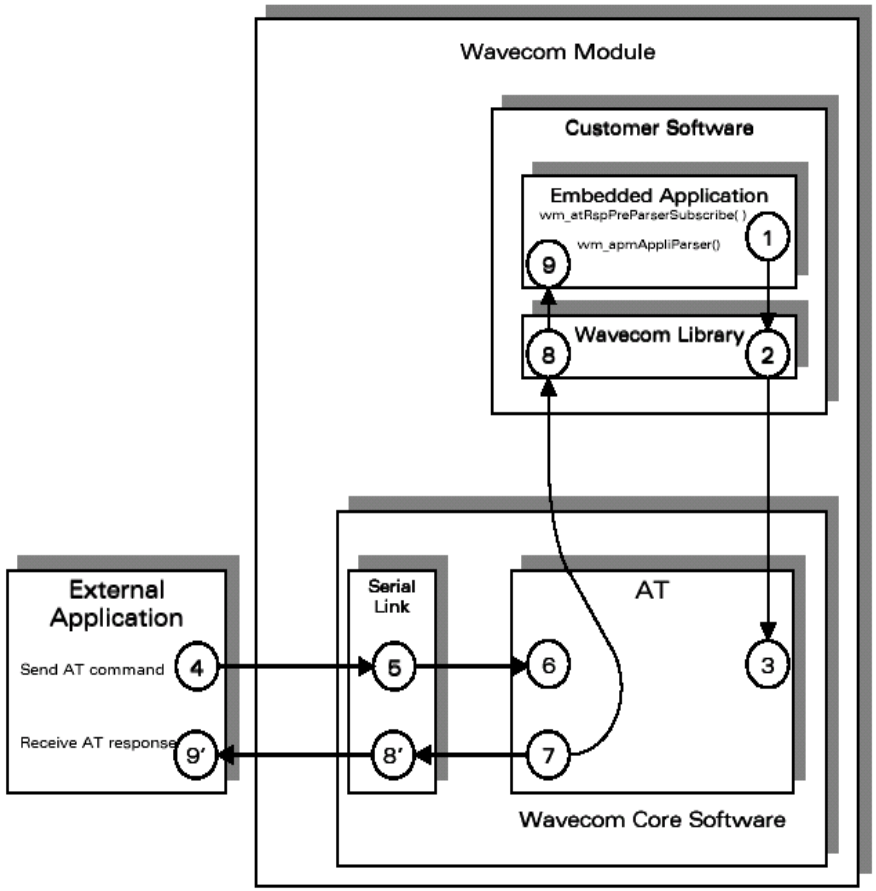
7) The AT function checks the subscription status of the response and sends it to both the External Application and the Embedded Application,

8) The response is dispatched by the Wavecom library, which calls the "wm_apmAppliParser" function of the Embedded Application (the Message type is WM_AT_RSP_PRE_PARSER),

9) This function processes the response (the parameters of the function include a broadcast response indication),

8') This response is sent through the serial link,

9') The External Application receives the response.



Example of Cooperative mode (Rsp spying)

```
/*************************************************/
/*  Appli.c   -  Copyright Wavecom S.A. (c) 2001    */
/*************************************************/

#include "wm_types.h"
#include "wm_apm.h"
#define TIMER 01

/*************************/
/*  Mandatory Variables    */
/*************************/

char wm_apmCustomStack[1024];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

/*************************/
/*  Mandatory Functions    */
/*************************/

/*************************************/
/*  wm_apmAppliInit                  */
/* Embedded Application initialisation */
/*************************************/
void 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 ) );
}
/*************************************/
/*  wm_apmAppliParser                */
/* Embedded Application message parser */
/*************************************/
bool 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 TRUE;
}
```

V1.1 New APIs

- SPI/I2C bus management
  - **wm_busOpen ()**
    - Allocates a Handle on the required bus, and opens it for further read/write operations.
  - **wm_busClose ()**
    - Closes a bus previously allocated by the wm_busOpen function
  - **wm_busWrite ()**
    - Writes on a bus previously allocated by the wm_busOpen function
  - **wm_busRead ()**
    - Reads on a bus previously allocated by the wm_busOpen function

V1.1 New APIs

- GPIO management
  - **wm_ioAllocate ()**
    - **Allocates one or more Gpio(s) for the embedded application use**
  - **wm_ioRelease ()**
    - **R**eleases one or more Gpio reserved by the wm_ioAllocate function
  - **wm_ioSetDirection ()**
    - Changes the direction of one or several allocated Gpio
  - **wm_ioRead ()**
    - Reads the current state of one or more allocated Gpio(s)
  - **wm_ioSingleRead ()**
    - Reads the current state of one single allocated Gpio
  - **wm_ioWrite ()**
    - Defines a new state for one or more allocated Gpio(s)
  - **wm_ioSingleWrite ()**
    - Define a new state for one single allocated Gpio

V1.1 New APIs

- Others
  - **wm_atSendIntermediateExternalApp ()**
    - Sends an AT intermediate response to an external application
  - **wm_atSendUnsolicitedExternalApp ()**
    - Sends an AT unsolicited response to an external application
  - **wm_osDeleteAllFlashData ()**
    - Deletes all the data previously stored in flash memory by the embedded application

SPI/I2C bus management

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

BusType:

Type of the bus to open. Defined values are:

- **WM_BUS_SPI1** for SPI bus ;
- **WM_BUS_SOFT_I2C** for I2C software 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_busI2CSoftSettings_t     I2C_Soft;
} wm_busSettings_u;
```

SPI/I2C bus management

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

```
typedef struct
{
        u32                     Clk_Speed;
        u32                     Clk_Mode;
} wm_busSPISettings_t;
```

The Clk_Speed parameter is the SPI clock speed ; defined values are:

- **WM_SCL_SPEED_101Khz ;**
- **WM_SCL_SPEED_812Khz ;**
- **WM_SCL_SPEED_1_625MHz ;**
- **WM_SCL_SPEED_3_25MHz.**

The Clk_Mode parameter is the SPI clock mode ; defined values are:

- **WM_SCK_MODE_0** (rest state 0, data valid on rising edge);
- **WM_SCK_MODE_1** (rest state 0, data valid on falling edge);
- **WM_SCK_MODE_2** (rest state 1, data valid on rising edge);
- **WM_SCK_MODE_3** (rest state 1, data valid on falling edge);

SPI/I2C bus management

To open the I2C soft bus, you must use the I2C_Soft parameter of the union, defined as below:

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

The Scl_Gpio parameter is the label of the Gpio used to handle the SCL signal.
The Sda_Gpio parameter is the label of the Gpio used to handle the SDA signal.
Each of these labels must be a member of the wm_ioLabel_u union (see §3.6.2.1.2).

3.7.2.2 Returned Values

On successful completion, the function returns a positive or null Handle, to use for further Read / Write / Close operations on this bus.

Otherwise, the function will return a negative error value (cf §3.7.1 "Return values definition").

SPI/I2C bus management

The wm_busClose function allows to close a bus previously allocated by the wm_busOpen function.
Its prototype is:

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

3.7.3.1 Parameters

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

3.7.3.2 Returned Values
On successful completion, the function returns 0.

Otherwise, the function will return a negative error value (cf §3.7.1 "Return values definition").

SPI/I2C bus management

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
                         u32 Address,
                         void * pDataToWrite,
                         u32 NbBytes );
```

SPI/I2C bus management

Handle:

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

Address:

Address of the device present on the requested bus, at which the function must write. This address depends on bus type:

For SPI: This parameter uses a set of chip select pins, dedicated to specific mapping of address:

- **WM_BUS_SPI_ADDRESS_NO_CS** : the function does not use any Chip Select (in order to use a GPIO as Chip Select, for example);
- **WM_BUS_SPI_ADDRESS_SPI_EN** : the function uses the SPI_EN pin as Chip Select ;
- **WM_BUS_SPI_ADDRESS_SPI_AUX** : the function uses the SPI_AUX pin as Chip Select.

For I2C soft: this parameter is the slave address byte. This is a 7-bits address, shift to left from 1 bit, padded with the LSB set to 0 (to write), and sent on the I2C bus before performing the writing operation.

pDataToWrite:

Buffer containing data to write on the requested bus.

NbBytes

Size of the pDataToWrite buffer. This size must not exceed 512 bytes.

SPI/I2C bus management

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
                        u32 Address,
                        void * pDataToRead,
                        u32 NbBytes );
```

SPI/I2C bus management

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

Address:
Address of the device present on the requested bus, at which the function must read. This address depends on bus type:
For SPI: this parametrer uses a set of chip of select pins, dedicated to specific mapping of address:

- **WM_BUS_SPI_ADDRESS_NO_CS** : the function does not use any Chip Select (in order to use a GPIO as Chip Select, for example) ;
- **WM_BUS_SPI_ADDRESS_SPI_EN** : the function uses the SPI_EN pin as Chip Select ;
- **WM_BUS_SPI_ADDRESS_SPI_AUX** : the function uses the SPI_AUX pin as Chip Select.

For I2C soft: this parameter is the slave address byte. This is a 7-bits address, shift to left from 1 bit, padded with the LSB set to 1 (ro read), and sent on the I2C bus before performing the readintg operation.

pDataToRead:
Buffer containing data to read from the requested bus.

For SPI bus, the 2 first bytes should be used to send an operation code byte to the slave, before performing the reading operation. The first byte is the operation code length, in bits (from 1 to 8). The second byte is operation code value (as the MSB in always sent first, if the length is less than 8 bits, only the most significant bytes will be sent (example: to send first a bit set to 1, the buffer must be set to "0180")).

Scope of the course

- Open-AT Overview
- Open-AT SDK Installation
- Development Tools Overview
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- **A Snapshot on the AT Commands**
- OpenAT Roadmap
- Q&A

A Snapshot on the AT Commands

15.14 Wavecom Downloading +WDWL

15.14.1 Description :

This **specific** command switches the product to download mode.
Downloading is performed using the 1K-XMODEM protocol.

15.14.2 Syntax :

Command syntax: AT+WDWL

Command	Possible responses
AT+WDWL *Note : Switch on downloading mode*	+WDWL: 0 *Note : Start the downloading*
	... *Note : Downloading in progress*
	AT+CFUN=1 *Note : Reset the product at the end*
OK	

A Snapshot on the AT Commands

4.10 Repeat last command A/

4.10.1 Description :

This command repeats the previous command. Only the A/ command itself cannot be repeated.

4.10.2 Syntax :

Command syntax : A/

Command	Possible responses
A/ *Note : Repeat last command*	

4.11 Power off +CPOF

4.11.1 Description :

This **specific** command stops the GSM software stack as well as the hardware layer. The AT+CFUN=0 command is equivalent to +CPOF.

4.11.2 Syntax :

Command syntax : AT+CPOF

Command	Possible responses
AT+CPOF *Note : Stop GSM stack*	OK *Note : Command valid*

A Snapshot on the AT Commands

4.12.2 Syntax :

Command syntax : AT+CFUN=<functionality level>

Command	Possible responses
AT+CFUN? *Note : Ask for current functionality level*	+CFUN: 1 OK *Note : Full functionality*
AT+CFUN=0 *Note : Set minimum functionality, IMSI detach procedure*	OK *Note : Command valid*
AT+CFUN=1 *Note : Set the full functionality mode with a complete software reset*	OK *Note : Command valid*

A Snapshot on the AT Commands

15.32.1 Description

This command allows Wavecom specific features to be enabled or disabled. Disabling a feature can be done with no restriction, but a password is required to enable one (or more) features.

15.32.2 Syntax

Command syntax AT+WCFM=<mode>,<FtrMask>[,<Password>]

Command	Possible responses
AT+WCFM=?	OK
AT+WCFM=0,"0A00" *Disable some features*	OK
AT+WCFM=1,"0003","1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF" *Enable features*	OK *The features are enabled (the password is correct)*
AT+WCFM=1,"0050","1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF" *Enable features*	+CME ERROR: 3 *Incorrect password*

15.32.3 Defined values

<mode>
0: disable some features of <FtrMask>
1: enable some features of <FtrMask>

<FtrMask> 16 bits hexadecimal string (4 characters from 0 to F)
<PassWord> 256 bits hexadecimal string (64 characters from 0 to F)

A Snapshot on the AT Commands

15.36.2 Syntax

Command syntax AT+WOPEN=<Mode>
Response syntax +WOPEN: <Mode>[,<IntVersion>[<ExtVersion>]]

Command	Possible responses
AT+WOPEN=?	+WOPEN: (0-2) OK
AT+WOPEN?	+WOPEN: 0 OK
AT+WOPEN=2 *Get the Open-AT library versions.*	+WOPEN: 2, "AT v1.00", "AT v1.00" OK *Open-AT v1.00 library version. An embedded application has been downloaded on this product.*
AT+WOPEN=1 *Start the embedded application.*	OK +WIND: 3 *Product reset in order to start the embedded application.*
AT+WOPEN=0 *Stop the embedded application.*	OK +WIND: 3 *Product reset in order to stop the embedded application.*
AT+WOPEN?	+CME ERROR: 3 *The Open AT feature is disabled.*

15.36.3 Defined values :

<Mode>
0: Stop the embedded application. If this one was running, the product resets.
1: Start the embedded application. If this one was stopped, the product resets.
2: Get the Open AT library versions.
<IntVersion> Ascii string giving the internal Open AT library version.
<ExtVersion> Ascii string giving the external Open AT library version.
Note :
If no embedded application is loaded, the <ExtVersion> parameter does not appear.

A Snapshot on the AT Commands

4.1 Manufacturer identification +CGMI

4.1.1 Description :

This command gives the manufacturer identification.

4.1.2 Syntax :

Command syntax : AT+CGMI

Command	Possible responses
AT+CGMI *Note : Get manufacturer identification*	WAVECOM MODEM OK *Note : Command valid, Wavecom modem*

4.2 Request model identification +CGMM

4.2.1 Description :

This command is used to get the supported frequency bands. With multi-band products the response may be a combination of different bands.

4.2.2 Syntax :

Command syntax : AT+CGMM

Command	Possible responses
AT+CGMM *Note : Get hardware version*	900P OK *Note : GSM 900 MHz primary band. Other possible answers: "900E" (extended band), "1800" (DCS), "1900" (PCS) or "MULTIBAND"*

A Snapshot on the AT Commands

4.3 Request revision identification +CGMR

4.3.1 Description :

This command is used to get the revised software version.

4.3.2 Syntax :

Command syntax : AT+CGMR

Command	Possible responses
AT+CGMR *Note : Get software version*	310_G250.51 806216 032199 17:04 OK *Note : Software release 3.10, revision 51 generated on the 21st of March 1999*

Scope of the course

- Open-AT Overview
- Open-AT SDK Installation
- Development Tools Overview
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- A Snapshot on the AT Commands
- OpenAT Roadmap
- Q&A

OpenAT Roadmap

- V432: (GSM only)
  - I2C bus management
  - SPI bus management
  - GPIO management through API
  - API to delete objects (i.e. user data) stored in flash
  - API to send “unsolicited” or “intermediate” indications

- V540:
  - P1: Open AT over GPRS
  - P2: Download of OpenAT application OTA

Scope of the course

- Open-AT Overview
- Open-AT SDK Installation
- Development Tools Overview
- Application Development process
- Open-AT technical overview
- A closer look on Open-AT APIs
- A Snapshot on the AT Commands
- OpenAT Roadmap
- **Q&A**