Porting Guide from Open AT® OS V2.10b to V3.10

Revision: 001

Date: August 2006

Make it wireless

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

Porting guide from Open AT® OS v2.10b to v3.10

Reference : **WM_DEV_OAT_UGD_022**

Version : **001**

Date : **August 11th, 2006**

Overview

This user guide describes the main differences between the Open AT® interfaces from version 2 generation to version 3 generation. For each modified interface, code samples are provided. New interfaces are also briefly described.

Document History

Level	Date	History of the evolution	
001	11th August, 2006	Creation	

Copyright

Trademarks

Table of Contents

1 Introduction

1.1 References

I. Open AT® Development Guide for revision 009
(Ref WM_ASW_OAT_UGD_002 revision 009)

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 Core software	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 Core Software functions.
Wavecom Core Software	Set of GSM and open functions supplied to the User.

1.3 Abbreviations

A&D	Application & Data
ADL	Application Development Layer
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
SDK	Software Development Kit
SMA	Small Adapter
SMS	Short Message Services
WAP	Wireless Application Protocol
IDE	Integrated Development Environment

2 Modified Basic Mode interfaces

This chapter lists all modified interfaces from v2.10b generation to v3.10, with code samples.

2.1 Minimum Embedded Application Code

The following minimum application code in Open AT® OS v3 must be added to write an Open AT® v3 application.

```
const wm_apmTask_t wm_apmTask [ WM_APM_MAX_TASK ] =
{
        { StackSize1, Stack1, InitFct1, ParseFct1 },
        { StackSize2, Stack2, InitFct2, ParseFct2 },
        { StackSize3, Stack3, InitFct3, ParseFct3 }
};
```

Here Stack*x* and StackSize*x* are variables used to define the application task call and stack size.

InitFct*x* are functions called during initialization. ParseFct*x* are functions which are called each time a message is received.

These functions have to be defined and implemented by the user.

2.2 Open AT® OS Specific AT Commands

2.2.1 AT+WOPEN command

This command is used to start/stop/configure an Open AT® application.

From Open AT® OS v3.10, this command has been modified to configure the A&D and Open AT® application space. The new parameters available with this command are mentioned below:

Parameter	Description
5	Suspend the Open AT® embedded application tasks
6	If the A&D size parameter is used, configure the Application and storage size, otherwise display the current A&D storage space size and Open application space size

2.3 Memory Management

The Embedded software runs within an RTK task: the user must define the size of the customer application call stack. The Wavecom Core Software and the Embedded Application manage their own RAM area. Any access from one of these programs to the other's RAM area is prohibited and causes a reboot.

In case an Embedded Application uses more than the maximum allocated RAM in global variables, or uses more than the maximum allocated ROM, then the behavior of the embedded software becomes erratic. Global variables, call stack and dynamic memory are all part of the RAM allocated to the Embedded Application.

The memory configuration is different in Open AT® OS v3.10. The difference in memory configuration is mentioned below:

Old Memory Configuration
For 16 Mbits flash size products: • 320 Kbytes of ROM • 32 Kbytes of RAM • 5 Kbytes of Flash Object Data For 32 Mbits flash size products: • 512 Kbytes of ROM • 128 Kbytes of RAM • 128 Kbytes of Flash Object Data
New Memory configuration
For 32 Mbits flash size products : • 768 Kbytes of ROM (configurable using AT+WOPEN command) • 128 Kbytes of RAM • 128 Kbytes of Flash Object Data • 768 Kbytes of Application and Data Storage Volume (configurable using AT+WOPEN command) Note: 16 Mbits product are not supported in this version

2.4 Security

2.4.1 Software Security

Two software safeguards are used in the Open AT® platform:

- RAM access protection
- Watchdog protection

After reboot, the input parameter of “wm_apmAppliInit ()” function is set to WM_APM_REBOOT_FROM_EXCEPTION. If a reboot is caused by a software crash, the application is started 20 seconds after the start of the Wavecom OS. This allows at least 20 seconds delay to re-download a new application. In case of normal reboot, the application restarts immediately.

2.4.2 WatchDog Protection

The Embedded Application software is protected from reaching a dead-end lock by a watchdog timer value. In case of a crash, the software reboots. If an Embedded Application crash is detected, the Target Monitoring Tool screen displays: “Customer watchdog”. The watchdog timer value has been changed in Open AT® OS v3.10 to 8 seconds from 4.2 seconds.

2.5 API

2.5.1 New Open AT® Tasks Configuration APIs

These APIs allows to configure the tasks parameters such as stack size. These APIs are added from Open AT® OS v3.10. This section describes the various parameters that needs to be configured for tasks.

2.5.1.1 Task Identifiers

The tasks are identified by the task 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,                      // Task number
   WM_OS_TASK_WAVECOM = 0xFF            // For message coming from
                                        //Wavecom Core
                                        // Software
} wm_osTask_e;
```

2.5.1.2 Task Table

The task table is used to define the embedded application tasks parameters

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

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

```
const wm_apmTask_t wm_apmTask [ WM_APM_MAX_TASK ] =

{

      { StackSize1, Stack1, Init1, Parser1 },

      { StackSize2, Stack2, Init2, Parser2 },

      { StackSize3, Stack3, Init3, Parser3 }

};
```

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

2.5.1.3 Stack initialization

The following variables are used to define stack size for each task.

```
#define StackSize1 1024       // Value 1024 is a sample value
#define StackSize2 1024       // Value 1024 is a sample value
#define StackSize3 1024       // Value 1024 is a sample value

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

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

2.5.1.4 The init functions

This function is called during initialization of the task.

Prototype:

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

2.5.1.5 The Parser functions

These functions are invoked, each time a message is received from the Wavecom core software.

Prototype:

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

New message structures and message types are added to the body of wm_apmMsg_t (shown in **bold**). The structures of these messages are explained below. The message body is as follows

```
/* Body Part of the Message */
typedef union
{
  wm_atResponse_t             ATResponse;
  wm_atUnsolicited_t          ATUnsolicited;
  wm_atIntermediate_t         ATIntermediate;
  wm_atCmdPreParser_t         ATCmdPreParser;
  wm_atRspPreParser_t         ATRspPreParser;
  wm_osTimer_t                OSTimer;
  wm_osRelease_t              OSRelease;
```

```
     wm_fcmReceiveBlock_t        FCMReceiveBlock;
     wm_fcmOpenFlow_t            FCMOpenFlow;
     wm_fcmFlow_e                FCMCloseFlow;
     wm_fcmFlow_e                FCMResumeFlow;
     wm_ioSerialSwitchStateRsp_t IOSerialSwitchStateRsp;
     wm_ioPortUpdateInfo_t       IOPortUpdateInfo;
    } wm_apmBody_t
```

WM_IO_PORT_UPDATE_INFO:

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

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;
```

2.6 AT Command API's

These API's are used to manage the AT commands. This includes

- sending of AT command string
- reception of AT responses
- pre-parsing of at commands and at responses
- intermediate at responses
- unsolicited responses

2.6.1 The wm_atSendCommand function *(Unchanged)*

Prototype:

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

This function is a shortcut to the wm_atSendCommandExt function, with the Dest parameter always set to the WM_IO_OPEN_AT_VIRTUAL_BASE value.

2.6.2 The wm_atSendCommandExt Function *(New)*

The wm_atSendCommand function allows sending AT commands on a required port.

Prototype:

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

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 ATcommand.
- **AtStringSize:** Size of the previous parameter, It equals the length + 1 and includes the 0x00 character.
- **ResponseType:** Determines whether the response will be sent to external and/or embedded application This parameter can take the values defiend in the below mentioned enumberation.:

```
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 commands has to be executed.

2.6.3 The wm_atUnsolicitedSubscription function *(Unchanged)*

Prototype:

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

The function subscribes to the unsolicited messages. The unsolicited messages include incoming call indication, WIND indications, etc.

2.6.4 The wm_atIntermediateSubscription function (*Unchanged*)

Prototype:

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

The function subscribes to the intermediate messages. The intermediate messages include responses like ">".

2.6.5 The wm_atCmdPreParserSubscribe function *(Unchanged)*

Prototype:

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

The function subscribes to the Command Pre Parsing of AT Command String.

2.6.6 The wm_atRspPreParserSubscribe function *(Unchanged)*

Prototype:

```
void wm_atRspPreParserSubscribe (wm_atRspPreSubscribe_e SubscribeType);
```

The function subscribes to the Response Pre Parsing of AT Command String.

2.6.7 The wm_atSendRspExternalApp function *(Unchanged)*

Prototype:

void wm_atSendRspExternalApp (u16 AtStringSize, ascii *AtString);

This function sends an AT response to the external application. This function is a shortcut to wm_atSendRspExternalAppExt function but the destination is always UART1.

2.6.8 The wm_atSendRspExternalAppExt function *(New)*

Prototype:

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

Parameters:

- AtStringSize: 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 ATcommand.
- AtStringSize: Size of the previous parameter, It equals the length + 1 and includes the 0x00 character
- Dest: Specifies the port where the specified responses have to be sent. Available ports may be opened and closed dynamically by any application (an external or an Open AT® one)

2.6.9 The wm_atSendUnsolicitedExternalApp function *(Unchanged)*

The wm_atSendUnsolicitedExternalApp function sends an AT unsolicited response to the External Application. The Unsolicited response will be sent to all ports.

Prototype:

void wm_atSendUnsolicitedExternalApp (u16 AtStringSize, ascii *AtString);

2.6.10 The wm_atSendUnsolicitedExternalAppExt function *(New)*

This function behaves similar to the wm_atSendUnsolicitedExternalApp function, It provides an additional parameter to send the unsolicited response on one specific port, instead of broadcasting it on all ports.

Prototype:

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

Parameters:

- AtStringSize: 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 ATcommand.
- AtStringSize: Size of the previous parameter, It equals the length + 1 and includes the 0x00 character
- Dest: Specifies the port where the specified responses have to be sent. Available ports may be opened and closed dynamically by any application (an external or an Open AT® one).

2.6.11 The wm_atSendIntermediateExternalApp function *(Unchanged)*

This function sends an AT intermediate response to the external application (always on UART1).

Prototype:

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

2.6.12 The wm_atSendIntermediateExternalAppExt function *(New)*

This function sends an intermediate AT response to the external application. The sender can decide the port on which the response should be sent

Prototype:

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

.Parameters:

- AtStringSize: 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 ATcommand.
- AtStringSize: Size of the previous parameter, It equals the length + 1 and includes the 0x00 character
- Dest: Specifies the port where the specified responses have to be sent. Available ports may be opened and closed dynamically by any application (an external or an Open AT® one).

2.7 Debug API's

2.7.1 The wm_osDebugTrace Function *(Unchanged)*

Prototype:

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

Debug function to print the debug information.

2.7.2 The wm_osDebugFatalError Function *(Unchanged)*

Prototype:

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

This stores the error code and performs a reboot.

2.7.3 The wm_osDebugEraseAllBackTraces Function *(New)*

Prototype:

```
    void wm_osDebugEraseAllBackTraces ( void )
```

This function reinitializes the product backtraces storage space. All the currently stored backtraces are erased.

2.7.4 The wm_ osDebugInitBacktracesAnalysis Function *(New)*

Prototype:

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

This function has to be called in order to start the debug analysis.

2.7.5 The wm_osDebugRetrieveBacktrace Function *(New)*

Prototype:

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

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 function will allow to retrieve all the wm_osDebugRetrieveBacktrace backtraces, until the function returns a negative value.

2.8 OS API's

These APIs manage the timer and the flash. The return value of OS API's is 'OK' on success and 'ERROR' on error.

2.8.1 The wm_osStartTimer function *(Unchanged)*

Prototype:

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

Starts the timer defined by timer Id

2.8.2 The wm_osStopTimer function *(Unchanged)*

Prototype:

s32 wm_osStopTimer (u8 TimerId);

Stops the timer identified by timer Id

2.8.3 The wm_osStartTickTimer function *(New)*

Prototype:

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

Starts the timer identified by timer Id. The 'TimerValue' is in steps of 18.5 ms ticks.

2.8.4 The wm_osStopTickTimer function *(New)*

Prototype:

s32 wm_osStopTickTimer (u8 TimerId);

Stops the timer identified by 'TimerId' and started by wm_osStartTickTimer().

Note on Flash Management

Flash identifiers up to 2000 can exist at the same time using Open AT® OS v3.10. Valid values are 0 to 0xFFFF

2.8.5 The wm_osWriteFlashData function *(Unchanged)*

Prototype:

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

Writes data in the flash memory and the Id is assigned to the stored data,

2.8.6 The wm_osReadFlashData function *(Unchanged)*

Prototype:

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

Reads the data from the flash referred by Id.

2.8.7 The wm_osGetLenFlashData function *(Unchanged)*

Prototype:

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

Gives the length of the data stored in flash memory referred by Id

2.8.8 The wm_osDeleteFlashData function *(Unchanged)*

Prototype:

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

Deletes the Data stored in flash memory referred by Id

2.8.9 The wm_osDeleteAllFlashData function *(Unchanged)*

Prototype:

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

Deletes all the Data stored in flash memory

2.8.10 The wm_osGetAllowedMemoryFlashData function *(Changed)*

Prototype:

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

Get the allowed flash memory

Old interface

//Get allocated memory
wm_osGetAllocatedMemoryFlashData( );

New interface

…
 	
// Get allocated memory
wm_osGetAllowedMemoryFlashData( );

2.8.11 The wm_osGetFreeMemoryFlashData function *(Unchanged)*

Prototype:

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

Get the free flash memory.

2.8.12 The wm_osGetUsedMemoryFlashData Function *(New)*

Prototype:

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

The wm_osGetUsedMemoryFlashData function returns the quantity of memory used by flash objects between the provided start & end IDs.

2.8.13 The wm_osDeleteAllFlashData Function *(New)*

Prototype:

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

The wm_osDeleteAllFlashData function deletes all the data previously stored in flash memory by the Embedded Application.

2.8.14 The wm_osDeleteRangeFlashData Function *(New)*

Prototype:

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

The wm_osDeleteRangeFlashData function deletes all the flash objects between the provided start & end IDs.

2.8.15 The wm_osGetHeapMemory function *(Unchanged)*

Prototype:

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

Allocate dynamic memory (in bytes).

2.8.16 The wm_osReleaseHeapMemory function *(Unchanged)*

Prototype:

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

Releases the previously allocated heap memory.

2.8.17 The wm_osSuspend function *(New)*

Prototype:

```
void wm_osSuspend( void);
```

Suspends all OAT tasks in the kernel; the tasks will not be run by the scheduler.

2.8.18 The wm_osGetTask function *(New)*

Prototype:

```
u8 wm_osGetTask ( void );
```

Get the current task Id.

2.8.19 The wm_osSendMsg function *(New)*

Prototype:

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

Sends a message to required task.

2.9 FCM Service

The flow control manager API's provide two I/O flows to the embedded application, one from V24 and other from a data communication. In Open-AT v3, the data communication is possible through GPRS along with GSM.

V24 serial link has following flows since OAT v3.10

- UART1
- UART2
- USB
- Logical CMUX flows
- Logical Bluetooth flows

2.9.1 Required Header

In the FCM header file, the **wm_fcmFlow_e** enum type maps to enum wm_ioPort_e which has additional entries since OAT v3 (Shown in bold). This corresponds to the added flows.

```
typedef enum
{
   WM_IO_NO_PORT,
   WM_IO_UART1,    // Uart 1 port
   WM_IO_UART2,    // Uart 2 port
#ifndef __WAVECOM_DEC__
   WM_IO_USB,      // USB port

   WM_IO_UART_MAX = WM_IO_USB,    // Number of physical Uarts
#else
   WM_IO_UART_MAX,
#endif  // __WAVECOM_DEC__
   WM_IO_UART1_VIRTUAL_BASE        = 0x10, // Base for UART1 virtual ports
   WM_IO_UART2_VIRTUAL_BASE        = 0x20, // Base for UART2 virtual ports
   WM_IO_USB_VIRTUAL_BASE        = 0x30, // Base for USB virtual ports
```

WM_IO_BLUETOOTH_VIRTUAL_BASE = 0x40, // Base for BlueTooth virtual ports

WM_IO_GSM_BASE = 0x50, // Base for GSM CSD FCM data flow

WM_IO_GPRS_BASE = 0x60, // Base for GPRS FCM Data flow

WM_IO_OPEN_AT_VIRTUAL_BASE = 0x80 // Base for Open-AT application tasks

} wm_ioPort_e;

2.9.2 Functions Removed

- wm_fcmOpenGPRSAndV24
- wm_fcmCloseGPRSAndV24
- wm_fcmOpenDataAndV24
- wm_fcmCloseDataAndV24

2.9.3 The wm_fcmIsAvailable Function *(New)*

Prototype:

bool wm_fcmIsAvailable (wm_fcmFlow_e FlowID);

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

2.9.4 The wm_fcmOpen function *(New)*

Prototype:

s32 wm_fcmOpen (wm_fcmFlow_e Flow, u16 DataMaxToReceive);

Opens the requested flow with specified maximum packet size. The packet size should not exceed 120 for serial link, 270 for GSM data flow and is not used for GPRS flow.

2.9.5 The wm_fcmClose function *(New)*

Prototype:

s32 wm_fcmClose (wm_fcmFlow_e Flow);

Closes the requested flow.

2.9.6 The wm_fcmSubmitData function *(Unchanged)*

Prototype:

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

The embedded application uses this function to submit data to FCM

The value returned by the function is as per the defined return values.

2.9.7 The wm_fcmCreditToRelease function *(Unchanged)*

Prototype:

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

This function tells the flow control manager that the customer has already used the data received in one or several messages.

The value returned by the function is as per the defined return values.

2.9.8 The wm_fcmQuery function *(New)*

Prototype:

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

The function gives the status of the FCM buffers.

2.10 Input Output API

These APIs manage serial link state and GPIO operations.

- The members in the union **wm_ioLabel_u** have been modified. The new member is specified in bold.

```
/* GPIO labels union */
typedef union
{
   wm_ioLabel_Q24X0_e   Q24X0_Label;  /*GPIO labels Q24X0 Product */
   wm_ioLabel_Q24X3_e   Q24X3_Label;  /* GPIO labels Q24X3 Product */
   wm_ioLabel_Q24X6_e   Q24X6_Label;  /* GPIO labels Q24X6 Product */
   wm_ioLabel_P32X3_e   P32X3_Label;  /* GPIO labels P3XX3 Product */
   wm_ioLabel_P32X6_e   P32X6_Label;  /* GPIO labels P32X6 Product */
   wm_ioLabel_Q31X6_e   Q31X6_Label;  /* GPIO labels Q31X6 Product */
   wm_ioLabel_P51X6_e   P51X6_Label;  /* GPIO labels P5186 Product */
   wm_ioLabel_P25X1_e   P25X1_Label;  /* GPIO labels Q2501 Product */
} wm_ioLabel_u;
```

- Following ENUMs have been added

```
/* GPIO labels for Wismo Quick P2501 product */
typedef enum
{
   WM_IO_Q25X1_GPI       = 0x00000001,
   WM_IO_Q25X1_GPO_0     = 0x00000002,
   WM_IO_Q25X1_GPO_1     = 0x00000004,
   WM_IO_Q25X1_GPO_2     = 0x00000008,
   WM_IO_Q25X1_GPO_3     = 0x00000010,
   WM_IO_Q25X1_GPIO_0    = 0x00000020,
   WM_IO_Q25X1_GPIO_1    = 0x00000040,
   WM_IO_Q25X1_GPIO_2    = 0x00000080,
   WM_IO_Q25X1_GPIO_3    = 0x00000100,
   WM_IO_Q25X1_GPIO_4    = 0x00000200,
   WM_IO_Q25X1_GPIO_5    = 0x00000400,
   WM_IO_P51X6_PAD       = 0x7FFFFFFF
} wm_ioLabel_P25X1_e;
```

2.10.1 The wm_ioSerialSwitchState function *(Changed)*

Prototype:

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

Sets the mode of the requested serial interface

- The wm_ioPort_e has following values

```
typedef enum
{
   WM_IO_UART1,    // Uart 1 port
   WM_IO_UART2,    // Uart 2 port
   WM_IO_USB,      // USB port

   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;
```

Old interface

`// Switch back to AT mode`
`wm_ioSerialSwitchState ( WM_IO_SERIAL_AT_MODE );`

New interface

`//  Switch back to AT mode`
`wm_ioSerialSwitchState ( WM_IO_UART1, WM_IO_SERIAL_AT_MODE );`

2.10.2 The wm_ioSerialGetSignal function *(New)*

Prototype:

s32 wm_ioSerialGetSignal(wm_ioPort_e Port, wm_ioSerialGetSignal_e SerialSignal);

Get the current value of a signal.

2.10.3 The wmioIsPortAvailable function *(New)*

Prototype:

bool wmioIsPortAvailable (wm_ioPort_e Port);

The function allows to query the current port state (open or closed).

2.10.4 The wm_ioAllocate function *(Unchanged)*

Prototype:

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

This function is used to reserve the GPIO(s) for Open AT® application.

2.10.5 The wm_ioRelease function*(Unchanged)*

Prototype:

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

Releases one or more GPIO(s) reserved by the application.

2.10.6 The wm_ioSetDirection function *(Unchanged)*

Prototype:

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

This function changes one or more GPIO(s) direction.

2.10.7 The wm_ioRead function *(Unchanged)*

Prototype:

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

Reads one or more GPIO(s) state.

2.10.8 The wm_ioSingleRead function *(Unchanged)*

Prototype:

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

Reads one GPIO state.

2.10.9 The wm_ioWrite function *(Unchanged)*

Prototype:

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

Writes one or more GPIO(s) state.

2.10.10 The wm_ioSingleWrite function *(Unchanged)*

Prototype:

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

This function writes one GPIO state.

2.11 GPRS Service *(Unchanged)*

This service allows management of GPRS connectivity through the Wavecom modem. The API's available are as follows

wm_gprsAuthentification: Allows setting the authentication parameters login/password to use with a context id during PDP activation

wm_gprsIPCPInformations: Allows getting the current IPCP information to use with a particular context id after PDP activation.

wm_gprsOpen: Allows setting OpenAT as the user of the GPRS bearer associated with the context id.

wm_gprsClose: Allows to unset OpenAT as the user of the GPRS bearer associated with the context id

2.12 List API *(Unchanged)*

None

2.13 BUS Service

For bus operations the Q2501 Wireless CPU behaves as a Q2406B wireless CPU.

2.13.1 The wm_busOpen function *(Unchanged)*

Prototype :

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

- The new member of the **chipselect** has following defined constant

  WM_BUS_SPI_ADDRESS_CS_NONE – The chipselect is not handled by the Open AT® API, but will be handled inside the Open AT® application.

2.13.2 The wm_busClose function *(Unchanged)*

Prototype :

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

This function allows to close a previously opened bus.

2.13.3 The wm_busWrite function *(Unchanged)*

Prototype:

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

Allows to write data on previously allocated bus. The second parameter “address” has been changed o “pAccessMode”.

2.13.4 The wm_busRead function *(Unchanged)*

Prototype:

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

This function allows to read data from a previously allocated bus.

2.14 LIST Management *(Unchanged)*

These APIs provide the management of LIST data structure.

wm_lstCreate : Creates the list

wm_lstDestroy : Allows to clear and destroy the list

wm_lstClear : Allows to clear all provided list items without destroying

wm_lstGetCount : Returns the current item count

wm_lstAddItem : Allows to add an item to the provided list

wm_lstnsertItem : Allows to add an item at a given location in list

wm_lstDeleteItem : Allows to delete an item at the given indexes in list

wm_lstFindItem : Allows to find an item in the provided list

wm_lstFindAllItem : Allows to find all items in the provided list

wm_lstFindNextItem :Allows to find the next item index of the given list

wm_lstResetItem :Allows to reset all previously found items by wm_lstFindNextItem().

2.15 Standard Library Functions *(Unchanged)*

None

3 Deprecated Basic Mode Features

3.1 Scratch Memory APIs

The Scratch memory API does no more exist from Open AT® OS v3.10. In order to implement Over the Air Download, the Application and Data Storage API has to be used. All the scratch memory functions will return WM_SCRATCH_MEMORY_ NOT_AVAIL error code.

The Application and Data storage APIs can be found in the section 4.

4 New Basic Mode Features

4.1 Application and Data Storage API

4.1.1 The wm_adAllocate function *(New)*

Prototype :

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

Allocates a new cell in the Application and data storage API.

4.1.2 The wm_adWrite function *(New)*

Prototype :

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

Writes data to the requested cell.

4.1.3 The wm_adInfo function *(New)*

Prototype :

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

Obtains information on requested cell handle.

4.1.4 The wm_adFinalise function *(New)*

Prototype :

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

Finalizes a cell; this one will be then in read-only mode.

4.1.5 The wm_adInstall function *(New)*

Prototype :

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

Installs the provided A&D cell content.

4.1.6 Other A&D APIs

The A&D service also provides these APIs :

wm_adRetrieve	:	Initialize a handle on an already allocated cell
wm_adFindInit	:	Initialize a cell search
wm_adFindNext	:	Get next cell handle from initialized search
wm_adResume	:	Allow to resume write operations on the requested non finalized cell
wm_adDelete	:	Delete the requested cell .
wm_adStats	:	Provide information on the Application & Data storage space
wm_adSpaceState	:	Return the Application & Data storage space state
wm_adFormat	:	Destroy the whole Application & Data storage space
wm_adRecompactInit	:	Start the recompaction process
wm_adRecompact	:	Perform a new recompaction step

4.2 Sound API

4.2.1 The wm_sndTonePlay function *(New)*

Prototype:

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

Plays a tone.

4.2.2 The wm_sndToneStop function *(New)*

Prototype:

s32 wm_sndToneStop (wm_snd_dest_e Destination);

Stops a tone.

4.2.3 The wm_sndDtmfPlay function *(New)*

Prototype:

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

Plays a dtmf tone.

4.2.4 The wm_sndDtmfStop function *(New)*

Prototype:

s32 wm_sndDtmfStop (wm_snd_dest_e Destination);

Stops a dtmf tone.

4.2.5 The wm_sndMelodyPlay function *(New)*

Prototype:

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

Plays a melody.

4.2.6 The wm_sndMelodyStop function *(New)*

Prototype:

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

Stops a melody.

4.3 GPS API

This service provides APIs to access the Q2501 product GPS data.

4.4 RTC API

This service provides APIs to access the Real Time Clock in the wireless CPU directly.

5 Modified ADL Interfaces

5.1 Imported API's from Standard Mode

The following APIs are added in Open AT® OS v3.10 from the standard mode.

Standard API	– Imported in both the versions
List API	– Imported in both the versions
Scratch Memory API	– Imported only in Open AT® OS v2.10b
Sound APIs	– Imported only in Open AT® OS v3.10

5.2 UART2 and GPIOs shared resources

When the second UART of the product is used some of the GPIOs are no more available to the Open AT® application. The impacted GPIOs are mentioned below:

Wavecom Module Series	Unavailable GPIOs
Q24X6	GPI, GPO 2, GPIO 0, GPIO 5
Q24X0	GPI, GPO 2, GPIO 0, GPIO 5
Q2501	GPI, GPO 2, GPIO 0, GPIO 5
Q32X6	GPI, GPIO 2
Q31X6	GPI, GPO 2, GPIO 4, GPIO 5
Q51X6	GPO 0, GPO 1, GPIO 5

5.3 Q2501 product external battery mechanism

On the Q2501 product, if the external battery charging mechanism is implemented (please refer to the AT+WHCNF command documentation), the Open AT® applications. GPIO 3 is locked on start-up, and is not available for Open AT®.

5.4 SIM Level Shifter and GPO shared resources

If any other feature than the "SIM3VONLY" one is enabled (please refer to the AT+WFM command documentation), a GPO (according to the table below, depending on the module) is locked for the SIM level shifter, and cannot be subscribed by the Open AT® application.

Wavecom Module Series	Unavailable GPO
Q24X6	GPO 0
Q24X0	GPO 0
Q25X1	GPO 1

5.5 Inner AT commands configuration

The ADL library needs for its internal processes to set-up some AT command configurations that differ from the default values. The concerned commands are listed below:

AT Command	Fixed Value
AT+CMEE 1	1
AT+WIND	All indications
AT+CREG	2
AT+CGREG	2
AT+CRC	1
AT+CGEREP	2
ATV	1
ATQ	0

The above fixed values are set-up internally by ADL. This means that all related error codes (for +CMEE) or unsolicited results are always all available to all ADL applications, without requiring them to be sent (using the AT corresponding configuration command).

5.6 AT Command Services

These API's are used to manage the AT commands. This includes

- sending of AT command string
- reception of AT responses
- pre-parsing of at commands and at responses
- intermediate at responses
- unsolicited responses

Due to the new UART support, AT commands related APIs have been modified to process multi-UART settings.

5.6.1 Unsolicited and Subscribed commands structure

The adl_atCmdPreParser_t structure (used when a subscribed command is received) includes a new "**Port**" field, which is set to the UART id from which the command has been entered.

```
typedef struct
{
        u16             Type;                   // Cmd type
        u8              NbPara;                 // Parameters number
        adl_atPort_e    Port;           // Source port
        wm_lst_t        ParaList;       // Parameters list
        u16             StrLength;      // Command string length
        ascii           StrData[1];             // Command string
} adl_atCmdPreParser_t;
```

The adl_atUnsolicited_t structure includes a new "**Dest**" field to get the destination port.

```
typedef struct
{
        adl_strID_e RspID;  // Standard response ID
        adl_atPort_e Dest;  // Unsolicited response destination port
        u16 StrLength;
        /* the length of the string (name) of the unsolicited response */
        ascii StrData[1];
        /* a pointer to the string (name) of the unsolicited response */
} adl_atUnsolicited_t;
```

5.6.2 The adl_atUnSoSubscribe function *(Unchanged)*

Prototype:

s16 adl_atUnSoSubscribe (ascii * UnSostr, adl_atUnSoHandler_t UnSohdl);

This function subscribes to a specific unsolicited response with an associated callback function. When the required unsolicited response is sent from the Wavecom Core Software, the callback function will be executed.

5.6.3 The adl_atUnSoUnSubscribe function *(Unchanged)*

Prototype:

s16 adl_atUnSoUnSubscribe (ascii *UnSostr, adl_atUnSoHandler_t UnSohdl);

This function un-subscribes from an unsolicited response and its handler.

5.6.4 adl_atSendResponse APIs *(Changed)*

Prototypes:

void adl_atSendResponse (u16 Type, ascii * Text);

void adl_atSendStdResponse (u16 Type, adl_strID_e RspID);

void adl_atSendStdResponseExt (u16 Type, adl_strID_e RspID, u32 arg);

The Type parameter of these APIs may now be set using the following macro :

ADL_AT_PORT_TYPE (port, type)

where port is the destination port where to send the response, and type is the response type (ADL_AT_RSP, ADL_AT_INT or ADL_AT_UNS).

Unsolicited responses are broadcasted to all ports. By default, if this macro is not used, responses are sent to the UART 1.

Old interface

```
// Command Handler
void CmdHdl (adl_atCmdPreParser_t * cmd)
{
    // Reply OK
    adl_atSendStdResponse                                          (
    ADL_AT_RSP,
    ADL_STR_OK );
}
```

New interface

```
// Command Handler
void CmdHdl (adl_atCmdPreParser_t * cmd)
{
    // Reply OK on incoming UART
    adl_atSendStdResponse    (    ADL_AT_PORT_TYPE    (ADL_AT_RSP,cmd->Port),
ADL_STR_OK );
}
```

5.6.5 Additional macros for specific port access

The above Response sending functions may be also used with the macros below, which provide the additional Port argument : it should avoid heavy code including each time the ADL_AT_PORT_TYPE macro call.

```
#define adl_atSendResponsePort(_t,_p,_r)
adl_atSendResponse(ADL_AT_PORT_TYPE(_p,_t),_r)

#define adl_atSendStdResponsePort(_t,_p,_r)
adl_atSendStdResponse(ADL_AT_PORT_TYPE(_p,_t),_r)

#define adl_atSendStdResponseExtPort(_t,_p,_r,_a)
adl_atSendStdResponseExt(ADL_AT_PORT_TYPE(_p,_t),_r,_a)
```

5.6.6 The adl_atCmdCreate function *(Changed)*

Prototype :

```
void adl_atCmdCreate ( ascii *atstr, u16 rspflag, adl_atRspHandler_t
rsphdl, ... );
```

The "rspflag" parameter of this API may now be set using the following macro:

```
ADL_AT_PORT_TYPE ( port, type )
```

where port is the destination port where to send unsubscribed responses, and type is a Boolean value (TRUE to forward unsubscribed responses, FALSE to filter them).

By default, if this macro is not used, responses are sent to the UART 1.

5.6.7 The adl_atCmdSendText function *(New)*

Prototype:

s8 adl_atCmdSendText (adl_port_e Port, ascii * Text)

This API is used to send intermediate text for a command which requires intermediate input text (For e.g. AT+CMGS).

Parameters

- **Port**: Port on which is currently running the "Text Mode" command, waiting for some text input.
- **Text**: Text to be provided to the running "Text Mode" command on the required port. If the text does not end with a 'Ctrl-Z' character (0x1A code), the function will add it automatically.

Returned values:

OK on success and negative values on error.

5.7 Timer Services

In the new Open AT® OS version 32 timers can run at the same time. There are no changes in the API interfaces for the timer.

5.8 Memory Services

A new API is added in this service to check the type of the memory.

5.8.1 The adl_memGetType function *(New)*

Prototype:

```
The adl_memType_e adl_memGetType (void);
```

The function returns the current module memory type. The adl_memType_e enumerated list is explained below:

```
typedef enum
{
  ADL_MEM_TYPE_A,
  ADL_MEM_TYPE_B
} adl_memType_e;
```

5.9 Debug Services

When the Full Debug configuration is selected in the used IDE (or with the command), the __DEBUG_APP__ and __DEBUG_FULL__ compilation flags are both defined and also the following macros. The macros defined in bold are the new ones.

TRACE ((u8 TL, ascii * T, ...))

DUMP (u8 TL, u8 * P, u16 L)

FULL_TRACE ((u8 TL, ascii * T, ...)) //Works exactly as the TRACE macro.

FULL_DUMP (u8 TL, u8 * P, u16 L) //Works exactly as the DUMP macro.

5.10 Flash service

In ADL v3.10 generation, flash objects' IDs are related to a provided handle, subscribed before any flash service API call.

5.10.1 The adl_flhSubscribe function *(New)*

Prototype:

```
s8 adl_flhSubscribe ( ascii * Handle, u16 NbObjectsRes );
```

Subscribes to a flash handle, and links a finished objects number to it.

5.10.2 The adl_flhExist function *(Changed)*

Prototype:

```
s32 adl_flhExist( ascii * Handle, u16 ID );
```

Gets the length of the requested flash object. May return a negative value on error.

Old interface

```
// Retrieve the length
u16 TheLength = adl_flhExist ( THE_ID );
// Test the length
if ( TheLength )
{
   // The object exists
}
```

New interface

```
// Retrieve the length
s32 TheLength = adl_flhExist ( MyFlhHandle, THE_ID );
// Test the length
if ( TheLength > OK )
{
   // The object exists
}
```

5.10.3 The adl_flhRead function _*(Changed)*

Prototype:

```
s8 adl_flhRead( ascii * Handle, u16 ID, u16 Len, u8 * ReadData );
```

Reads data from the requested object.

Old interface

// Reads the object data
u8 pData [10];
s8 sRet = adl_flhRead (THE_ID, 10, pData);

New interface

// Reads the object data
u8 pData [10];
s8 sRet = adl_flhRead (MyFlhHandle, THE_ID, 10, pData);

5.10.4 The adl_flhWrite function *(Changed)*

Prototype:

```
s8 adl_flhWrite( ascii * Handle, u16 ID, u16 Len, u8 * WriteData );
```

Writes data in the requested object.

Old interface

// Write the object data
u8 pData [10];
s8 sRet = adl_flhWrite (THE_ID, 10, pData);

New interface

// Write the object data
u8 pData [10];
s8 sRet = adl_flhWrite (MyFlhHandle, THE_ID, 10, pData);

5.10.5 The adl_flhErase function *(Changed)*

Prototype:

```
s8 adl_flhErase( ascii * Handle, u16 ID );
```

Erases requested object.

Old interface
// Erase the object s8 sRet = adl_flhErase (THE_ID);
New interface
// Erase the object s8 sRet = adl_flhErase (MyFlhHandle, THE_ID);

5.10.6 The adl_flhGetFreeMem function *(Unchanged)*

Prototype:

```
u32 adl_flhGetFreeMem( void );
```

Gets currently free flash memory.

5.10.7 The adl_flhGetIDCount function *(New)*

Prototype:

```
s32 adl_flhGetIDCount( ascii * Handle );
```

Gets the provided handle ID count, or the remaining ID count, if the parameter is set to NULL.

5.10.8 The adl_flhGetUsedSize function *(New)*

Prototype:

```
        s32 adl_flhGetUsedSize( ascii * Handle, u16 StartID, u16 EndID );
```

Gets the used size by an ID range in the handle.

5.11 FCM service

Due to new UART support, FCM constants have been modified to process multi-UART settings.

Flow IDs

The different flow IDs are now defined by the following enum

```
/* Ports identifiers */
typedef enum
{
   ADL_PORT_NONE,

   ADL_PORT_UART1,
   ADL_PORT_UART2,
   ADL_PORT_USB,

   ADL_PORT_MAX = ADL_PORT_USB,

   ADL_PORT_UART1_VIRTUAL_BASE      = 0x10, // Base for UART1
virtual ports
   ADL_PORT_UART2_VIRTUAL_BASE      = 0x20, // Base for UART2
virtual ports
   ADL_PORT_USB_VIRTUAL_BASE        = 0x30, // Base for USB virtual
ports
   ADL_PORT_BLUETOOTH_VIRTUAL_BASE  = 0x40, // Base for
BlueTooth virtual ports
   ADL_PORT_GSM_BASE                = 0x50, // GSM CSD call data port
   ADL_PORT_GPRS_BASE               = 0x60, // GPRS session port
   ADL_PORT_OPEN_AT_VIRTUAL_BASE    = 0x80  // Base for Open-AT
application tasks
} adl_port_e;
```

All flows may be subscribed in slave mode, with a bit-wise OR with following constant

```
// Flow subscribed as slave only
#define ADL_FCM_FLOW_SLAVE  0x20
```

To keep compatibility with v2 generation, old IDs are also redefined

```
// Constants for compatibility
#define ADL_FCM_FLOW_V24_MASTER ADL_FCM_FLOW_V24_UART1
#define  ADL_FCM_FLOW_V24            ( ADL_FCM_FLOW_V24_UART1  |
                                ADL_FCM_FLOW_SLAVE )

/* GPRS preferred on V24 Master subscription */
```

```
// Constant kept for compatibility ; not used anymore
#define ADL_FCM_FLOW_GPRS_PREFERRED 0x80
```

Old interface

`// Subscribe to V24 flow`
`s8 FcmHdl = adl_fcmSubscribe ( ADL_FCM_FLOW_V24_MASTER, MyFcmCtrlHandler, MyFcmDataHandler );`

New interface

`// Subscribe to V24 flow`
`s8 FcmHdl = adl_fcmSubscribe ( ADL_FCM_FLOW_V24_UART1, MyFcmCtrlHandler, MyFcmDataHandler );`

5.12 Scratch Memory API

This API has been replaced by the Application & Data Storage service. Please refer to this New Features in ADL description for a porting sample.

5.13 GPIO Service *(Unchanged)*

There are no changes in the API for the GPIO service. Please find below the GPIOs for the new Q2501 product.

For Wismo Quik Q25X1 product:

ADL_IO_Q25X1_GPI

ADL_IO_Q25X1_GPO_0

ADL_IO_Q25X1_GPO_1

ADL_IO_Q25X1_GPO_2

ADL_IO_Q25X1_GPO_3

ADL_IO_Q25X1_GPIO_0

ADL_IO_Q25X1_GPIO_1

ADL_IO_Q25X1_GPIO_2

ADL_IO_Q25X1_GPIO_3

ADL_IO_Q25X1_GPIO_4

ADL_IO_Q25X1_GPIO_5

5.14 Bus Service *(Unchanged)*

The new member of the Chip Select has following defined constant ADL_BUS_SPI_ADDRESS_CS_NONE – The chipselect is not handled by the Open AT® API, but will be handled inside the Open AT® application.

5.15 Errors Management

This feature contains the APIs for error management.

5.15.1 The adl_errHalt function *(New)*

Prototype:

```
void adl_errHalt ( u16 ErrorID const ascii *ErrorString );
```

This function causes an error, defined by its ID and string. If an error handler is defined, it will be called, otherwise a product reset will occur.

5.15.2 The adl_errEraseAllBacktraces function *(New)*

Prototype:

```
void adl_errEraseAllBacktraces ( void );
```

Backtraces (caused by the adl_errHalt function, ADL or the Wavecom Core Software) are stored in the product non-volatile memory. A limited number of backtraces may be stored in memory (depending on each backtrace size, and other internal parameters stored in the same storage place). The function allows to free and re-initialize this storage place.

5.15.3 The adl_errStartBacktraceAnalysis function *(New)*

Prototype:

```
void adl_errStartBacktraceAnalysis (void);
```

This function is to start the analysis to retrieve the backtraces from the memory.

5.15.4 The adl_errGetAnalysisState function *(New)*

Prototype:

```
adl_errAnalysisState_e  adl_errGetAnalysisState ( void );
```

This function is used to know the current backtrace analysis process state.

5.15.5 The adl_errRetrieveNextBacktrace function *(New)*

Prototype:

```
s32 adl_errRetrieveNextBacktrace ( u8 Handle , u8 *  BacktraceBuffer, u16
Size );
```

This function allows the application to retrieve the next backtrace buffer stored in the product memory. The backtrace analysis may have been started first with the function *adl_errGetAnalysisState*.

5.16 SIM Service

5.16.1 The adl_simGetState function *(New)*

Prototype

```
adl_simState_e adl_simGetState ( void );
```

This function gets the current SIM service state.

5.17 Call Service

5.17.1 The adl_callSetup function *(Unchanged)*

This function calls the adl_callSetupExt function on ADL_PORT_OPEN_AT_VIRTUAL_BASE port. Please note that events generated by the adl_callSetup will not be able to be forwarded to any external port, since the setup command was running on the Open AT® OS.

5.17.2 The adl_callSetupExt function *(New)*

Prototype:

```
s8 adl_callSetupExt ( ascii * PhoneNb,  u8 Mode, adl_port_e Port );
```

This function sets up a call to a specified phone number.

5.17.3 The adl_callHangup function *(Unchanged)*

This function calls the adl_callHangupExt function on the ADL_PORT_OPEN_AT_VIRTUAL_BASE port.

5.17.4 The adl_callHangupExt function *(New)*

Prototype:

```
s8 adl_callHangupExt ( adl_port_e Port );
```

This function hangs up the phone call.

5.17.5 The adl_callAnswer function *(Unchanged)*

This function calls the adl_callAnswerExt function on the ADL_PORT_OPEN_AT_VIRTUAL_BASE port.

5.17.6 The adl_callAnswerExt function *(New)*

Prototype:

```
s8 adl_callAnswerExt ( adl_port_e Port );
```

This function answers the incoming phone call.

5.18 GPRS Service

5.18.1 The adl_gprsSetup function *(Unchanged)*

This function calls the adl_gprsSetupExt function on ADL_PORT_OPEN_AT_VIRTUAL_BASE port.

5.18.2 The adl_gprsSetupExt function *(New)*

Prototype:

```
    s8 adl_gprsSetupExt ( u8 Cid, adl_gprsSetupParams_t  Params, adl_port_e
                                                   Port  );
```

This function sets up the PDP context activated by its CID with specific parameters.

5.18.3 The adl_gprsAct function *(Unchanged)*

This function calls the adl_gprsActExt function on ADL_PORT_OPEN_AT_VIRTUAL_BASE port.

5.18.4 The adl_gprsActExt function *(New)*

Prototype:

```
    s8 adl_gprsActExt ( u8 Cid, adl_port_e Port );
```

This function activates the specific PDP context identified by its CID.

5.18.5 The adl_gprsDeact function *(Unchanged)*

This function calls the adl_gprsDeactExt on ADL_PORT_OPEN_AT_VIRTUAL_BASE port.

5.18.6 The adl_gprsDeactExt function *(New)*

Prototype:

```
    s8 adl_gprsDeactExt ( u8 Cid, adl_port_e Port );
```

This function deactivates the PDP context identified by its CID with specific parameters.

5.18.7 The adl_gprsIsAnIpAddress function *(New)*

Prototype:

```
s8 adl_gprsIsAnIpAddress ( ascii *AddressStr );
```

This function checks whether the provided string is a valid IP address.

5.19 AT Strings Service

The structure containing the list of the AT standard response string has been modified in the new version. The area in bold gives the modifications

```
typedef enum
{
    ADL_STR_NO_STRING,  // Unknown string
    ADL_STR_OK,         // "OK"
    ADL_STR_BUSY,       // "BUSY"
    ADL_STR_NO_ANSWER,  // "NO ANSWER"
    ADL_STR_NO_CARRIER, // "NO CARRIER"
    ADL_STR_CONNECT,    // "CONNECT"
    ADL_STR_ERROR,      // "ERROR"
    ADL_STR_CME_ERROR,  // "+CME ERROR:"
    ADL_STR_CMS_ERROR,  // "+CMS ERROR:"
    ADL_STR_CPIN,       // "+CPIN:"
    ADL_STR_LAST_TERMINAL, // Terminal resp. are before this line
    ADL_STR_RING = ADL_STR_LAST_TERMINAL,   // "RING"
    ADL_STR_WIND,       // "+WIND:"
    ADL_STR_CRING,      // "+CRING:"
    ADL_STR_CPINC,      // "+CPINC:"
    ADL_STR_WSTR,       // "+WSTR:"
    ADL_STR_CMEE,       // "+CMEE:"
    ADL_STR_CREG,       // "+CREG:"
    ADL_STR_CGREG,      // "+CGREG:"
    ADL_STR_CRC,        // "+CRC:"
    ADL_STR_CGEREP,     // "+CGEREP:"
    // Last string ID
    ADL_STR_LAST
} adl_strID_e;
```

6 New Features in ADL

6.1 Application & Data Storage service

This service provides an enhanced large flash storage service, usable to download new data or software elements. This service replaces the old v2 generation Scratch Memory API.

6.1.1 The adl_adSubscribe function *(New)*

Prototype :

```
    s32 adl_adSubscribe ( u32 CellID, u32 Size );
```

Subscribes to an A&D cell identifier in order to read / write it.

6.1.2 The adl_adWrite function *(New)*

Prototype :

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

Writes data in an A&D subscribed cell.

Old interface

```
// Writes data
s32 sRet = wm_scmWrite ( Size, pData );
```

New interface

```
// Writes data
s32 sRet = adl_adWrite ( ADHdl, Size, pData );
```

6.1.3 The adl_adInfo function *(New)*

Prototype :

```
    s32 adl_adInfo ( u32 Handle, adl_adInfo_t * Info );
```

Gets information from an A&D subscribed cell.

Old interface

```
// Reads data
s32 sRet = wm_scmRead ( Size, pData );
```

New interface

```
// Gets information
adl_adInfo_t Info;

s32 sRet = adl_adInfo ( ADHdl, &Info );

// Reads data (direct memory access)
wm_memcpy ( pData, Info.data, Size );
```

6.1.4 The adl_adFinalize function *(New)*

Prototype :

```
    s32 adl_adFinalise ( u32 Handle );
```

Finalizes a cell; After this, cell is in read-only mode.

Old interface

```
// Close Scratch Memory
s32 sRet = wm_scmClose();
```

New interface

```
// Finalize cell
s32 sRet = adl_adFinalize ( ADHdl );
```

6.1.5 The adl_adInstall function *(New)*

Prototype :

```
     s32 adl_adInstall ( u32 Handle );
```

Installs the subscribed A&D cell content.

Old interface

```
// Install Scratch Memory content
s32 sRet = wm_scmInstall();
```

New interface

```
// Installs cell
s32 sRet = adl_adInstall ( ADHdl );
```

6.1.6 Other A&D APIs

The A&D service also provides these APIs :

`adl_adDelete`	: to delete a subscribed A&D cell.
`adl_adRecompact`	: to free the deleted cells space.
`adl_adGetState`	: to get the A&D service state.
`adl_adGetCellList`	: to get the current A&D cells list.

6.2 GPS service

This service provides APIs to access the Q2501 product GPS data.

6.3 AT/FCM IO ports service

This ADL application can use this service to be informed about the product AT/FCM IO port states.

6.4 RTC Service

This service provides APIs to access the Real Time Clock in the wireless CPU directly

7 Upgrading Module with Open AT® OS V3.10 applications

Important Warning:

Once a new 6.55 Open AT® FW (or upper) is downloaded on the product, any existing Open AT® V2.10b or older application must firstly be erased, using the "AT+WOPEN=4" command.

- On 'B' memory WISMO products, existing v2.10b Open AT® application will automatically be erased by the download process ;

Existing applications on the module may be checked with the "AT+WOPEN=2" command. Answers may be:

Firmware Version / Open AT® OS version	Answer to AT+WOPEN=2
6.55 / 2.10b	+WOPEN: 2,"AT v03.10","AT v02.10b"
6.55 / 3.10	+WOPEN: 2,"AT v03.10","AT v03.10"

In the second case above:

- the "AT+WOPEN=4" command has to be used to delete the existing v2.10b Open AT® application **or**
- a new v3.10 Open AT® application should be downloaded in order to overwrite the existing one.

8 SDK Tools evolutions

8.1 SDK Setup

The Open AT® setup is re-designed in Open AT® OS V3.10:

- Already installed components are automatically detected ;
- No more additional setup script to call from the Cygwin command line ;
- No more manually environment variable update.

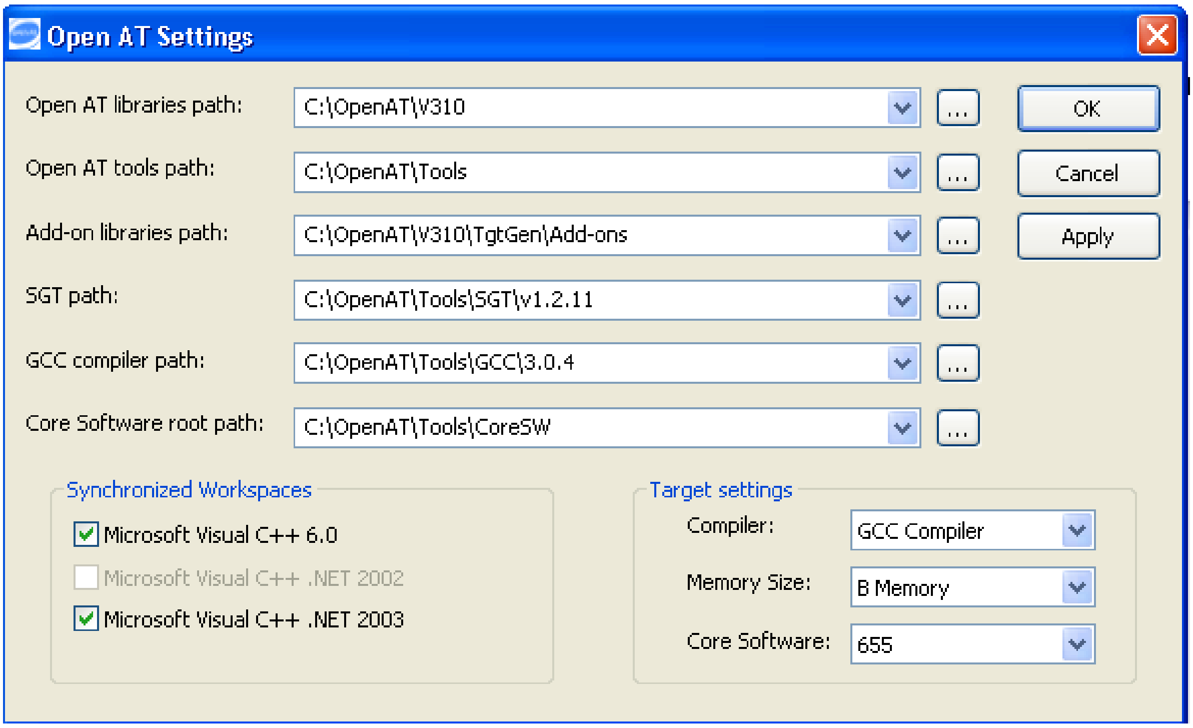
Please refer to the Getting Started guide to know how the Open AT® V3.10 setup runs.

8.2 Sample HTML documentation

To provide more information, and clearer implementation documentation about provided Open AT® samples, a new Sample HTML documentation set is available on Open AT® V3.10 IDE.

8.3 Open AT® Settings

A new graphical application is designed to setup the Open AT® related environment variables and options.



Open AT Settings

Open AT libraries path: C:\OpenAT\V310

Open AT tools path: C:\OpenAT\Tools

Add-on libraries path: C:\OpenAT\V310\TgtGen\Add-ons

SGT path: C:\OpenAT\Tools\SGT\v1.2.11

GCC compiler path: C:\OpenAT\Tools\GCC\3.0.4

Core Software root path: C:\OpenAT\Tools\CoreSW

Synchronized Workspaces

☑ Microsoft Visual C++ 6.0

□ Microsoft Visual C++ .NET 2002

☑ Microsoft Visual C++ .NET 2003

Target settings

Compiler: GCC Compiler

Memory Size: B Memory

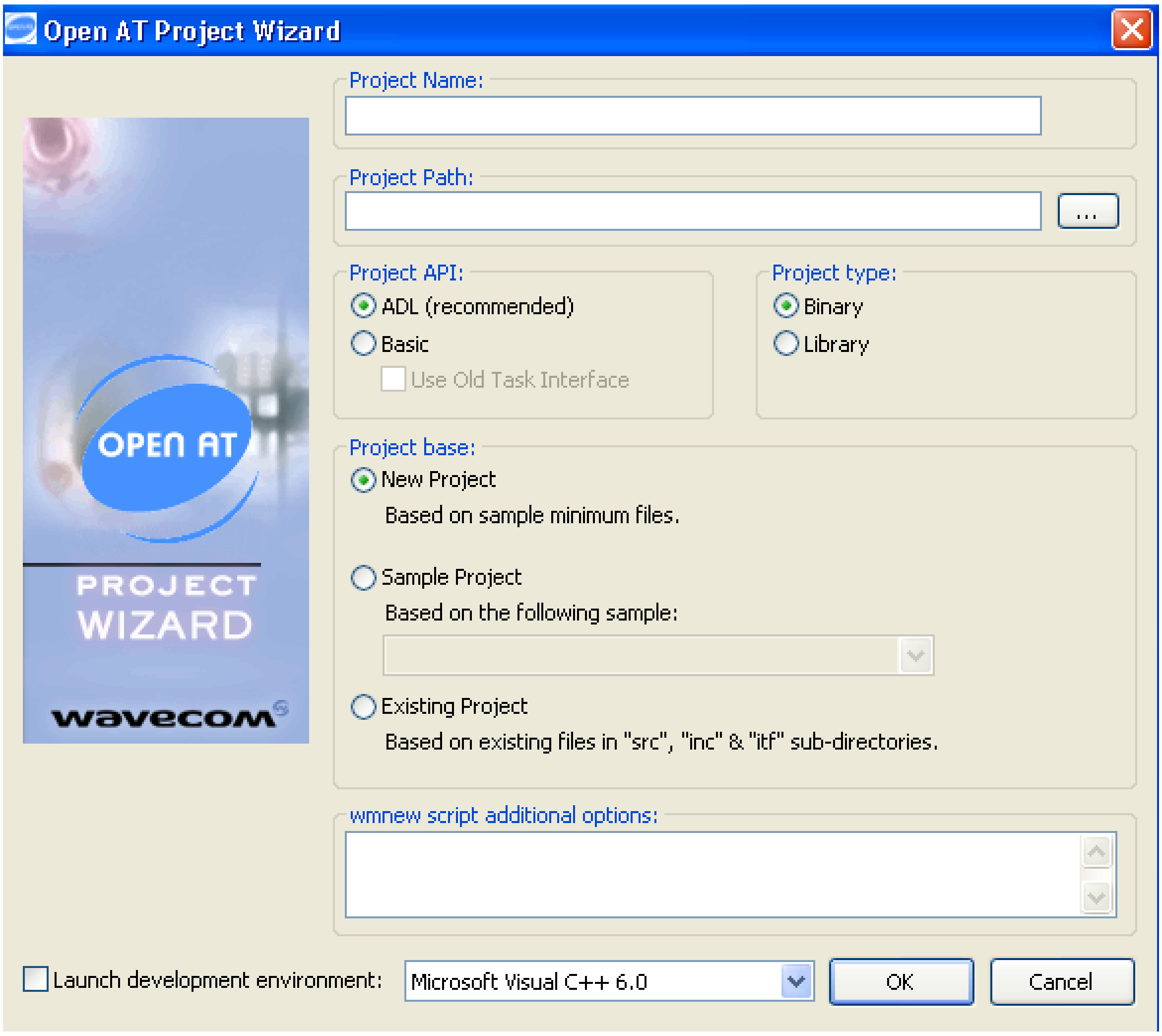
Core Software: 655

Figure 1: Open AT® settings window

Please refer to the Tools Manual to know how the Open AT® Settings application runs.

8.4 Open AT® Project Wizard

The “Wizard” concept of previous Open AT® OS versions is extended in Open AT® OS V3.10b. A generic Project Wizard application is provided, to create all your Remote & Target modes Open AT® projects.



Open AT Project Wizard

Project Name:

Project Path: ...

Project API:
● ADL (recommended)
○ Basic
□ Use Old Task Interface

Project type:
● Binary
○ Library

Project base:
● New Project
Based on sample minimum files.
○ Sample Project
Based on the following sample:
○ Existing Project
Based on existing files in "src", "inc" & "itf" sub-directories.

wmnew script additional options:

□ Launch development environment: Microsoft Visual C++ 6.0

OK Cancel

Figure 2: Project wizard window

The Visual C++ integrated wizards still exist, but these ones just start to Generic Wizard.

The wmnew script still exists (the Project Wizard is in fact a graphical interface which the wmnew script in background), but should not be called anymore from the Cygwin command line.

The compiler & memory settings are moved from wmnew script to wmmake script. It means that the created Open AT® V3.10 projects are generic for all environments and compilers. Environment & compiler configuration is set at compilation time.

8.5 Extended IDE support

Supported Integrated Development Environments may be used to build both Remote & Target mode applications (the wmmake script should not be called any more from the Cygwin command line).

The currently supported IDE are:

- Microsoft Visual C++ 6.0 ;
- Microsoft Visual C++ .NET 2002 ;
- Microsoft Visual C++ .NET 2003.

8.6 Enhanced Remote Task Environment

The Open AT® V3 Remote mode stability is quite better than previous versions one. Limitations due to Real-Time & Compiler behavior differences still exist, but many Remote mode issues were solved in Open AT® OS V3.10.

Graphical interface is also modified, to provided easier RTE mode configuration.

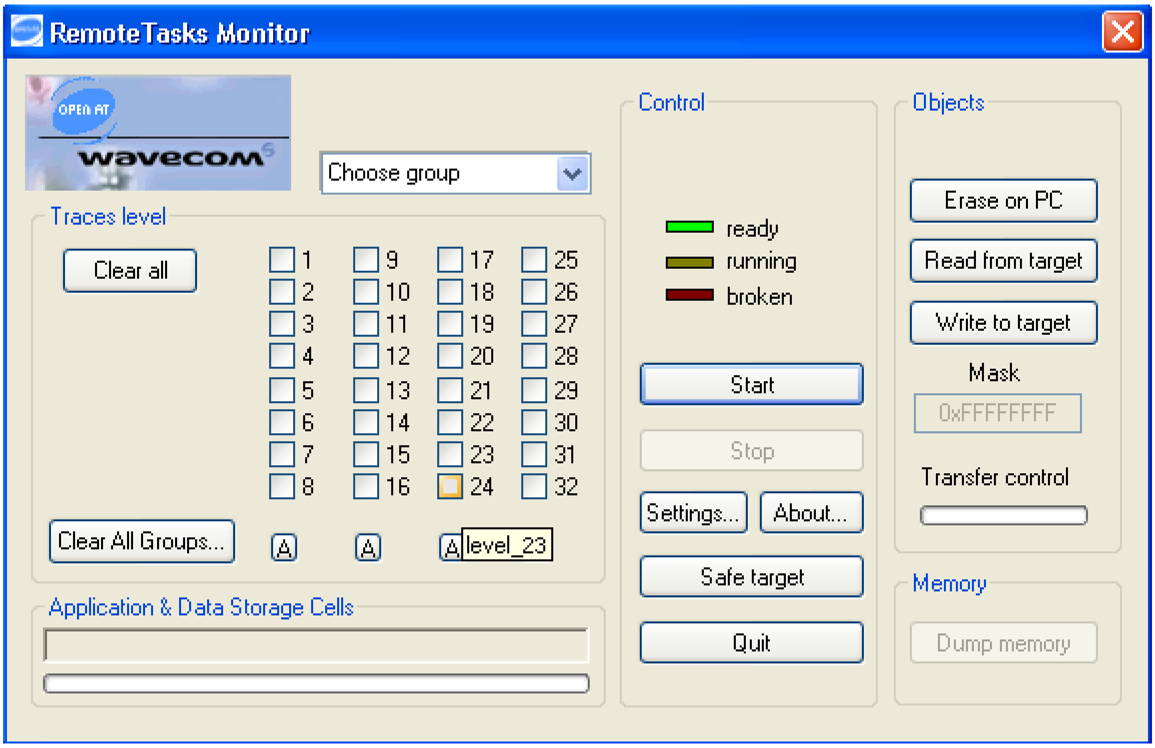


Figure 3: RTE configuration window

9 Specificities between v3.0x and v3.1x

This section lists the new features from the v3.0x and v3.1x version.

9.1 Features from v3.01

9.1.1 No Interface break

This version does not include any interface break in the Open AT API.

Important Note:

In order to run properly an Open AT® application using the FCM service, a 650a or higher firmware has to run on the target.

With applications which do not use the FCM service, existing 650 firmware is still compatible with this version.

9.2 Features from v3.02

9.2.1 Open AT® API

WAP services have been removed from the ADL & Basic APIs

9.3 Features from v3.03

9.3.1 Text mode commands

A new adl_atCmdSendText () function is available since 3.03 API version, and has to be used to handle text mode commands (e.g. AT+CMGW or others). Applications which were using such commands have to be modified.

9.4 Features from v3.04

None

9.5 Features from v3.10

9.5.1 New IO Ports service

A new ADL Port service is now available to monitor hardware (UART1, UART2) or virtual (MUX 27.010 DLC, Bluetooth,) ports on the module, usable to exchange AT commands & FCM data blocks between the WAVECOM OS and the Open AT® OS & external applications.

A new Open AT® virtual port has been created, especially to process only AT commands coming from the embedded application (in order not to be disturbed by any command coming from an external port).

9.5.2 AT commands / Call / GPRS services updates

These services have been modified in order to handle the extended WAVECOM 6.55 Open AT® FW ports capabilities:

- New macros (adl_atSendResponsePort, adl_atSendStdResponsePort & adl_atSendStdResponseExtPort), usable to send responses on a specific port.
- **adl_atCmdCreate** function now returns error codes on specific cases (a bad AT command syntax, or an unknown port); by default, this function also sends AT commands on the new Open AT® virtual port (instead of the UART1 on former versions).
- New functions (**adl_callSetupExt**, **adl_callHangupExt** & **adl_callAnswerExt**), usable to execute call service actions on a specific port, instead of the default virtual Open AT® one.
- New functions (**adl_gprsSetupExt**, **adl_gprsActExt** & **adl_gprsDeactExt**), usable to execute GPRS service actions on a specific port, instead of the default virtual Open AT® one.

9.5.3 Extended A&D storage capabilities

No A&D service API update was performed, but the A&D storage size is now configurable through the **AT+WOPEN=6** command mode. Please refer to the AT commands interface guide for more information.

9.5.4 New RTC service

A new RTC service is now available to provide Open AT® applications with Real Time Clock synchronous information. Please refer to the ADL Development Guide for more information.

9.5.5 Extended Errors management service

The Errors management service has been updated to provide Open AT® applications with the capability to retrieve errors stored in the module's memory (called back-traces), in order to read these ones on a remote computer.

9.5.6 New DTL & GRL add-on libraries

Two new add-on libraries are now provided on the SDK:

- DTL (Data Transfer Layer) which provides an IP link configuration layer (through an API & AT commands set), and a generic data transfer layer API, usable to send/receive a data buffer to/from a remote server, through an FTP, e-mail or SMS bearer.
- GRL (Generic Reporting Layer) which provides a generic report storage API, and a configurable report sending mechanism (scheduler or size limit) in order to regularly send the stored reports to a remote server.

9.5.7 Text commands handling

A new `adl_atCmdSendText` function is available since 3.10 API versions, and has to be used to handle text mode commands (e.g. AT+CMGW or others). Applications which were using such commands have to be modified.

9.6 Features from v3.12

9.6.1 Handle 32/16 memories

This version handles memories 32/4 type B and 32/16 type F; only samples for 32/4 memory are provided in the SDK.

9.6.2 9.7.2 IP plugin WIP

Plug-in WIP is a new plug-in to handle UDP and TCP socket. In this beta version All functionality are not available, and the behaviour is not guaranteed.

9.6.3 Cmux Tool

The last driver for Cmux tool is now available in the Cdrom, and can be installed from the new folder Cmux Tools.

9.6.4 I2C Hard

I2C Hard can be now handled with Open AT® OS.

9.6.5 WTP Plug-in

WTP Plug-in has been removed.

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

www.wavecom.com