



ADL User Guide for Open AT[®] OS v3.13

Revision: 011
Date: May 2007



wavecom[®]
Make it wireless

ADL User Guide for Open AT[®] OS v3.13

Reference: WM_ASW_OAT_UGD_00006


Date: May 3, 2007

Revision: 011

Document History

Index	Date	Versions	
001	06/01/03	Created	
002	04/06/03	Updates for Open AT® 2.10	
003	29/01/04	Updates for Open AT® 2.10a (Q2400 module integration)	
004	21/10/04	Updates for Open AT® 3.0	
005	11/01/05	Updates for Open AT® 3.01	
006	30 th May2005	Updates for Open AT® 3.02	
007	13/06/05	Updates for Open AT® 3.10	
008	October, 2006	Updates for Open AT® 3.12	
009	November 6, 2006	Update	
010	February 23, 2007	Update	
011	May 3, 2007	Small Updates	

Trademarks

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

Copyright

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

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

Overview

This user guide describes the Application Development Layer (ADL). The aim of the Application Development Layer is to ease the development of Open AT[®] embedded applications. It applies to revision Open AT[®] OS v3.13 and higher (until the next version of this document).

Table of Contents

1	INTRODUCTION	11
1.1	Important remarks	11
1.2	References.....	11
1.3	Glossary	11
1.4	Abbreviations	12
2	DESCRIPTION.....	13
2.1	Software Architecture.....	13
2.2	Minimum Embedded Application Code.....	14
2.3	Imported APIs from Open AT® library.....	15
2.4	ADL limitations	15
2.5	UART 2 and GPIOs shared resources.....	15
2.6	Q2501 product external battery charging mechanism GPIO shared resource	16
2.7	SIM Level Shifter and GPO shared resources	16
2.8	Open AT® Memory resources.....	16
2.9	Defined compilation flags	17
2.10	Inner AT commands configuration.....	18
2.11	Open AT® specific AT Commands.....	19
2.11.1	AT+WDWL Command	19
2.11.2	AT+WOPEN Command.....	19
3	API.....	20
3.1	Commands.....	20
3.1.1	Required Header File	20
3.1.2	Unsolicited Responses	20
3.1.3	Responses	22
3.1.4	Incoming AT commands.....	25
3.1.5	Run AT commands	29
3.2	Timers	36
3.2.1	Required Header Files	36
3.2.2	The adl_tmrSubscribe function.....	36
3.2.3	The adl_tmrUnSubscribe function	37
3.2.4	Example.....	38
3.3	Memory Service.....	39
3.3.1	Required Header File.....	39

ADL User Guide for Open AT® OS v3.13

3.3.2	The adl_memGetType function [DEPRECATED].....	39
3.3.3	The adl_memGetInfo function	40
3.3.4	The adl_memGet function	41
3.3.5	The adl_memRelease function.....	42
3.3.1	Example.....	42
3.4	Debug traces	43
3.4.1	Required Header File	43
3.4.2	Debug configuration	43
3.4.3	Full Debug configuration.....	44
3.4.4	Release configuration.....	45
3.5	Flash	46
3.5.1	Required Header File	46
3.5.2	Flash Objects Management	46
3.5.3	The adl_flhSubscribe function	47
3.5.4	The adl_flhExist function	48
3.5.5	The adl_flhErase function	48
3.5.6	The adl_fhWrite function	49
3.5.7	The adl_flhRead function.....	49
3.5.8	The adl_flhGetFreeMem function	50
3.5.9	The adl_flhGetIDCount function	50
3.5.10	The adl_flhGetUsedSize function	51
3.6	FCM Service	52
3.6.1	Required Header File	53
3.6.2	The adl_fcmlsAvailable function	54
3.6.3	The adl_fcmSubscribe function	54
3.6.4	The adl_fcmUnsubscribe function	58
3.6.5	The adl_fcmReleaseCredits function	58
3.6.6	The adl_fcmSwitchV24State function	59
3.6.7	The adl_fcmSendData function	59
3.6.8	The adl_fcmSendDataExt function	61
3.6.9	The adl_fcmGetStatus function	62
3.7	GPIO Service.....	63
3.7.1	Required Header File	63
3.7.2	The adl_ioSubscribe function	63
3.7.3	The adl_ioUnsubscribe function	67
3.7.4	The adl_ioRead function	68
3.7.5	The adl_ioWrite function	68
3.7.6	The adl_io GetProductType function.....	69
3.8	Bus Service.....	70
3.8.1	Required Header File	70
3.8.2	The adl_busSubscribe function	70
3.8.3	The adl_busUnsubscribe function	76
3.8.4	The adl_busRead function	77
3.8.5	The adl_busWrite function	78
3.9	Errors management	81
3.9.1	Required Header File	81
3.9.2	The adl_errSubscribe function	81
3.9.3	The adl_errUnsubscribe function	82

ADL User Guide for Open AT® OS v3.13

3.9.4	The adl_errHalt function	83
3.9.5	The adl_errEraseAllBacktraces function	83
3.9.6	The adl_errStartBacktraceAnalysis function	84
3.9.7	The adl_errGetAnalysisState function.....	84
3.9.8	The adl_errRetrieveNextBacktrace function	84
3.10	SIM Service	86
3.10.1	Required Header File	86
3.10.2	The adl_simSubscribe function	86
3.10.3	The adl_simUnsubscribe function	87
3.10.4	The adl_simGetState function	87
3.11	SMS Service	88
3.11.1	Required Header File	88
3.11.2	The adl_smsSubscribe function.....	88
3.11.3	The adl_smsSend function	90
3.11.4	The adl_smsUnsubscribe function	91
3.12	Call Service	92
3.12.1	Required Header File	92
3.12.2	The adl_callSubscribe function.....	92
3.12.3	The adl_callSetup function	95
3.12.4	The adl_callSetupExt function	95
3.12.5	The adl_callHangup function.....	96
3.12.6	The adl_callHangupExt function.....	96
3.12.7	The adl_callAnswer function	96
3.12.8	The adl_callAnswerExt function	96
3.12.9	The adl_callUnsubscribe function.....	97
3.13	GPRS Service.....	98
3.13.1	Required Header File	98
3.13.2	The adl_gprsSubscribe function	98
3.13.3	The adl_gprsSetup function	101
3.13.4	The adl_gprsSetupExt function	101
3.13.5	The adl_gprsAct function	102
3.13.6	The adl_gprsActExt function	103
3.13.7	The adl_gprsDeact function.....	104
3.13.8	The adl_gprsDeactExt function.....	104
3.13.9	The adl_gprsGetCidInformations function	105
3.13.10	The adl_gprsUnsubscribe function	106
3.13.11	The adl_gprsIsAnIPAddress function.....	107
3.13.12	Example.....	108
3.14	Application Safe Mode Service	110
3.14.1	Required Header File	110
3.14.2	The adl_safeSubscribe function.....	110
3.14.3	The adl_safeUnsubscribe function.....	112
3.14.4	The adl_safeRunCommand function.....	113
3.15	AT Strings Service	114
3.15.1	Required Header File	114
3.15.2	The adl_strID_e type.....	114
3.15.3	The adl_strGetID function.....	115
3.15.4	The adl_strGetIDExt function.....	115

ADL User Guide for Open AT® OS v3.13

3.15.5	The adl_strIsTerminalResponse function	116
3.15.6	The adl_strGetResponse function	116
3.15.7	The adl_strGetResponseExt function	117
3.16	Application & Data storage Service.....	118
3.16.1	Required Header File	118
3.16.2	The adl_adSubscribe function	118
3.16.3	The adl_adUnsubscribe function	119
3.16.4	The adl_adEventSubscribe function.....	120
3.16.5	The adl_adEventHdlr_f call-back type	120
3.16.6	The adl_adEventUnsubscribe function	122
3.16.7	The adl_adWrite function	122
3.16.8	The adl_adInfo function.....	123
3.16.9	The adl_adFinalise function	123
3.16.10	The adl_adDelete function	124
3.16.11	The adl_adInstall function	124
3.16.12	The adl_adRecompact function	125
3.16.13	The adl_adGetState function	126
3.16.14	The adl_adGetCellList function	127
3.16.15	The adl_adFormat function.....	127
3.16.16	Example.....	128
3.17	GPS Service	132
3.17.1	Required Header File	132
3.17.2	GPS Data structures	132
3.17.3	The adl_gpsSubscribe function	134
3.17.4	The adl_gpsUnsubscribe function	135
3.17.5	The adl_gpsGetState function	136
3.17.6	The adl_gpsGetPosition function	136
3.17.7	The adl_gpsGetSpeed function.....	137
3.17.8	The adl_gpsGetSatView function	137
3.18	AT/FCM IO Ports Service	138
3.18.1	Required Header File	138
3.18.2	AT/FCM IO Ports	138
3.18.3	Ports test macros.....	139
3.18.4	The adl_portSubscribe function.....	140
3.18.5	The adl_portUnsubscribe function.....	141
3.18.6	The adl_portIsAvailable function	142
3.18.7	The adl_portGetSignalState function	142
3.18.8	The adl_portStartSignalPolling function	143
3.18.9	The adl_portStopSignalPolling function.....	145
3.19	RTC Service	146
3.19.1	Required Header File	146
3.19.2	RTC service types	146
3.19.3	The adl_rtcGetTime function	148
3.19.4	The adl_rtcConvertTime function.....	148
3.19.5	The adl_rtcDiffTime function	149
3.20	DAC Service	150
3.20.1	Required Header File	150
3.20.2	The adl_dacSubscribe function	150

ADL User Guide for Open AT® OS v3.13

3.20.3	The adl_dacUnsubscribe function	151
3.20.4	The adl_dacWrite function	151
3.20.5	Example.....	152
4	ERROR CODES	153
4.1	General error codes	153
4.2	Specific FCM service error codes	153
4.3	Specific flash service error codes	154
4.4	Specific GPRS service error codes.....	154
4.5	Specific GPS service error codes.....	154
4.6	Specific A&D storage service error codes	154

List of Figures

Figure 1: Software architecture	13
Figure 2: Open AT RAM mapping, with adl_memInfo_t structure fields names	41
Figure 3: Flow Control Manager representation	52
Figure 4: LCD_EN Address Setup chronogram	75

1 Introduction

1.1 Important remarks

- It is strongly recommended before reading this document, to read the Open AT® Basic Development Guide and specifically the Introduction (chapter 1) and the Description (chapter 2) to have a better overview of what Open AT® is about.
- The ADL library and the standard embedded Open AT® API layer must not be used in the same application code. As ADL APIs will encapsulate commands and trap responses, applications may enter error mode if synchronization is no longer guaranteed.

1.2 References

- I. Open AT® Basic Development Guide for Open AT® OS v3.13 (ref WM_ASW_OAT_UGD_002 revision 15).

1.3 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.
IDE	Integrated Development Environment
Target	Open AT® compatible product supporting an Embedded Application.

ADL User Guide for Open AT® OS v3.13

Introduction

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.

1.4 Abbreviations

A&D	Application & Data
ADL	Application Development Layer
API	Application Programming Interface
APN	Access Point Name
CID	Context identifier
CPU	Central Processing Unit
DAC	Digital Analog Converter
GPRS	General Packet Radio Service
GGSN	Gateway GPRS Support Node
IP	Internet Protocol
IR	Infrared
KB	Kilobyte
MS	Mobile Station
OS	Operating System
PDU	Protocol Data Unit
PDP	Packet Data Protocol
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

2 Description

2.1 Software Architecture

The Application Development Layer software library, based on the standard embedded Open AT® API layer, is included in the Wavecom library since Open AT® release 2.00 (as defined in section 2.1.1 "Software Organization" of the Basic Development Guide).

The aim of the ADL is to provide a high level interface to the Open AT® software developer. The ADL supplies the mandatory software skeleton for an embedded application, for instance the message parser (see 2.2: "Minimum Embedded Application Code" of Open AT® Basic Development Guide) and some messages states machines for given complex services (SIM service, SMS service...).

Thus, the Open AT® software developer can concentrate on the contents of his application. He or she simply has to write the callback functions associated to each service he or she wants to use.

Therefore the software supplied by Wavecom contains the items listed below:

- ADL software library wmadl.lib,
- A set of header files (.h) defining the ADL API functions,
- Source code samples,

It relies on the following software architecture:

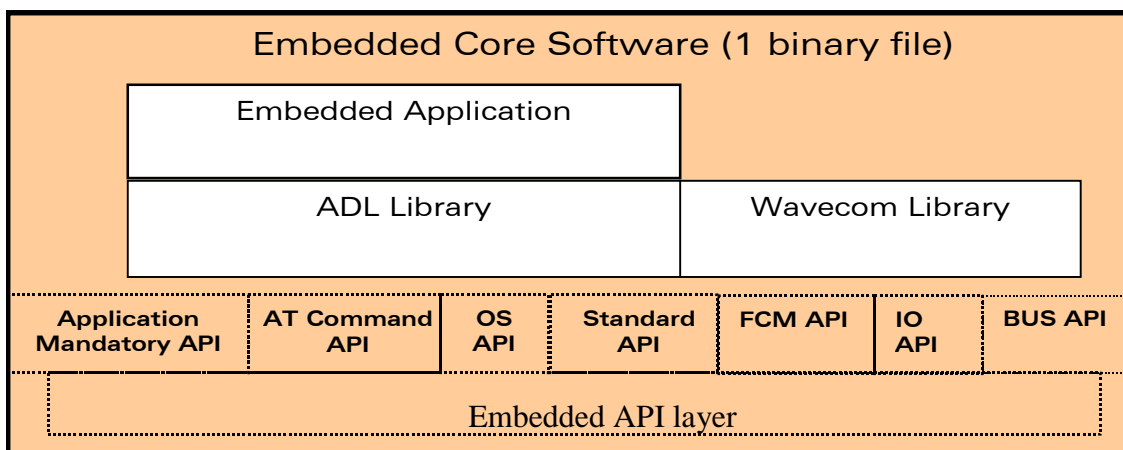


Figure 1: Software architecture

2.2 Minimum Embedded Application Code

The minimum embedded application code requested for ADL is the following:

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

And the entry point to the ADL code is the main function `adl_main()`:

```
/*main function */  
void adl_main(adl_apmInitType_e InitType) {}
```

The `adl_InitType_e` is described below:

```
typedef enum  
{  
    ADL_INIT_POWER_ON,          // Normal power on  
    ADL_INIT_REBOOT_FROM_EXCEPTION, // Reboot after an embedded application  
                                   exception  
    ADL_INIT_DOWNLOAD_SUCCESS,   // Reboot after a successful install process  
                                   (cf. adl_adInstall API)  
    ADL_INIT_DOWNLOAD_ERROR // Reboot after an error in install process (cf.  
                                   adl_adInstall API)  
} adl_InitType_e;
```

`wm_apmCustomStack` and `wm_apmCustomStackSize` are two mandatory variables, used to define the application call stack size (see § "Minimum Embedded Application Code" and § "Mandatory Functions" of Open AT® Basic Development Guide).

For more information about AT command size, downloading, memory limitation or security, please see § "Description" in the Open AT® Basic Development Guide.

Important note:

Please keep in mind that the `adl_main` function is NOT like a standard "C" main function, since the application does not end as soon as `adl_main` returns. An Open AT® application is stopped only if the "AT+WOPEN=0" command is used. The `adl_main` function is just the application entry point, and has to subscribe to some services and events to go further. Moreover, since the whole software is protected by a watchdog mechanism, the application cannot use infinite loops, otherwise the module will reset after an 8-second security delay.

2.3 Imported APIs from Open AT® library

The following APIs can be used as in Open AT® standard applications. The required headers are already included in the global ADL header file. The APIs available in this way are listed below:

- Standard API (defined in `wm_stdio.h` file) ;
- List API (defined in `wm_list.h` file) ;
- Sound API (defined in `wm_snd.h` file) ;

Please refer to Open AT® Basic Development Guide for a description of these APIs.

2.4 ADL limitations

- Concatenated commands (for example "AT+CREG?;+CGMR") may be used from the embedded application, but not from external applications while ADL is running. If subscribed commands are concatenated, command handlers will not be notified.
- Since ADL uses its own internal process of the +WIND indications, the current value of the AT+WIND command may not be the same when the AT+WOPEN command state is 0 or 1.

2.5 UART 2 and GPIOs shared resources

When the product's second UART is used (started with the AT+WMFM command, or reserved for the GPS component in internal mode on a Q25X1-based product), some of the GPIOs are no longer available for the embedded application. The impacted GPIOs depend on product type, as described below:

WAVECOM Wireless CPU® series	Unavailable GPIOs
Q24X6	<ul style="list-style-type: none"> • GPIO 0 and GPIO 5 • GPO 2 • GPI
Q24X0	<ul style="list-style-type: none"> • GPIO 0 and GPIO 5 • GPO 2 • GPI
Q25X1	<ul style="list-style-type: none"> • GPIO 0 and GPIO 5 • GPO 2 • GPI
P32X6	<ul style="list-style-type: none"> • GPIO 2 • GPI
Q31X6	<ul style="list-style-type: none"> • GPIO 4 and GPIO 5 • GPO 2 • GPI
P51X6	<ul style="list-style-type: none"> • GPIO 5 • GPO 0 and GPO 1

2.6 Q2501 product external battery charging mechanism GPIO shared resource

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

2.7 SIM Level Shifter and GPO shared resources

If any other feature than "SIM3VONLY" 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 Wireless CPU® series	Unavailable GPO
Q24X6	GPO0
Q24X0	GPO0
Q25X1	GPO1
Q24 CLASSIC	GPO0
Q24 PLUS	GPO0
Q24 AUTO	GPO0
Q24 EXTENDED	GPO0

2.8 Open AT® Memory resources

The available memory resources for the Open AT® applications depends on the product memory size:

For products with 32-Mbit flash size and 4Mbit RAM size:

768 Kbytes of ROM (application code)
(configurable; see AT+WOPEN command)

128 Kbytes of RAM

128 Kbytes of Flash Object Data

768 Kbytes of Application & Data Storage Volume
(configurable; see AT+WOPEN command)

ADL User Guide for Open AT® OS v3.13

Description

For products with 32-Mbit flash size and 16Mbit RAM size:

768 Kbytes of ROM (application code)
(configurable; see AT+WOPEN command)

1664 Kbytes of RAM

128 Kbytes of Flash Object Data

768 Kbytes of Application & Data Storage Volume
(configurable; see AT+WOPEN command)

The total available flash space for both Open AT® application place and A&D storage place is 1536 Kbytes. This space is shared between the two places.

The maximum A&D storage place size is 1280 Kbytes (1.2 Mbytes: usable for Wavecom Core Software upgrade capability); the Open AT® application maximum size, in this case, will be 256 Kbytes.

The minimum A&D storage place size is 0 Kbytes (usable for applications with huge hard coded data); the Open AT® application maximum size will, in this case, be 1.5 Mbytes.

Warning:

Any A&D size change will lead to this area format process (some seconds on start-up; all A&D cells data will be erased).

2.9 Defined compilation flags

The Open AT(R) IDE defines some compilation flags, related to the chosen generation environment. Please refer to the Tools Manual for more information.

2.10 Inner AT commands configuration

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

AT Command	Fixed value
AT+CMEE	1
AT+WIND	All indications (*)
AT+CREG	2
AT+CGREG	2
AT+CRC	1
AT+CGEREP	2
ATV	1
ATQ	0

(*) All +WIND unsolicited indications are always required by the ADL library. The "+WIND: 3" indication (product reset) will be enabled only if the external application required this.

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 Open AT® ADL applications, without requiring them to be sent (using the corresponding configuration command).

Important Warning:

User is strongly advised against modifying the current values of these commands from any Open AT® application. Wavecom would not guarantee ADL correct processing if these values are modified by any embedded application.

External applications may modify these AT commands' parameter values without any constraints. These commands and related unsolicited results behavior is the same with our without a running ADL application.

If error codes or unsolicited results related to these commands are subscribed and then forwarded by an ADL application to an external application, these results will be displayed for the external application only if this latter has required them using the corresponding AT commands (same behavior as the Wavecom AT firmware without a running ADL application).

When ATQ1 mode is running, though terminal responses are not sent to the external application, they are always received from the firmware in the embedded application.

2.11 Open AT® specific AT Commands

See document WM_ASW_OAT_UGD_00044, AT Commands Interface Guide for Open AT® Firmware v6.57c.

2.11.1 AT+WDWL Command

The AT+WDWL command is usable to download .dwl files through the serial link, using the 1K Xmodem protocol.

Dwl files may be Wavecom Core software updates, Open AT® application binaries, or E2P configuration files.

By default this command is not pre-parsed (it cannot be filtered by the Open AT® application), except if the Application Safe Mode service is used.

Note:

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

2.11.2 AT+WOPEN Command

The AT+WOPEN command is used to control Open AT® applications mode & parameters.

Parameters:

- 0 Stop the application (the application will be stopped on all product resets)
- 1 Start the application (the application will be started on all product resets)
- 2 Get the Open AT® libraries versions
- 3 Erase the objects flash of the Open AT® Embedded Application (allowed only if the application is stopped)
- 4 Erase the Open AT® Embedded Application (allowed only if the application is stopped)
- 5 Suspend the Open AT® application, until the AT+WOPENRES command is used, or an hardware interruption occurs
- 6 Configures the Application & Data storage place and Open AT® application place sizes.

Note:

Refer to the document [Ref II] for more information about this command

By default this command is not pre-parsed (it cannot be filtered by the Open AT® application), except if the Application Safe Mode service is used.

3 API

3.1 Commands

3.1.1 Required Header File

The header file for the functions dealing with AT commands is:

`adl_at.h`

3.1.2 Unsolicited Responses

An unsolicited response is a string sent by the Wavecom Core Software to applications in order to provide them with unsolicited event information (ie. not in response to an AT command).

ADL applications may subscribe to an unsolicited response in order to receive the event in the handler provided.

Once an application has subscribed to an unsolicited response, it will have to unsubscribe from it to stop the callback function being executed every time the matching unsolicited response is sent from the Wavecom Core Software.

Multiple subscriptions: each unsolicited response may be subscribed several times. If an application subscribes to an unsolicited response with handler 1 and then subscribes to the same unsolicited response with handler 2, every time the ADL parser receives this unsolicited response handler 1 and then handler 2 will be executed.

3.1.2.1 The `adl_atUnSoSubscribe` function

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.

- **Prototype**

```
s16 adl_atUnSoSubscribe ( ascii * UnSostr,  
                        adl_atUnSoHandler_t UnSohdl )
```

- **Parameters**

UnSostr:

The name (as a string) of the unsolicited response we want to subscribe to. This parameter can also be set as an `adl_rspID_e` response ID. Please refer to §3.15 for more information.

UnSohdl:

A handler to the callback function associated to the unsolicited response.

The callback function is defined as follow:

```
typedef bool (* adl_atUnSoHandler_t) (adl_atUnsolicited_t *)
```

The argument of the callback function will be a 'adl_atUnsolicited_t' structure, holding the unsolicited response we subscribed to.

The 'adl_atUnsolicited_t' structure defined as follow (it is declared in the adl_at.h header file):

```
typedef struct
{
    adl_strID_e RspID; // Standard response ID
    adl_atPort_e Dest; // Unsolicited response destination port
    ul6 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;
```

The RspID field is the parsed standard response ID if the received response is a standard response. Refer to §3.15 for more information.

The Dest field is the unsolicited response original destination port. If it is set to ADL_PORT_NONE, unsolicited response is required to be broadcasted on all ports.

The return value of the callback function will have to be TRUE if the unsolicited string is to be sent to the external application (on the port indicated by the Dest field, if not set to ADL_PORT_NONE, otherwise on all ports), and FALSE otherwise.

Note that in the case of several handlers associated to the same unsolicited response, all of them have to return TRUE for the unsolicited response to be sent to the external application.

- **Returned values**

- OK on success
 - ERROR if an error occurred.

3.1.2.2 The adl_atUnSoUnSubscribe function

This function unsubscribes from an unsolicited response and its handler.

- **Prototype**

```
s16 adl_atUnSoUnSubscribe ( ascii * UnSostr,
                           adl_atUnSoHandler_t UnSohdl )
```

- **Parameters**

UnSostr:

The string of the unsolicited response we want to unsubscribe to.

UnSohdl:

The callback function associated to the unsolicited response.

- **Returned values**

OK if the unsolicited response was found,
ERROR otherwise.

3.1.2.3 Example

```
/* callback function */
bool Wind4_Handler(adl_atUnsolicited_t *paras)
{
    /* Unsubscribe to the '+WIND: 4' unsolicited response */
    adl_atUnSoUnSubscribe("+WIND: 4",
                          (adl_atUnSoHandler_t)Wind4_Handler);
    adl_atSendResponse(ADL_AT_RSP, "\r\nWe have received a Wind 4\r\n");
    /* We want this response to be sent to the external application,
     * so we return TRUE */
    return TRUE;
}

/*main function */
void adl_main(adl_InitType_e adlInitType)
{
    /* Subscribe to the '+WIND: 4' unsolicited response */
    adl_atUnSoSubscribe("+WIND: 4",
                       (adl_atUnSoHandler_t)Wind4_Handler);
}

```

3.1.3 Responses

3.1.3.1 The adl_atSendResponse function

This function sends the provided text to any external application connected to the required port, as a response, an unsolicited response or an intermediate response, according to the requested type.

- **Prototype**

```
void adl_atSendResponse ( u16 Type,
                          ascii * String )
```

- **Parameters**

Type:

This parameter is composed of the response type, and the destination port where to send the response. The type & destination combination has to be done with the following macro:

```
ADL_AT_PORT_TYPE ( _port, _type )
```

The `_port` argument has to be a defined value of the `adl_atPort_e` type, and this required port has to be available (cf. the AT/FCM port Service); sending a response on an Open AT® the GSM or GPRS based port will have no effects). Note that with the `ADL_AT_UNE` type value, if the `ADL_AT_PORT_TYPE` macro is not used, the unsolicited response will be broadcasted on all currently opened ports.

If the `ADL_AT_PORT_TYPE` macro is not used with the `ADL_AT_RSP` & `ADL_AT_INT` types, responses will be by default sent on the UART 1 port. If this port is not opened, responses will not be displayed.

The `_type` argument has to be one of the values defined below:

- **ADL_AT_RSP:**
Terminal response (has to end an incoming AT command).
A destination port has to be specified.
Sending such a response will flush all previously buffered unsolicited responses on the required port.
- **ADL_AT_INT:**
Intermediate response (text to display while an incoming AT command is running).
A destination port has to be specified.
Sending such a response will just display the required text, without flushing all previously buffered unsolicited responses on the required port.
- **ADL_AT_UNE:**
Unsolicited response (text to be displayed out of a currently running command process).
For the required port (if any) or for each currently opened port (if the `ADL_AT_PORT_TYPE` macro is not used), if an AT command is currently running (ie. the command was sent by the external application, but this command answer has not been sent back yet), any unsolicited response will automatically be buffered, until a terminal response is sent on this port.

String:

The text to be sent.

Please note that this is exactly the text string to be displayed on the required port (i.e. all carriage return & line feed characters (“\r\n” in C language) have to be sent by the application itself).

3.1.3.2 The `adl_atSendStdResponse` function

This function sends the provided standard response to the required port, as a response, an unsolicited response or an intermediate response, according to the requested type.

- **Prototype**

```
void adl_atSendStdResponse ( u8 Type,  
                             adl_strID_e RspID )
```

- **Parameters**

Type:

Same use as the `adl_atSendResponse` Type parameter.

RspID:

Standard response ID to be sent (see §3.15 for more information).

3.1.3.3 The `adl_atSendStdResponseExt` function

This function sends the provided standard response with an argument to the required port, as a response, an unsolicited response or an intermediate response, according to the requested type.

- **Prototype**

```
void adl_atSendStdResponseExt ( u8 Type,  
                                adl_strID_e RspID,  
                                u32 arg )
```

- **Parameters**

Type:

Same use as the `adl_atSendResponse` Type parameter.

RspID:

Standard response ID to be sent (see §3.15 for more information).

arg:

Standard response argument. According to response ID, this argument should be an `u32` integer, or an `ascii * string`.

3.1.3.4 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)
```

3.1.4 Incoming AT commands

An ADL application may subscribes to an AT command string, in order to receive events each time an external application sends this AT command on one of the module's ports.

Once the application has subscribed to a command, it will have to unsubscribe to stop the callback function being executed every time this command is sent by an external application.

Multiple subscriptions: if an application subscribes to a command with a handler and subscribes then to the same command with another handler, every time this command is sent by the external application both handlers will be successively executed (in the subscription order).

3.1.4.1 The `adl_atCmdSubscribe` function

This function subscribes to a specific command with an associated callback function, so that next time the required command is sent by an external application, the callback function will be executed.

- **Prototype**

```
s16 adl_atCmdSubscribe ( ascii * Cmdstr,
                        adl_atCmdHandler_t Cmdhdl,
                        u16 Options )
```

- **Parameters**

Cmdstr:

The string (name) of the command we want to subscribe to. Since this service only handles AT commands, this string has to begin with the "AT" characters.

Cmdhdl:

The handler of the callback function associated to the command.

The callback function is defined as follows:

```
typedef void (* adl_atCmdHandler_t) (adl_atCmdPreParser_t *)
```

The argument of the callback function will be an 'adl_atCmdPreParser_t' structure holding the command we subscribed to.

The "adl_atCmdPreParser_t" structure is defined as follow (it is declared in the adl_at.h header file):

```
typedef struct
{
    u16          Type;           // Incoming Command 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;
```

This structure members are defined below:

- o Type:
Incoming command type (will be one of the required ones at subscription time), detected by the ADL pre-processing.
- o NbPara:
Non NULL parameters number (if Type is **ADL_CMD_TYPE_PARA**), or 0 (with other type values).
- o Port:
Port on which the command was sent by the external application.
- o ParaList:
Parameters list (if Type is **ADL_CMD_TYPE_PARA**). Each parameter may be accessed by the **ADL_GET_PARAM(_p, _i)** macro, where **_p** is the command handler parameter (**adl_atCmdPreParser_t * pointer**), and **_i** is the parameter index (from 0 to NbPara - 1). NbPara is the number of arguments received and it is a number between the minimum arguments number ('a') and the maximum arguments number ('b') (eg. a=1, b=5 and "AT+MYCMD=0,1,2", **_i** can be between 0 and 3 - 1 = 2).
If a string parameter is provided (eg. AT+MYCMD="string"), the quotes will be removed from the returned string (eg. **ADL_GET_PARAM(para,0)** will return "string" (without quotes) in this case).
If a parameter is not provided (eg. AT+MYCMD=,1), the matching list element will be set to NULL (eg. **ADL_GET_PARAM(para,0)** will return NULL in this case).
StrLength, StrData:
Incoming command string length and value.

ADL User Guide for Open AT® OS v3.13

API

Options:

This flag combines with an arithmetic 'OR' ('|' in C language) the following information:

- Minimum arguments number 'a' stored in the least significant byte (as in 0x000a); only if the **ADL_CMD_TYPE_PARA** type is required.
- Maximum arguments number 'b' stored in the second least significant byte (as in 0x00b0); only if the **ADL_CMD_TYPE_PARA** type is required.
- A combination of the available types:

Command type	Value	Meaning
ADL_CMD_TYPE_PARA	0x0100	'AT+cmd=x, y' is allowed. The execution of the callback function also depends on whether the number of argument is valid or not.
ADL_CMD_TYPE_TEST	0x0200	'AT+cmd=?' is allowed.
ADL_CMD_TYPE_READ	0x0400	'AT+cmd?' is allowed.
ADL_CMD_TYPE_ACT	0x0800	'AT+cmd' is allowed.
ADL_CMD_TYPE_ROOT	0x1000	All commands starting with the subscribed string are allowed. The handler will only receive the whole AT string (no parameters detection). For example, if the "at-" string is subscribed, all "at-cmd1", "at-cmd2", etc. strings will be received by the handler. but the only string "at-" is not received..

Incoming commands which are matching with these options combinations will lead to the callback function execution. If options do not match, the command will be forwarded to be processed by the Wavecom Core Software.

- **Returned values**

OK
ERROR if an error occurred.

- **Important note about incoming concatenated command**

ADL is able to recognize and process concatenated commands coming from external applications (Please refer to AT Commands Interface Guide for more information on concatenated commands syntax).

In this case, this port enters a specific concatenation processing mode, which will end as soon as the last command replies OK, or if one of the commands used replies with an ERROR code. During this specific mode, all other external command requests will be refused on this port: any external application connected on this port will receive a "+CME ERROR: 515" code if it tries to

ADL User Guide for Open AT® OS v3.13

API

send another command. The embedded application can continue using this port for its specific processes, but it has to be careful to send one (at least one, and only one) terminal response for each subscribed command.

If a subscribed command is used in a concatenated command string, the corresponding handler will be notified as if the command was used alone.

In order to handle the concatenation mechanism properly, each subscribed command has to finally answer with a single terminal response (**ADL_STR_OK**, **ADL_STR_ERROR** or other ones), otherwise the port will stay in concatenation processing mode, refusing all internal and external commands on this one.

3.1.4.2 The `adl_atCmdUnSubscribe` function

This function unsubscribes from a command and its handler.

- **Prototype**

```
s16 adl_atCmdUnSubscribe ( ascii * Cmdstr,  
                           adl_atCmdHandler_t Cmdhdl )
```

- **Parameters**

Cmdstr:

The string (name) of the command we want to unsubscribe from.

Cmdhdl:

The handler of the callback function associated to the command.

- **Returned values**

OK if the command was found,
ERROR otherwise.

3.1.4.3 Example

```

/* callback function */
void atabc_Handler(adl_atCmdPreParser_t *paras)
{
    /* Unsubscribe (therefore the command at+abc will only work once) */
    adl_atCmdUnSubscribe("at+abc",
        (adl_atCmdHandler_t)atabc_Handler);
    if(paras->Type == ADL_CMD_TYPE_READ)
        adl_atSendResponsePort(ADL_AT_RSP, paras->Port,
            "\r\nhandling at+abc?\r\n");
    else if(paras->Type == ADL_CMD_TYPE_TEST)
        adl_atSendResponsePort(ADL_AT_RSP, paras->Port,
            "\r\nhandling at+abc=?\r\n");
    else if(paras->Type == ADL_CMD_TYPE_ACT)
        adl_atSendResponsePort(ADL_AT_RSP, paras->Port,
            "\r\nhandling at+abc\r\n");
    else if(paras->Type == ADL_CMD_TYPE_PARA)
    {
        ascii buffer[25];
        wm_strcpy(buffer, "\r\nhandling at+abc=");
        wm_strcat(buffer, ADL_GET_PARAM(paras, 0));
        wm_strcat(buffer, "\r\n");
        adl_atSendResponsePort(ADL_AT_RSP, paras->Port, buffer);
    }
    adl_atSendResponsePort(ADL_AT_RSP, paras->Port, "\r\nOK\r\n");
}

/*main function */
void adl_main(adl_InitType_e adlInitType)
{
    /* Subscribe to the 'at+abc' command in all modes and accepting 1 parameter */
    adl_atCmdSubscribe("at+abc",
        (adl_atCmdHandler_t)atabc_Handler,
        ADL_CMD_TYPE_TEST/ADL_CMD_TYPE_READ/
        ADL_CMD_TYPE_ACT/ADL_CMD_TYPE_PARA/0x0011);
}

```

3.1.5 Run AT commands

3.1.5.1 The adl_atCmdCreate function

This function sends a command on the required port and allows the subscription to several responses and intermediate responses with one associated callback function, so that when any of the responses or intermediates responses we subscribe to is received by the ADL parser, the callback function will be executed.

- **Prototype**

```
s8 adl_atCmdCreate (  ascii * Cmdstr,  
                    u16 Rspflag,  
                    adl_atRspHandler_t Rsphdl,  
                    [...,]  
                    NULL)
```

- **Parameters**

Cmdstr:

The string (name) of the command we want to send. If the string does not end with the CR character (“\r” in C language), it will be added by ADL. In case of text mode commands (as +CMGW for example), the text end character has to be the ^Z (“\x1A” in C language) character.

Rspflag:

This parameter is composed of the unsubscribed responses destination flag, and the port to which the command is to be sent. The flag & destination combination has to be done with the following macro :

```
ADL_AT_PORT_TYPE ( _port, _flag )
```

The `_port` argument has to be a defined value of the `adl_atPort_e` type, and this required port has to be available (cf. the AT/FCM port Service). If this port is not available, or if it is a GSM or GPRS based one, the command will not be executed.

The `_flag` argument has to be one of the values defined below:

If set to TRUE: the responses and intermediate responses of the sent command that are not subscribed (i.e. not listed in the `adl_atCmdCreate` function arguments) will be sent on the required port.

If set to FALSE they will not be sent to the external application.

If the `ADL_AT_PORT_TYPE` macro is not used, by default the command will be sent to the Open AT® virtual port (see next paragraph for more information about At command ports).

Rsphdl:

Handler of the callback function associated to all the responses and intermediate responses subscribed in the `adl_atCmdCreate` function call.

Note that the callback function will be called one time on each response line sent back by the Wavecom Core Software. For example, since the “AT+CGMR” commands replies with two lines (Software version response, and then “OK” response), the response handler will be called two times if all responses are subscribed.

ADL User Guide for Open AT® OS v3.13

API

The callback function is defined as follows:

```
typedef bool (* adl_atRspHandler_t) (adl_atResponse_t *)
```

The argument of the callback function will be an 'adl_atResponse_t' structure holding the received response.

The 'adl_atResponse_t' structure is defined as follows (declared in the adl_at.h header file):

```
typedef struct
{
    adl_strID_e          RspID;
    adl_atPort_e        Dest;
    u16                 StrLength;
    ascii               StrData[1];
} adl_atResponse_t;
```

This structure members are defined below:

- o RspID:
Detected standard response ID if the received response is a standard response. See § 3.15 for more information.
- o Dest:
Port on which the command has been executed; it is also the destination port where the response will be forwarded if the handler returns TRUE.
- o StrLength & StrData:
Response string length & value.

The return value of the callback function has to be TRUE if the response string has to be sent to the port provided, FALSE otherwise.

This allows a variable number of arguments, where ADL expects a list of responses and intermediate responses to subscribe to. When the command is executed, its responses are compared with each item of this list. For each matching response, the callback function is called; the other responses are processed as required by the RspFlag parameter.

Note that the last element of the list must be NULL.

If the list is set to only 2 elements "*" and NULL, when the command will be sent, all the responses and intermediate responses received by the ADL parser will execute the callback function until a terminal response is received by the ADL parser.

The elements of this response list can also be set as an adl_rsp_ID_e response ID. Please refer to §3.15 for more information.

ADL User Guide for Open AT® OS v3.13

API

- **Returned values**

- OK on success (the command will be executed on the required port as soon as possible)
- ADL_RET_ERR_PARAM on parameter error (NULL command string, or "a" command required (this command cannot be used with the adl_atCmdCreate function))
- ADL_RET_ERR_UNKNOWN_HDL if the required port is not available.

- **Note 1**

This function can be associated with the `adl_atCmdSubscribe` one for filtering or spying any intermediate response or response of a specific command send by the external application. (See the example below).

- **Note 2**

Commands sent through the `adl_atCmdCreate` function are directly submitted to the Wavecom Core Software AT interface: they cannot be filtered by an `adl_atCmdSubscribe` mechanism. The `adl_atCmdSubscribe` function filters only the commands coming from external module ports.

- **Note 3**

This function can be used to send "Text Mode" commands (such as "AT+CMGW", etc.); in order to provide the text related to this command, the `adl_atCmdSendText` function has then to be used as soon as the prompt ('> ') response is received in the response handler.

Any further calls to `adl_atCmdCreate` on this port will just store the required command, in order to send these as soon as the running "Text Mode" command has ended.

- **Example**

In the following example, we spy the ATD command by sending the AT+CLCC command every time a subscribed intermediate response or response is received by the ADL parser

```

/* atd responses callback function */
s16 ATD_Response_Handler(adl_atResponse_t *paras)
{
    /* None of the response of the 'at+clcc' command is subscribed but
    because
    * the 2nd argument is set to TRUE, all will be sent to the external
    application */
    adl_atCmdCreate("at+clcc",
                    ADL_AT_PORT_TYPE ( paras->Port, TRUE),
                    (adl_atRspHandler_t)NULL,
                    NULL);

    return TRUE;
}

/* atd callback function */
void ATD_Handler(adl_atCmdPreParser_t *paras)
{
    adl_atCmdUnSubscribe("atd",
                        (adl_atCmdHandler_t) ATD_Handler);
    /* We unsubscribe the command so that when we resend the command
    * it won't be received by the ADL parser anymore.*/
    /* We resend the command (for the phone call to be made) and subscribe
    to some
    * of its responses. We also set the 2nd argument to TRUE so that the
    response not
    * subscribed will be directly sent to the external application */
    adl_atCmdCreate(paras->StrData,
                    TRUE,
                    (adl_atRspHandler_t)ATD_Response_Handler,
                    ADL_AT_PORT_TYPE ( paras->Port, TRUE),
                    "+WIND: 2",
                    "OK",
                    NULL);
}

/*main function */
void adl_main(adl_InitType_e adlInitType)
{
    /* Subscribe to the 'atd' command.*/
    adl_atCmdSubscribe("atd",
                      (adl_atCmdHandler_t)ATD_Handler,
                      ADL_CMD_TYPE_ACT);
}

```

3.1.5.2 The `adl_atCmdSendText` function

This function is used to provide a running “Text Mode” command on a specific port (e.g. “AT+CMGW”) with the required text. This function has to be used as soon as the prompt response (“>”) comes in the response handler provided on `adl_atCmdCreate` function call.

- **Prototype**

```
s8 adl_atCmdSendText ( adl_port_e Port,  
                      ascii * Text )
```

- **Parameters**

Port:

Port on which the “Text Mode” command is currently running and 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; the text has been provided to the running “Text Mode” command: the response handler provided on `adl_atCmdCreate` call will be notified with the command responses.
- `ADL_RET_ERR_PARAM` on parameter error (NULL text)
- `ADL_RET_ERR_UNKNOWN_HDL` if the required port is not available.
- `ADL_RET_ERR_BAD_STATE` if there is no “Text Mode” command currently running on the required port.

- **Note**

It is not possible to send the text in several steps. As soon as the `adl_atCmdSendText` function is used, the text provided will immediately be sent, and the command will be executed (further calls to `adl_atCmdSendText` will return `ADL_RET_ERR_BAD_STATE`, until a new “Text Mode” command is sent on this port).

It is possible to insert new lines (\‘r’ characters) in the text body.

ADL User Guide for Open AT® OS v3.13

API

3.1.5.3 AT commands ports processing

Several AT commands ports are available on the module; an application may know each port's current state using the AT/FCM Port service.

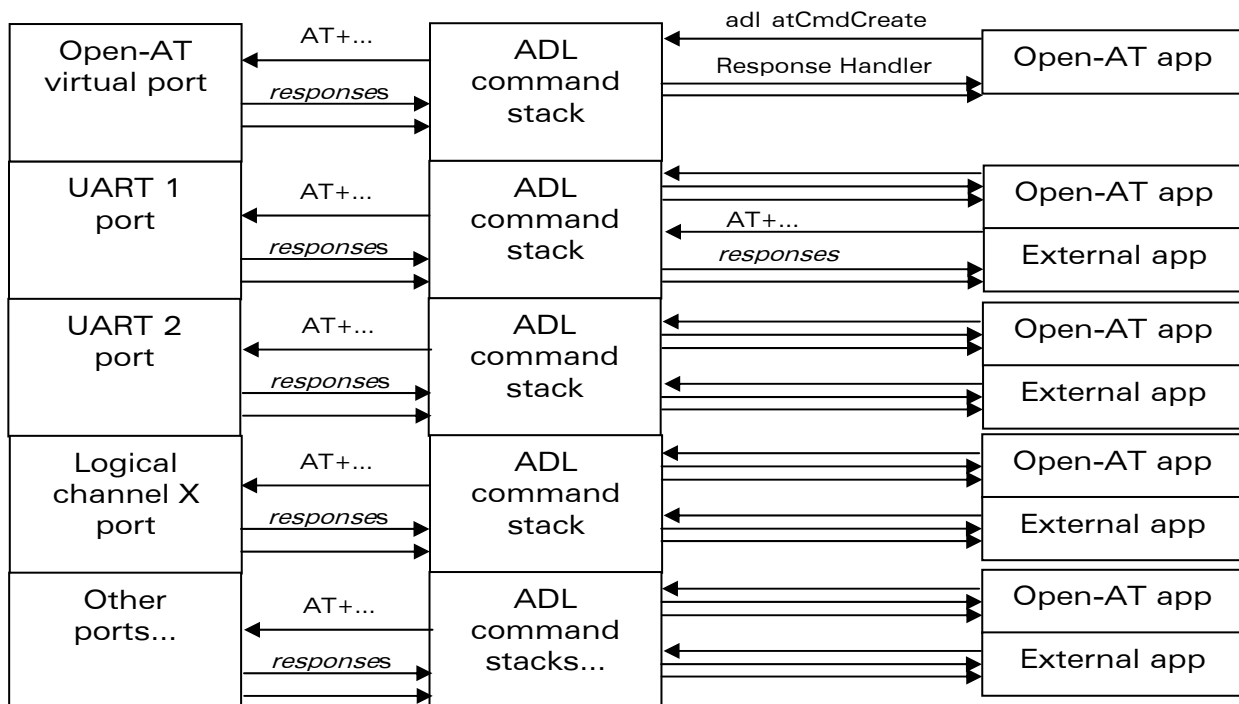
When an AT command is sent using the `adl_atCmdCreate` function, this is pushed on the required port inner command stack. ADL is processing one command stack by available port on the module.

When an AT command is sent from an external application on a specific port, this command is also pushed on the required port inner command stack.

For each command stack, while this stack is not empty, ADL sends the commands one by one (ie. ADL sends the command on the required port, waits until the terminal response is received, and then continue with the next command) until reaching the stack's end.

In addition to module physical UART ports and logical 27.010 channel ports, there is an additional Open AT® virtual port, usable to send commands only with Open AT® applications (in order not to be disturbed, or not to disturb applications running on the module physical ports).

ADL AT command stacks architecture is summarized in the following diagram:



3.2 Timers

3.2.1 Required Header Files

The header file for the functions dealing with timers is:
adl_TimerHandler.h

3.2.2 The adl_tmrSubscribe function

This function starts a timer with an associated callback function. The callback function will be executed as soon as the timer expires.

Note:

Since the WAVECOM products time granularity is 18.5 ms, the 100 ms steps are emulated, reaching a value as close as possible to the requested one modulo 18.5. For example, if a 20 * 100ms timer is required, the real time value will be 1998 ms (108 * 18.5ms).

- **Prototype**

```
adl_tmr_t *adl_tmrSubscribe( bool bCyclic,
                             u32 TimerValue,
                             u8 TimerType,
                             adl_tmrHandler_t Timerhdl )
```

- **Parameters**

bCyclic:

This boolean flag indicates whether the timer is cyclic (TRUE) or not (FALSE). The cyclic timer is automatically set up when a cycle is over.

TimerValue:

The number of periods after which the timer expires (TimerType dependant).

TimerType:

Unit of the TimerValue parameter. The allowed values are defined below:

Timer type	Timer unit
ADL_TMR_TYPE_100MS	TimerValue is in 100 ms steps
ADL_TMR_TYPE_TICK	TimerValue is in 18.5 ms tick steps

Timerhdl:

The handler of the callback function associated to the timer.

It is defined following the type below:

```
typedef void (*adl_tmrHandler_t) ( u8 )
```

The argument of the callback function will be the timer ID received by the ADL parser.

• **Returned values**

A pointer to the timer started (that will be later used, for instance for the un-subscription). There can only be 32 timers running at the same time, if you try to get more this function will return a NULL pointer.

Note: The function will return a NULL pointer if the timer value is zero. The timer will not be started.

3.2.3 The adl_tmrUnSubscribe function

This function stops the timer and unsubscribes to it and his handler. The call to this function is only meaningful to a cyclic timer or a timer that has not expired yet.

• **Prototype**

```
s32 adl_tmrUnSubscribe( adl_tmr_t *tim,
                       adl_tmrHandler_t Timerhdl,
                       u8 TimerType )
```

• **Parameters**

tim:

The timer we want to unsubscribe to.

Timerhdl:

The handler of the callback function associated to the timer.

Note: this parameter is only used to verify the coherence of **tim** parameter.

Timerhdl has to be the timer handler used in the subscription procedure.

For example

```
PhoneTaskTimerPtr = adl_tmrSubscribe (TRUE, OneSecond,
                                       ADL_TMR_TYPE_100MS, PhoneTaskTimer) ;
.....
adl_tmrUnSubscribe (PhoneTaskTimerPtr, PhoneTaskTimer,
                   ADL_TMR_TYPE_100MS) ;
```

TimerType:

Unit of the TimerValue parameter. The allowed values are defined below:

Timer type	Timer unit
ADL_TMR_TYPE_100MS	TimerValue is in 100 ms steps
ADL_TMR_TYPE_TICK	TimerValue is in 18.5 ms tick steps

• **Returned values**

- ERROR if the timer was not found or could not be stopped,
- the remaining time of the timer before it expires (unit according to the TimerValue parameter)
- ADL_RET_ERR_BAD_HDL if the handler provided is not the timer's handler
- ADL_RET_ERR_BAD_STATE if the handler has already expired.

3.2.4 Example

```
adl_tmr_t *tt;
u16 timeout_period = 5;          // in 100 ms steps;

void Timer_Handler( u8 Id )
{
    /* We don't unsubscribe to the timer because it has 'naturally' expired
    */
    adl_atSendResponse(ADL_AT_RSP, "\r\Timer timed out\r\n");}

/*main function */
void adl_main(adl_InitType_e adlInitType)
{
    /* We set up a timer */
    tt = (adl_tmr_t *)adl_tmrSubscribe, (FALSE,
        timeout_period,
        ADL_TMR_TYPE_100MS,
        (adl_tmrHandler_t)Timer_Handler);
}
```

3.3 Memory Service

3.3.1 Required Header File

The header file for the memory functions is:
adl_memory.h

3.3.2 The adl_memGetType function [DEPRECATED]

This function returns the current Wireless CPU® memory type.

Note:

This function is deprecated, and will always return the `ADL_MEM_TYPE_B` value, whatever is the Wireless CPU® memory type.

- **Prototype**

```
adl_memType_e adl_memGetType ( void )
```

- **Parameters**

None

- **Returned values**

The current Wireless CPU® memory type, defined by one of the `adl_memType_e` values below:

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

ADL_MEM_TYPE_A

A memory type Wireless CPU®. Please refer to the Open AT® Memory Resources chapter for more information about this memory type available resource.

ADL_MEM_TYPE_B

B memory type Wireless CPU®. Please refer to the Open AT® Memory Resources chapter for more information about this memory type available resource.

3.3.3 The `adl_memGetInfo` function

This function returns information about the Open AT RAM areas sizes.*

- **Prototype**

```
s32 adl_memGetInfo ( adl_memInfo_t * Info )
```

- **Parameters**

Info:

Structure updated by the function, using the following type:

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

- TotalSize

Total RAM size for the Open AT application (in bytes).

Please refer to the § 2.8 "Memory Resources" for more information.

- StackSize

Open AT application call stack area size (in bytes).

This size is defined by the Open AT application through the `wm_apmCustomStackSize` constant (Please refer to the Mandatory API chapter for more information (§ 2.2)).

- HeapSize

Open AT application total heap memory area size (in bytes).

This size is the difference between the total Open AT memory size and the Global & Stack areas sizes.

- GlobalSize

Open AT application global variables area size (in bytes).

This size is reckoned at the binary link step; it includes the ADL library, plug-in libraries (if any) and Open AT application global variables.

• **Reminder:**

The Open AT RAM is divided in three areas (Call stack, Heap memory & Global variables). This function allows to know the several area sizes.

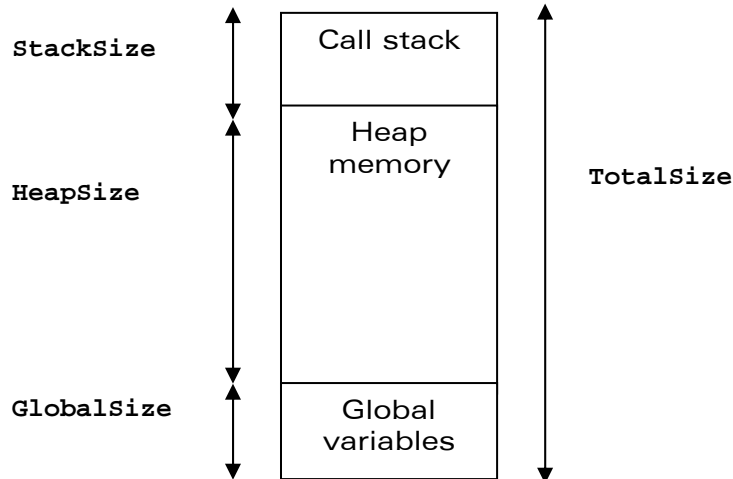


Figure 2: Open AT RAM mapping, with `adl_memInfo_t` structure fields names

• **Returned values**

- OK on success; the Info parameter is also updated with the Open AT RAM information.
- `ADL_RET_ERR_PARAM` on parameter error.

3.3.4 The `adl_memGet` function

This function allocates the memory for the requested **size** into the client application RAM memory.

• **Prototype**

`void * adl_memGet (u16 size)`

• **Parameters**

size:

The memory buffer requested size (in bytes).

• **Returned values**

A pointer to the allocated memory buffer on success.

If the memory allocation fails, this function will lead to a `ADL_ERR_MEM_GET` error, which can be handled by the Management Service. If this error is filtered and refused by the error handler, the function will return NULL. Please refer to the § 3.9 "Management service " for more information.

3.3.5 The `adl_memRelease` function

This function releases the allocated memory buffer designed by the supplied pointer.

- **Prototype**

```
bool adl_memRelease ( void * ptr )
```

- **Parameters**

ptr:

A pointer on the allocated memory buffer.

- **Returned values**

TRUE if the memory was correctly released.

In this case, the pointer provided is set to NULL.

If the memory release fails, this function will lead to a `ADL_ERR_MEM_RELEASE` error, which can be handled by the Management Service. If this error is filtered and refused by the error handler, the function will return FALSE. Please refer to the § 3.9 "Management service" for more information.

3.3.1 Example

```
// Somewhere in the application code, used as an event handler
void MyFunction ( void )
{
    // Local variables
    adl_memInfo_t MemInfo;
    u8 * MyByteBuffer

    // Gets Open AT RAM information
    adl_memGetInfo ( &MemInfo );

    // Allocates a 10 bytes memory buffer
    MyByteBuffer = ( u8 * ) adl_memGet ( 10 );

    // Releases the previously allocated memory buffer
    adl_memRelease ( MyByteBuffer );
}
```

3.4 Debug traces

This service is used to display software « trace » strings on the Target Monitoring Tool. The different ways to embed these trace strings in an Open AT® application depends on the selected configuration in the used IDE (or with the `wmmake` command). For more information about the Target Monitoring Tool, the configurations and the Integrated Development Environments, please refer to the Tools Manual.

3.4.1 Required Header File

The header file for the flash functions is:
`adl_traces.h`

3.4.2 Debug configuration

When the Debug configuration is selected in the used IDE (or with the `wmmake` command), the `__DEBUG_APP__` compilation flag is defined, and also the following macros.

- `TRACE ((u8 TL, ascii * T, ...))`
Prints a "trace" in the Target Monitoring Tool.

`TL` defines the trace level (traces will be displayed on the CUS4 element of the Target Monitoring Tool).
Trace levels range is from 1 to 32.
`T` is the trace string, which may use the standard C "sprintf" syntax.
Please note that maximum string length displayed is 256 bytes. If the string is longer, it will be truncated on display.

Example:

```
u8 I = 123;  
TRACE (( 1, "Value if I : %d", I ));
```

At runtime, this will display the following string on the CUS4 level 1 on the Target Monitoring Tool:
Value of I: 123

- `DUMP (u8 TL, u8 * P, u16 L)`
Displays the content (each byte in hexadecimal format) of the buffer provided in the Target Monitoring Tool.

`TL` defines the trace level (traces will be displayed on the CUS4 element of the Target Monitoring Tool).
Trace level range is from 1 to 32.
`P` is the buffer's address to dump.
`L` is the length (in bytes) of the required dump.

ADL User Guide for Open AT® OS v3.13

API

Since the maximum length of a display line is 255 bytes, if the display length is greater than 80 (each byte is displayed on 3 ascii characters), the dump will be segmented on several lines. Each 80 bytes truncated line will end with the "... " character sequence.

Example 1:

```
u8 * Buffer = "\x0\x1\x2\x3\x4\x5\x6\x7\x8\x9";
DUMP ( 1, Buffer, 10 );
```

At runtime, this will display the following string on the CUS4 level 1 on the Target Monitoring Tool:

```
00 01 02 03 04 05 06 07 08 09
```

Example 2:

```
u8 Buffer [ 200 ], i;
for ( i = 0 ; i < 200 ; i++ ) Buffer [ i ] = i;
DUMP ( 1, Buffer, 200 );
```

At runtime, this will display the following three lines on the CUS4 level 1 on the Target Monitoring Tool:

```
00 01 02 03 04 05 06 07 08 09 0A [bytes from 0B to 4D] 4E 4F...
50 51 52 53 54 55 56 57 58 59 5A [bytes from 5B to 9D] 9E 9F...
A0 A1 A2 A3 A4 A5 A6 A7 [bytes from A8 to C4] C5 C6 C7
```

In this Debug configuration, the FULL_TRACE and FULL_DUMP macros are ignored (even if these are used in the application source code, they will neither be compiled, nor displayed on the Target Monitoring Tool at runtime).

3.4.3 Full Debug configuration

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

- `TRACE ((u8 TL, ascii * T, ...))`
Cf. the Debug configuration
- `DUMP (u8 TL, u8 * P, u16 L)`
Cf. the Debug configuration
- `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.

3.4.4 Release configuration

When the Release configuration is selected in the used IDE (or with the `wmmake` command), neither the `__DEBUG_APP__` nor `__DEBUG_FULL__` compilation flags are defined.

In this configuration, the `TRACE`, `DUMP`, `FULL_TRACE` and `FULL_DUMP` macros are ignored (even if these are used in the application source code, they will neither be compiled, nor displayed on the Target Monitoring Tool at runtime).

3.5 Flash

3.5.1 Required Header File

The header file for the flash functions is:
adl_flash.h

3.5.2 Flash Objects Management

An ADL application may subscribe to a set of objects identified by a handle, used by all ADL flash functions.

This handle is chosen and given by the application at subscription time.

To access to a particular object, the application gives the handle and the ID of the object to access.

At first subscription, the Handle and the associated set of IDs are saved in flash. The number of flash object IDs associated to a given handle may be only changed after have erased the flash objects (with the AT+WOPEN=3 command).

For a particular handle, the flash objects ID take any value, from 0 to the ID range upper limit provided on subscription.

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

3.5.2.1 Flash objects write/erase inner process overview

Written flash objects are queued in the flash object storage place. Each time the adl_flhWrite function is called, the process below is performed:

- If the object already exists, it is now considered as "erased" (ie. "adl_flhWrite(X);" <=> "adl_flhDelete(X); adl_flhWrite(X);")
- The flash object driver checks if there is enough place to store the new object. If not, a Garbage Collector process is performed (see below).
- The new object is created.

About the erase process, each time the adl_flhDelete (or adl_flhWrite) function is called on an ID, this object is from this time "considered as erased", even if it is not physically erased (an inner "erase flag" is set on this object).

Objects are physically erased only when the Garbage Collector process is performed, when an adl_flhWrite function call needs a size bigger than the available place in the flash objects storage place. The Garbage Collector process erases the flash objects storage place and re-write only the objects which do not have their "erase flag" set. Please note that the flash memory physical limitation is the erasure cycle number, which is granted to be at least 100.000 times.

3.5.2.2 Flash objects in Remote Task Environment

When an application is running in Remote Task Environment, the flash object storage place is emulated on the PC side: objects are read/written from/to files on the PC hard disk, and not from/to the module's flash memory. The two storage places (module and PC) may be synchronized using the RTE Monitor interface (cf. the Tools Manual for more information).

3.5.3 The `adl_flhSubscribe` function

This function subscribes to a set of objects identified by the given Handle.

- **Prototype**

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

- **Parameters**

Handle:

The Handle of the set of objects to subscribe to.

NbObjectRes:

The number of objects related to the given handle. It means that the IDs available for this handle are in the range [0 , (NbObjectRes - 1)].

- **Returned values**

- OK on success (first allocation for this handle)
- ADL_RET_ERR_PARAM on parameter error,
- ADL_RET_ERR_ALREADY_SUBSCRIBED if space is already created for this handle,
- ADL_FLH_RET_ERR_NO_ENOUGH_IDS if there are no longer enough object IDs to allocate the handle.

Notes:

- Only one subscription is necessary. It is not necessary to subscribe to the same handle at each application start.
- It is not possible to unsubscribe from a handle. To release the handle and the associated objects, the user must do an AT+WOPEN=3 to erase the flash objects of the Open AT® Embedded Application.

3.5.4 The `adl_flhExist` function

This function checks if a flash object exists from the given Handle at the given ID in the flash memory allocated to the ADL developer.

- **Prototype**

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

- **Parameters**

Handle:

The Handle of the subscribed set of objects.

ID:

The ID of the flash object to investigate (in the range allocated to the Handle provided).

- **Returned values**

- the requested Flash object length on success
- 0 if the object does not exist.
- `ADL_RET_ERR_UNKNOWN_HDL` if handle is not subscribed
- `ADL_FLH_RET_ERR_ID_OUT_OF_RANGE` if ID is out of handle range

3.5.5 The `adl_flhErase` function

This function erases the flash object from the given Handle at the given ID.

- **Prototype**

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

- **Parameters**

Handle:

The Handle of the subscribed set of objects.

ID:

The ID of the flash object to be erased.

Important note: If ID is set to `ADL_FLH_ALL_IDS`, all flash objects related to the handle provided will be erased.

- **Returned values**

- OK on success
- `ADL_RET_ERR_UNKNOWN_HDL` if handle is not subscribed
- `ADL_FLH_RET_ERR_ID_OUT_OF_RANGE` if ID is out of handle range
- `ADL_FLH_RET_ERR_OBJ_NOT_EXIST` if the object does not exist
- `ADL_RET_ERR_FATAL` if a fatal error occurred (`ADL_ERR_FLH_DELETE` error event will then be generated)

3.5.6 The `adl_flhWrite` function

This function writes the flash object from the given Handle at the given ID, for the length provided with the string provided. A single flash object can use up to 30 Kbytes of memory.

- **Prototype**

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

- **Parameters**

Handle:

The Handle of the subscribed set of objects.

ID:

The ID of the flash object to write.

Len:

The length of the flash object to write.

WriteData:

The string provided to write in the flash object.

- **Returned values**

- OK on success
- ADL_RET_ERR_PARAM if one at least of the parameters has a bad value.
- ADL_RET_ERR_UNKNOWN_HDL if handle is not subscribed
- ADL_FLH_RET_ERR_ID_OUT_OF_RANGE if ID is out of handle range
- ADL_RET_ERR_FATAL if a fatal error occurred (ADL_ERR_FLH_WRITE error event will then occur).
- ADL_FLH_RET_ERR_MEM_FULL if flash memory is full.
- ADL_FLH_RET_ERR_NO_ENOUGH_IDS if the object cannot be created due to the global ID number limitation.

3.5.7 The `adl_flhRead` function

This function reads the flash object from the given Handle at the given ID, for the length provided and stores it in a string.

- **Prototype**

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

- **Parameters**

Handle:

The Handle of the subscribed set of objects

ID:

The ID of the flash object to read.

Len:

The length of the flash object to read.

ReadData:

The string allocated to store the read flash object.

• **Returned values**

- OK on success
- ADL_RET_ERR_PARAM if one at least of the parameters has a bad value.
- ADL_RET_ERR_UNKNOWN_HDL if handle is not subscribed
- ADL_FLH_RET_ERR_ID_OUT_OF_RANGE if ID is out of handle range
- ADL_FLH_RET_ERR_OBJ_NOT_EXIST if the object does not exist.
- ADL_RET_ERR_FATAL if a fatal error occurred (ADL_ERR_FLH_READ error event will then occur).

3.5.8 The `adl_flhGetFreeMem` function

This function gets the current remaining flash memory size.

• **Prototype**

```
u32 adl_flhGetFreeMem ( void )
```

• **Returned values**

Current free flash memory size in bytes.

3.5.9 The `adl_flhGetIDCount` function

This function returns the ID count for the handle provided, or the total remaining ID count.

• **Prototype**

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

• **Parameters**

Handle:

The Handle of the subscribed set of objects. If set to NULL, the total remaining ID count will be returned.

• **Returned values**

- ID count on success: allocated on the handle provided, if any, or the total remaining ID count if the handle is set to NULL.
- ADL_RET_ERR_UNKNOWN_HDL if handle is not subscribed

3.5.10 The `adl_flhGetUsedSize` function

This function returns the used size by the provided ID range from the handle provided. The handle should also be set to NULL to get the whole used size.

- **Prototype**

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

- **Parameters**

Handle:

The Handle of the subscribed set of objects. If set to NULL, the whole flash memory used size will be returned.

StartID:

First ID of the range from which to get the used size; has to be lower than EndID.

EndID:

Last ID of the range from which to get the used size; has to be greater than StartID. To get the used size by all a handle's IDs, the [0 , ADL_FLH_ALL_IDS] range may be used

- **Returned values**

- Used size on success: from the Handle provided, if any, otherwise the whole flash memory used size
- ADL_RET_ERR_PARAM on parameter error
- ADL_RET_ERR_UNKNOWN_HDL if handle is not subscribed
- ADL_FLH_RET_ERR_ID_OUT_OF_RANGE if ID is out of handle range

3.6 FCM Service

ADL provides a FCM service to handle all FCM events, and to access to the data ports provided on the product.

An ADL application may subscribe to a specific flow (UART 1, UART 2 or USB physical/virtual ports, GSM CSD call data port, GPRS session data port or Bluetooth virtual data ports) to exchange data on it.

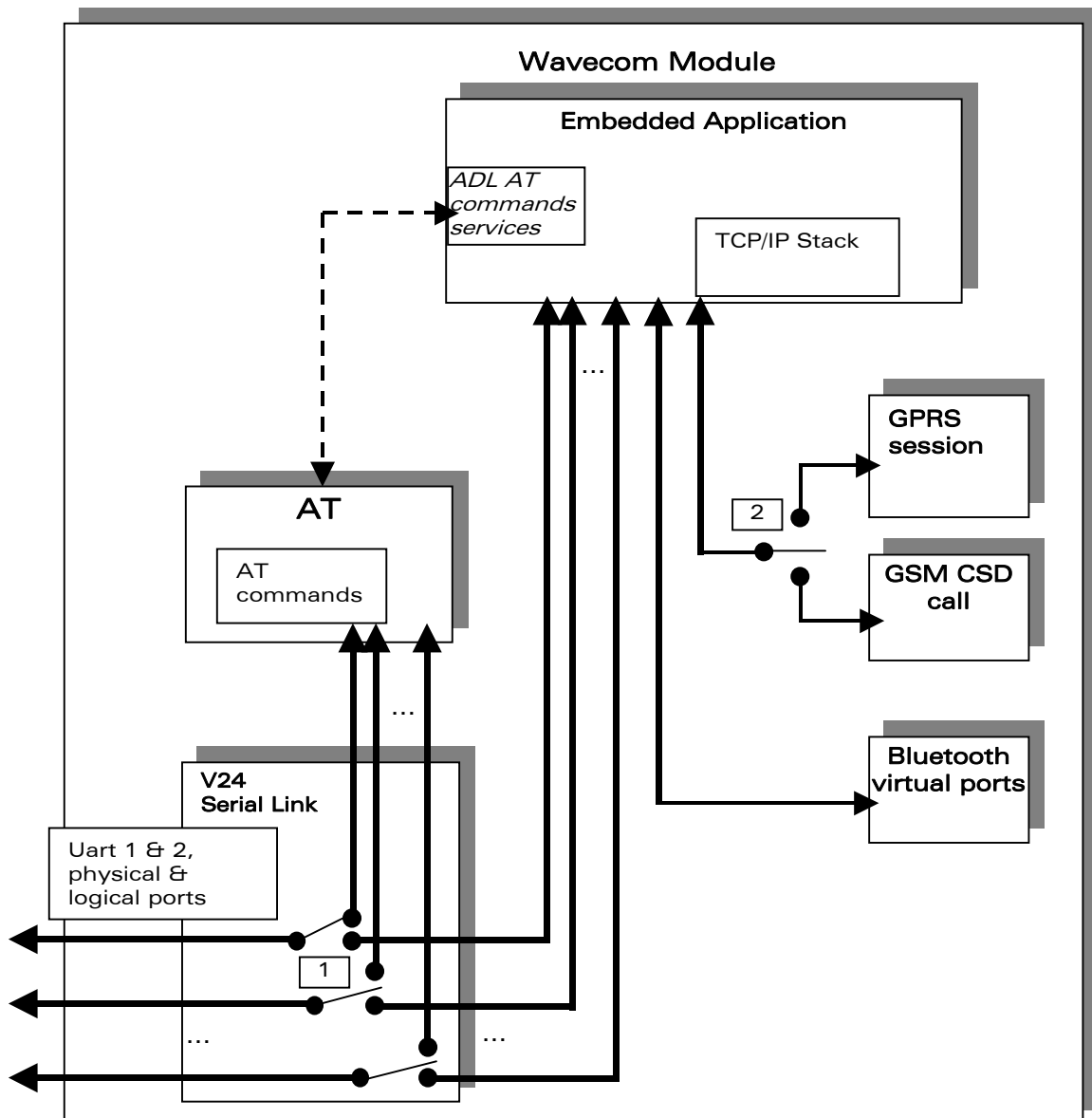


Figure 3: Flow Control Manager representation

ADL User Guide for Open AT® OS v3.13

API

By default (ie. without any Open AT® application, or if the application does not use the FCM service), all the module's ports are processed by the Wavecom Core Software. The default behaviors are:

- When a GSM CSD call is set up, the GSM CSD data port is directly connected to the UART port where the ATD command was sent;
- When a GPRS session is set up, the GPRS data port is directly connected to the UART port where the ATD or AT+CGDATA command was sent;
- When a Bluetooth peripheral is detected & connected through an SPP based profile, a local data bridge may be set up between a Bluetooth virtual data port and the required UART port, using the AT+WLDB command.

Once subscribed by an Open AT® application with the FCM service, a port is no longer available to be used with the AT commands by an external application. The available ports are the ones listed in the ADL AT/FCM Ports service:

- ADL_PORT_UART_X / ADL_PORT_UART_X_VIRTUAL_BASE identifiers may be used to access the module's physical UARTS, or logical 27.010 protocol ports;
- ADL_PORT_GSM_BASE identifier may be used to access a remote modem (connected through a GSM CSD call) data flow;
- ADL_PORT_GPRS_BASE identifier may be used to exchange IP packets with the operator network and the Internet;
- ADL_PORT_BLUETOOTH_VIRTUAL_BASE may be used to access a connected Bluetooth device data stream with the Serial Port Profile (SPP).

The "1" switches on the figure above means that UART based ports may be used with AT commands or FCM services as well. These switches are processed by the `adl_fcmSwitchV24State` function.

The "2" switch on the figure above means that either the GSM CSD port or the GPRS port may be subscribed at one time, but not both together.

Important note

GPRS provides only **packet** mode transmission. This means that the embedded application can only send/receive **IP packets** to/from the GPRS flow.

3.6.1 Required Header File

The header file for the FCM functions is:

```
adl_fcm.h
```

3.6.2 The `adl_fcmIsAvailable` function

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

- **Prototype**

```
bool adl_fcmIsAvailable ( adl_fcmFlow_e Flow );
```

- **Parameters**

Flow:

Port from which to require the state.

- **Returned values**

- TRUE if the port is ready to handle the FCM service
- FALSE if it is not ready

- **Notes**

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

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

3.6.3 The `adl_fcmSubscribe` function

This function subscribes to the FCM service, opening the requested port and setting the control and data handlers. The subscription will be effective only when the control event handler has received the `ADL_FCM_EVENT_FLOW_OPENED` event. Each port may be subscribed only one time.

Additional subscriptions may be performed, using the `ADL_FCM_FLOW_SLAVE` flag (see below). Slave subscribed handles will be able to send & receive data on/from the flow, but will know some limitations:

- For serial-line flows (UART physical & logical based ports), only the main handle will be able to switch the Serial Link state between AT & Data mode;
- If the main handle unsubscribes from the flow, all slave handles will also be unsubscribed.

Important note:

For serial-link related flows (UART physical & logical based ports), the corresponding port has to be opened first with the `AT+WMFM` command (for physical ports), or with the 27.010 protocol driver on the external application side (for logical ports), otherwise the subscription will fail. See the AT Commands Interface guide for more information.

By default, only the UART1 physical port is opened.

ADL User Guide for Open AT® OS v3.13

API

A specific port state may be known using the ADL AT/FCM port service.

- **Prototype**

```
s8    adl_fcmSubscribe    (    adl_fcmFlow_e        Flow,
                             adl_fcmCtrlHdlr_f    CtrlHandler,
                             adl_fcmDataHdlr_f    DataHandler );
```

- **Parameters**

Flow:

The allowed values are the available ports of the `adl_port_e` type. Only ports with the FCM capability may be used with this service (i.e. all ports except the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` and not SPP `ADL_PORT_BLUETOOTH_VIRTUAL_BASE` based ones).

Please note that `adl_fcmFlow_e` type is the same as the `adl_port_e`, except for the fact that it may handle some additional FCM specific flags (see below). Previous version FCM flows identifiers have been kept for upward compatibility. However, these constants should be considered as deprecated, and the `adl_port_e` type members should now be used instead.

```
#define ADL_FCM_FLOW_V24_UART1    ADL_PORT_UART1
#define ADL_FCM_FLOW_V24_UART2    ADL_PORT_UART2
#define ADL_FCM_FLOW_V24_USB      ADL_PORT_USB
#define ADL_FCM_FLOW_GSM_DATA     ADL_PORT_GSM_BASE
#define ADL_FCM_FLOW_GPRS         ADL_PORT_GPRS_BASE
```

To perform a slave subscription (see above), a bit-wise or has to be done with the flow ID and the `ADL_FCM_FLOW_SLAVE` flag; for example:

```
adl_fcmSubscribe (    ADL_PORT_UART1 | ADL_FCM_FLOW_SLAVE,
                     MyCtrlHandler, MyDataHandler );
```

CtrlHandler:

FCM control events handler, using the following type:

```
typedef bool ( * adl_fcmCtrlHdlr_f ) (adl_fcmEvent_e event );
```

The FCM control events are defined below (All handlers related to the concerned flow (master and slaves) will be notified together with these events):

- `ADL_FCM_EVENT_FLOW_OPENED` (related to `adl_fcmSubscribe`),
- `ADL_FCM_EVENT_FLOW_CLOSED` (related to `adl_fcmUnsubscribe`),
- `ADL_FCM_EVENT_V24_DATA_MODE` (related to `adl_fcmSwitchV24State`),
- `ADL_FCM_EVENT_V24_DATA_MODE_EXT` (see note below),
- `ADL_FCM_EVENT_V24_AT_MODE` (related to `adl_fcmSwitchV24State`),
- `ADL_FCM_EVENT_V24_AT_MODE_EXT` (see note below),
- `ADL_FCM_EVENT_RESUME` (related to `adl_fcmSendData` and `adl_fcmSendDataExt`),
- `ADL_FCM_EVENT_MEM_RELEASE` (related to `adl_fcmSendData` and `adl_fcmSendDataExt`),

ADL User Guide for Open AT® OS v3.13

API

This handler return value is not relevant, except for ADL_FCM_EVENT_V24_AT_MODE_EXT.

DataHandler:

FCM data events handler, using the following type:

```
typedef bool ( * adl_fcmDataHdlr_f ) ( u16 DataLen, u8 * Data );
```

This handler receives data blocks from the associated flow.

Once the data block is processed, the handler must return TRUE to release the credit, or FALSE if the credit must not be released. In this case, all credits will be released next time the handler returns TRUE.

On all flows, all data handlers (master and slaves) subscribed are notified with a data event, and the credit will be released only if all handlers return TRUE: each handler should return TRUE as default value.

If a credit is not released on the data block reception, it will be released the next time the data handler returns TRUE. The `adl_fcmReleaseCredits()` should also be used to release credits outside of the data handler.

Maximum size of each data packet to be received by the data handlers depends on the flow type:

- On serial link flows (UART physical & logical based ports): 120 bytes;
- On GSM CSD data port: 270 bytes ;
- On GPRS port: 1500 bytes ;
- On Bluetooth virtual ports: 120 bytes.

If data size to be received by the Open AT® application exceeds this maximum packet size, data will be segmented by the Flow Control Manager, which will call the Data Handlers several times with the segmented packets.

Please note that on GPRS flow, whole IP packets will always be received by the Open AT® application.

• Returned values

- A positive or null handle on success (which will have to be used in all further FCM operations). The Control handler will also receive a ADL_FCM_EVENT_FLOW_OPENED event when flow is ready to process,
- ADL_RET_ERR_PARAM if one parameter has an incorrect value,
- ADL_RET_ERR_ALREADY_SUBSCRIBED if the flow is already subscribed in master mode,
- ADL_RET_ERR_NOT_SUBSCRIBED if a slave subscription is made when master flow is not subscribed,
- ADL_FCM_RET_ERROR_GSM_GPRS_ALREADY_OPENED if a GSM or GPRS subscription is made when the other one is already subscribed.
- ADL_RET_ERR_BAD_STATE if the required port is not available.

- **Notes**

- When "7 bits" mode is enabled on a v24 serial link, in data mode, payload data is located on the 7 least significant bits (LSB) of every byte.
- When a serial link is in data mode, if the external application sends the sequence "1s delay ; +++ ; 1s delay", this serial link is switched to AT mode, and corresponding handler is notified by the ADL_FCM_EVENT_V24_AT_MODE_EXT event. Then the behavior depends on the returned value.
If it is TRUE, all this flow remaining handlers are also notified with this event. The main handle cannot be un-subscribed in this state.
If it is FALSE, this flow remaining handlers are not notified with this event, and this serial link is immediately switched back to data mode.
In the first case, after the ADL_FCM_EVENT_V24_AT_MODE_EXT event, the main handle subscriber should switch the serial link to data mode with the adl_fcmSwitchV24State API, or wait for the ADL_FCM_EVENT_V24_DATA_MODE_EXT event. This event will come when the external application sends the "ATO" command: the serial link is switched to data mode, and then all V24 clients are notified.
- When a GSM data call is released from the remote part, the GSM flow will automatically be unsubscribed (the ADL_FCM_EVENT_FLOW_CLOSED event will be received by all the flow subscribers).
- When a GPRS session is released, or when a GSM data call is released from the module side (with the adl_callHangUp function), the corresponding GSM or GPRS flow have to be unsubscribed. These flows will have to be subscribed again before starting up a new GSM data call, or a new GPRS session.
- For serial link flows, the serial line parameters (speed, character framing, etc...) must not be modified while the flow is in data state. In order to change these parameters' value, the flow concerned has firstly to be switched back in AT mode with the adl_fcmSwitchV24State API. Once the parameters have changed, the flow may be switched again to data mode, using the same API.

3.6.4 The `adl_fcmUnsubscribe` function

This function unsubscribes from a previously subscribed FCM service, closing the previously opened flows. The unsubscription will be effective only when the control event handler has received the `ADL_FCM_EVENT_FLOW_CLOSED` event.

If slave handles were subscribed, as soon as the master unsubscribes from the flow, all the slaves will also be unsubscribed.

- **Prototype**

```
s8 adl_fcmUnsubscribe ( u8 Handle );
```

- **Parameters**

Handle:

Handle returned by the `adl_fcmSubscribe` function.

- **Returned values**

- OK on success. The Control handler will also receive a `ADL_FCM_EVENT_FLOW_CLOSED` event when flow is ready to process
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle is incorrect,
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the flow is already unsubscribed,
- `ADL_RET_ERR_BAD_STATE` if the serial link is not in AT mode.

3.6.5 The `adl_fcmReleaseCredits` function

This function releases some credits for requested flow handle. The slave subscribers should not use this API.

- **Prototype**

```
s8 adl_fcmReleaseCredits ( u8 Handle,  
                          u8 NbCredits );
```

- **Parameters**

Handle:

Handle returned by the `adl_fcmSubscribe` function.

NbCredits:

Number of credits to release for this flow. If this number is greater than the number of previously received data blocks, all credits are released. If an application wants to release all received credits at any time, it should call the `adl_fcmReleaseCredits` API with `NbCredits` parameter set to `0xFF`.

- **Returned values**

- OK on success.
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided handle is unknown,
- `ADL_RET_ERR_BAD_HDL` if the handle is a slave one.

3.6.6 The `adl_fcmSwitchV24State` function

This function switches a serial link state to AT mode or to Data mode. The operation will be effective only when the control event handler has received an `ADL_FCM_EVENT_V24_XXX_MODE` event. Only the main handle subscriber can use this API.

- **Prototype**

```
s8 adl_fcmSwitchV24State ( u8 Handle,
                          u8 V24State );
```

- **Parameters**

Handle:

Handle returned by the `adl_fcmSubscribe` function.

V24State:

Serial link state to switch to. Allowed values are defined below:

`ADL_FCM_V24_STATE_AT`,
`ADL_FCM_V24_STATE_DATA`

- **Returned values**

- OK on success. The Control handler will also receive a `ADL_FCM_EVENT_V24_XXX_MODE` event when the serial link state has changed
- `ADL_RET_ERR_PARAM` if one parameter has an incorrect value
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided handle is unknown
- `ADL_RET_ERR_BAD_HDL` if the handle is not the main flow one

3.6.7 The `adl_fcmSendData` function

This function sends a data block on the requested flow.

- **Prototype**

```
s8 adl_fcmSendData ( u8 Handle,
                    u8 * Data,
                    u16 DataLen );
```

- **Parameters**

Handle:

Handle returned by the `adl_fcmSubscribe` function.

Data:

Data block buffer to write.

DataLen:

Data block buffer size.

Maximum data packet size depends on the subscribed flow:

- On serial link based flows : 2000 bytes;
- On GSM data flow : no limitation (memory allocation size);
- On GPRS flow : 1500 bytes;
- On Bluetooth virtual ports: 2000 bytes.

- **Returned values**

- OK on success. The Control handler will also receive a ADL_FCM_EVENT_MEM_RELEASE event when the data block memory buffer is released;
- ADL_FCM_RET_OK_WAIT_RESUME on success, but the last credit was used. The Control handler will also receive a ADL_FCM_EVENT_MEM_RELEASE event when the data block memory buffer will be released;
- ADL_RET_ERR_PARAM is a parameter has an incorrect value;
- ADL_RET_ERR_UNKNOWN_HDL if the provided handle is unknown;
- ADL_RET_ERR_BAD_STATE if the flow is not ready to send data;
- ADL_FCM_RET_ERR_WAIT_RESUME if the flow has no more credit to use.

On ADL_FCM_RET_XXX_WAIT_RESUME returned value, the subscriber has to wait for a ADL_FCM_EVENT_RESUME event on Control Handler to continue sending data.

3.6.8 The `adl_fcmSendDataExt` function

This function sends a data block on the requested flow. This API do not perform any processing on the data block provided, which is sent directly on the flow.

- **Prototype**

```
s8 adl_fcmSendDataExt ( u8 Handle,
                       adl_fcmDataBlock_t * DataBlock);
```

- **Parameters**

Handle:

Handle returned by the `adl_fcmSubscribe` function.

DataBlock:

Data block buffer to write, using the following type:

```
typedef struct
{
    u16 Reserved1[4];
    u32 Reserved3;
    u16 DataLength; /* Data length */
    u16 Reserved2[5];
    u8 Data[1]; /* Data to send */
} adl_fcmDataBlock_t;
```

The block must be dynamically allocated and filled by the application, before sending it to the function. The allocation size has to be `sizeof (adl_fcmDataBlock_t) + DataLength`, where `DataLength` is the value to be set in the `DataLength` field of the structure.

Maximum data packet size depends on the subscribed flow:

- On serial link based flows: 2000 bytes,
- On GSM data flow: no limitation (memory allocation size),
- On GPRS flow: 1500 bytes,
- On Bluetooth virtual ports: 2000 bytes.

- **Returned values**

- OK on success. The Control handler will also receive a `ADL_FCM_EVENT_MEM_RELEASE` event when the data block memory buffer is released,
- `ADL_FCM_RET_OK_WAIT_RESUME` on success, but the last credit was used. The Control handler will also receive a `ADL_FCM_EVENT_MEM_RELEASE` event when the data block memory buffer is released,
- `ADL_RET_ERR_PARAM` is a parameter has an incorrect value,
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided handle is unknown,
- `ADL_RET_ERR_BAD_STATE` if the flow is not ready to send data,
- `ADL_FCM_RET_ERR_WAIT_RESUME` if the flow has no more credit to use.

On ADL_FCM_RET_XXX_WAIT_RESUME returned value, the subscriber has to wait for an ADL_FCM_EVENT_RESUME event on Control Handler to continue sending data.

Important Remark:

The Data block will be released by the adl_fcmSendDataExt() API on OK and ADL_FCM_RET_OK_WAIT_RESUME return values (the memory buffer will be effectively released once the ADL_FCM_EVENT_MEM_RELEASE event is received in the Control Handler). The application has to use only dynamic allocated buffers (with adl_memGet function).

3.6.9 The adl_fcmGetStatus function

This function gets the buffer status for requested flow handle, in the requested way.

- **Prototype**

```
s8 adl_fcmGetStatus ( u8          Handle,  
                    adl_fcmWay_e Way );
```

- **Parameters**

Handle:

Handle returned by the adl_fcmSubscribe function.

Way:

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

```
typedef enum {  
    ADL_FCM_WAY_FROM_EMBEDDED,  
    ADL_FCM_WAY_TO_EMBEDDED  
} adl_fcmWay_e;
```

- **Returned values**

- ADL_FCM_RET_BUFFER_EMPTY if the requested flow and way buffer is empty,
- ADL_FCM_RET_BUFFER_NOT_EMPTY if the requested flow and way buffer is not empty; the Flow Control Manager is still processing data on this flow,
- ADL_RET_ERR_UNKNOWN_HDL if the provided handle is unknown,
- ADL_RET_ERR_PARAM if the way parameter value is out of range.

3.7 GPIO Service

ADL provides a GPIO service to handle GPIO operations.

3.7.1 Required Header File

The header file for the GPIO functions is:

adl_gpio.h

3.7.2 The adl_ioSubscribe function

This function subscribes to some GPIOs and sets up a polling system if required.

Note: using the product's second UART locks some GPIOs, which will not be available for allocation by the application; please refer to the corresponding section for more information.

- **Prototype**

```
s8    adl_ioSubscribe (  u32    GpioMask,
                        u32    GpioDir,
                        u32    GpioDefValues,
                        u32    PollingTime,
                        adl_ioHdlr_f  GpioHandler );
```

- **Parameters**

GpioMask:

Mask of GPIOs to subscribe, using the following defined values. One or several GPIOs may be subscribed, by performing a logical OR between the requested identifiers.

For Wismo Pac P31X3 and P32X3 products:

```
ADL_IO_P32X3_GPI,
ADL_IO_P32X3_GPIO_0,
ADL_IO_P32X3_GPIO_2,
ADL_IO_P32X3_GPIO_3,
ADL_IO_P32X3_GPIO_4,
ADL_IO_P32X3_GPIO_5
```

For Wismo Pac P32X6 product:

```
ADL_IO_P32X6_GPI,
ADL_IO_P32X6_GPO_0,
ADL_IO_P32X6_GPIO_0,
ADL_IO_P32X6_GPIO_2,
ADL_IO_P32X6_GPIO_3,
ADL_IO_P32X6_GPIO_4,
ADL_IO_P32X6_GPIO_5,
ADL_IO_P32X6_GPIO_8
```


For Wismo Quik Q23X3 and Q24X3 products:

ADL_IO_Q24X3_GPI,
ADL_IO_Q24X3_GPO_1,
ADL_IO_Q24X3_GPO_2,
ADL_IO_Q24X3_GPIO_0,
ADL_IO_Q24X3_GPIO_4,
ADL_IO_Q24X3_GPIO_5

For Wismo Quik Q24X6 products:

ADL_IO_Q24X6_GPI,
ADL_IO_Q24X6_GPO_0,
ADL_IO_Q24X6_GPO_1,
ADL_IO_Q24X6_GPO_2,
ADL_IO_Q24X6_GPO_3,
ADL_IO_Q24X6_GPIO_0,
ADL_IO_Q24X6_GPIO_4,
ADL_IO_Q24X6_GPIO_5

For Wismo Quik Q2400 products:

ADL_IO_Q24X0_GPI,
ADL_IO_Q24X0_GPO_0,
ADL_IO_Q24X0_GPO_1,
ADL_IO_Q24X0_GPO_2,
ADL_IO_Q24X0_GPO_3,
ADL_IO_Q24X0_GPIO_0,
ADL_IO_Q24X0_GPIO_4,
ADL_IO_Q24X0_GPIO_5

For Wismo Quik Q31X6 product:

ADL_IO_Q31X6_GPI,
ADL_IO_Q31X6_GPO_1,
ADL_IO_Q31X6_GPO_2,
ADL_IO_Q31X6_GPIO_3,
ADL_IO_Q31X6_GPIO_4,
ADL_IO_Q31X6_GPIO_5,
ADL_IO_Q31X6_GPIO_6,
ADL_IO_Q31X6_GPIO_7

For Wismo Pac P5186 product:

ADL_IO_P51X6_GPO_0
ADL_IO_P51X6_GPO_1,
ADL_IO_P51X6_GPIO_0,
ADL_IO_P51X6_GPIO_4,
ADL_IO_P51X6_GPIO_5,
ADL_IO_P51X6_GPIO_8,
ADL_IO_P51X6_GPIO_9,
ADL_IO_P51X6_GPIO_10,
ADL_IO_P51X6_GPIO_11,
ADL_IO_P51X6_GPIO_12

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

For Wismo Quik Q24 CLASSIC products:

ADL_IO_Q24CLASSIC_GPI,
ADL_IO_Q24CLASSIC_GPO_0,
ADL_IO_Q24CLASSIC_GPO_1,
ADL_IO_Q24CLASSIC_GPO_2,
ADL_IO_Q24CLASSIC_GPO_3,
ADL_IO_Q24CLASSIC_GPIO_0,
ADL_IO_Q24CLASSIC_GPIO_4,
ADL_IO_Q24CLASSIC_GPIO_5

For Wismo Quik Q24 PLUS products:

ADL_IO_Q24PLUS_GPI,
ADL_IO_Q24PLUS_GPO_0,
ADL_IO_Q24PLUS_GPO_1,
ADL_IO_Q24PLUS_GPO_2,
ADL_IO_Q24PLUS_GPO_3,
ADL_IO_Q24PLUS_GPIO_0,
ADL_IO_Q24PLUS_GPIO_4,
ADL_IO_Q24PLUS_GPIO_5

For Wismo Quik Q24 AUTO products:

ADL_IO_Q24AUTO_GPI,
ADL_IO_Q24AUTO_GPO_0,
ADL_IO_Q24AUTO_GPO_1,
ADL_IO_Q24AUTO_GPO_2,
ADL_IO_Q24AUTO_GPO_3,
ADL_IO_Q24AUTO_GPIO_0,
ADL_IO_Q24AUTO_GPIO_4,
ADL_IO_Q24AUTO_GPIO_5

For Wismo Quik Q24 EXTENDED products:

ADL_IO_Q24EXTENDED_GPI,
ADL_IO_Q24EXTENDED_GPO_0,
ADL_IO_Q24EXTENDED_GPO_1,
ADL_IO_Q24EXTENDED_GPO_2,
ADL_IO_Q24EXTENDED_GPO_3,
ADL_IO_Q24EXTENDED_GPIO_0,
ADL_IO_Q24EXTENDED_GPIO_4,
ADL_IO_Q24EXTENDED_GPIO_5

GpioDir:

Mask of GPIO directions to subscribe. For each allocated GPIO, the corresponding bit in the mask should be set to one of the following values:

- o 1: input
- o 0: output.

The "GpioMask" constants should be used also for this parameter. If this parameter is set to 0, all subscribed GPIOs are allocated as outputs. If it is set to 0xFFFFFFFF, all subscribed GPIOs are allocated as inputs.

Note: this parameter is only relevant for GPIOs; GPs are always subscribed as inputs, and GPOs are always subscribed as outputs, whatever the **GpioDir** corresponding bit value.

GpioDefValues:

Mask of GPIO default values when set as an output. For each subscribed output GPIO, the corresponding bit in the mask is the default value after allocation (0 or 1). The "GpioMask" constants should also be used for this parameter. If this parameter is set to 0, all subscribed output GPIOs are set to 0. If it is set to 0xFFFFFFFF, all subscribed output GPIOs are set to 1.

PollingTime:

If some IO is allocated as input, this parameter represents the time interval between two GPIO polling operations (unit is 100ms);
If no polling is requested, this parameter must be 0.

GpioHandler:

Handler receiving the status of the GPIOs specified by the mask. Must be NULL if no polling is requested. The following type is used:

```
typedef void (*adl_ioHdlr_f) ( u8 GpioHandle, u32 GpioState );
```

GpioHandle: handle on which the polling GPIOs are allocated

GpioState: mask of read values on polling GPIOs.

This handler is called every time the "GpioState" value changes (ie. one of the allocated GPIOs has changed).

- **Returned values**

- A positive or null GPIO handle on success,
- ADL_RET_ERR_PARAM if a parameter has an incorrect value,
- ADL_RET_ERR_ALREADY_SUBSCRIBED if a requested GPIO was not free,
- ADL_RET_ERR_FATAL if a fatal error occurred (a ADL_ERR_IO_ALLOCATE error event will also be sent)

- **Note:**

Some product hardware related features (UART2, external battery charging mechanism on Q2501) may lock some GPIOs, which will not be available for allocation by the application; please refer to the corresponding section for more information.

3.7.3 The `adl_ioUnsubscribe` function

This function unsubscribes from a previously allocated GPIO handle.

- **Prototype**

```
s8      adl_ioUnsubscribe ( u8      Handle );
```

- **Parameters**

Handle:

Handle previously returned by a call to `adl_ioSubscribe` function.

- **Returned values**

- OK on success.
- ADL_RET_ERR_UNKNOWN_HDL if the provided handle is unknown
- ADL_RET_ERR_FATAL if a fatal error occurred (a ADL_ERR_IO_RELEASE error event will also be sent)

3.7.4 The `adl_ioRead` function

This function reads all GPIOs from a previously allocated handle.

- **Prototype**

```
s32    adl_ioRead        ( u8        Handle );
```

- **Parameters**

Handle:

Handle previously returned by a call to `adl_ioSubscribe` function.

- **Returned values**

The function returns:

- the Gpio read values mask on success
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle is unknown
- `ADL_RET_ERR_FATAL` if a fatal error has occurred
- `ADL_RET_ERR_BAD_STATE` if there is nothing to read corresponding to the handle

3.7.5 The `adl_ioWrite` function

This function writes on one or more GPIOs from a previously allocated handle.

- **Prototype**

```
s8      adl_ioWrite      ( u8        Handle,  
                          u32        GpioMask,  
                          u32        GpioValues );
```

- **Parameters**

Handle:

Handle previously returned by a call to `adl_ioSubscribe` function.

GpioMask:

Mask of GPIO to write.

GpioValues:

Mask of GPIO values to write.

- **Returned values**

- OK on success.
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle provided is unknown
- `ADL_RET_ERR_PARAM` if one parameter has an incorrect value
- `ADL_RET_ERR_FATAL` if a fatal error occurred (a `ADL_ERR_IO_WRITE` error event will also be sent)

3.7.6 The `adl_io GetProductType` function

This function returns the product type.

- **Prototype**

```
adl_ioProductTypes_e adl_ioGetProductType ( void );
```

- **Returned values**

This function returns the product type, with the following defined values:

`ADL_IO_PRODUCT_TYPE_Q24X3` *(for Q23X3 and Q24X3 products)*

`ADL_IO_PRODUCT_TYPE_Q24X6`

`ADL_IO_PRODUCT_TYPE_P32X3` *(for P31X3 and P32X3 products)*

`ADL_IO_PRODUCT_TYPE_P32X6`

`ADL_IO_PRODUCT_TYPE_Q31X6`

`ADL_IO_PRODUCT_TYPE_P5186`

`ADL_IO_PRODUCT_TYPE_Q24X0`

`ADL_IO_PRODUCT_TYPE_Q25X1`

`ADL_IO_PRODUCT_TYPE_Q24CLASSIC`

`ADL_IO_PRODUCT_TYPE_Q24PLUS`

`ADL_IO_PRODUCT_TYPE_Q24AUTO`

`ADL_IO_PRODUCT_TYPE_Q24EXTENDED`

3.8 Bus Service

ADL provides a bus service to handle all SPI, I2C soft, I2C hard and Parallel bus operations.

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

3.8.1 Required Header File

The header file for the bus functions is:
adl_bus.h

3.8.2 The adl_busSubscribe function

This function subscribes to a specific bus type.

- **Prototype**

```
s8      adl_busSubscribe ( u32      BusAddress,  
                          u32      Param );
```

- **Parameters**

BusAddress:

Type and address of the bus to subscribe to, using the following defined values, by performing a logical OR between **type** and **address**.

ADL User Guide for Open AT® OS v3.13

API

	<i>Possible type</i> values	<i>Possible address</i> values
SPI bus	ADL_BUS_TYPE_SPI	<p>ADL_BUS_SPI_ADDR_CS_SPI_EN: use SPI_EN pin as Chip Select <i>(for Q24X6, Q2400, Q24 Classic, Q24 Plus, Q24 Extended and Q24 Auto products, this setting is automatically mapped on GPO 3 used as Chip Select ; for P32X6 product, this setting is automatically mapped on GPIO 8 used as Chip Select);</i> <i>Not available for P5186 product).</i></p> <p>ADL_BUS_SPI_ADDR_CS_SPI_AUX: use SPI_AUX pin as Chip Select <i>(for Q24X6, Q2400, P32X6, Q24 Classic, Q24 Plus, Q24 Extended and Q24 Auto products, this setting is automatically mapped on GPO 0 used as Chip Select ;</i> <i>Not available for P5186 product</i> <i>Not available for Q31X6 product).</i></p> <p>ADL_BUS_SPI_ADDR_CS_GPIO : a GPIO or GPO is used as Chip Select. The used GPIO index is given by a logical OR with the index defined in IO service <i>This IO must not be allocated by any application.</i></p> <p>ADL_BUS_SPI_ADDR_CS_NONE The Chip Select signal is not handled by the ADL BUS service. The application should subscribe to a GPIO in order to handle the SPI Chip Select signal.</p>
IC2 soft bus	ADL_BUS_TYPE_I2C_SOFT	Less Significant Byte of BusAddress parameter is used as 7 bits slave address for devices on I2C bus.

ADL User Guide for Open AT® OS v3.13

API

	<i>Possible type</i> values	<i>Possible address</i> values
IC2 hard bus	ADL_BUS_TYPE_I2C_HARD	Less Significant Byte of BusAddress parameter is used as 7 bits slave address for devices on I2C bus (for <u>Q24X6, Q2400, Q3106, Q24 Classic, Q24 Plus, Q24 Extended and Q24 Auto</u> products).
Parallel bus	ADL_BUS_TYPE_PARALLEL	<p>ADL_BUS_PARA_LCDEN_AS_CS: use LCD_EN pin as Chip Select <u>On P32X6 product, the LCD_EN pin is the same as the GPIO 8 pin; it must not be allocated by any application.</u></p> <p>ADL_BUS_PARA_CSUSR_AS_CS: use CS_USER pin as Chip Select (GPIO 5 on Pac products, GPIO 3 on Q31X6 product). <u>This GPIO pin must not be allocated by any application.</u></p>

Param:

Bus parameters, defined by following values, using a logical OR to combine the different settings:

ADL User Guide for Open AT® OS v3.13

API

for SPI bus:

- o Clock speed:

Speed constant	Supported on Q2XX3 and P3XX3 products	Supported Q24 Classic, Q24 Plus, Q24 Extended, Q24 Auto and products	Supported on P5186 product
ADL_BUS_SPI_SCL_SPEED_13Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_6_5Mhz		Yes	Yes
ADL_BUS_SPI_SCL_SPEED_4_33Mhz		Yes	Yes
ADL_BUS_SPI_SCL_SPEED_3_25Mhz	Yes	Yes	Yes
ADL_BUS_SPI_SCL_SPEED_2_6Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_2_167Mhz		Yes	Yes
ADL_BUS_SPI_SCL_SPEED_1_857Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_1_625Mhz	Yes	Yes	
ADL_BUS_SPI_SCL_SPEED_1_44Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_1_3Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_1_181Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_1_083Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_1Mhz		Yes	
ADL_BUS_SPI_SCL_SPEED_926Khz		Yes	
ADL_BUS_SPI_SCL_SPEED_867Khz		Yes	
ADL_BUS_SPI_SCL_SPEED_812Khz	Yes	Yes	
ADL_BUS_SPI_SCL_SPEED_101Khz	Yes		

- o Clock mode:

ADL_BUS_SPI_CLK_MODE_0

(the rest state is 0, data valid on rising edge)

ADL_BUS_SPI_CLK_MODE_1

(the rest state is 0, data valid on falling edge)

ADL_BUS_SPI_CLK_MODE_2

(the rest state is 1, data valid on rising edge)

ADL_BUS_SPI_CLK_MODE_3

(the rest state is 1, data valid on falling edge)

ADL User Guide for Open AT® OS v3.13

API

- **Chip Select Polarity:**
ADL_BUS_SPI_CS_POL_LOW, *for low polarity*
ADL_BUS_SPI_CS_POL_HIGH, *for high polarity*
- **Lsb or Msb first:**
ADL_BUS_SPI_MSB_FIRST, *to send data MSB first*
ADL_BUS_SPI_LSB_FIRST, *to send data LSB first*
- **Gpio Handling:**
(only when an IO is used as Chip Select)
ADL_BUS_SPI_BYTE_HANDLING,
the IO signal pulse on each data byte,
ADL_BUS_SPI_FRAME_HANDLING,
the IO signal works as a normal chip select.

For I2C Soft-bus:

- **SCL signal GPIO:**
The GPIO index to use to handle the SCL signal (shifted to the two MSBytes)
- **SDA signal GPIO:**
The GPIO index to use to handle the SDA signal (on the two LSBytes)

Remark: the ADL_IO_ID_U32_TO_U16 macro should be used to convert the used GPIO ID to u16 type before calling the API.

Example:

```
Adl_busSubscribe( ADL_BUS_TYPE_IC2_SOFT,  
                 ADL_IO_ID_U32_TO_U16(MySDAGpio) |  
                 (ADL_IO_ID_U32_TO_U16(MySCLGpio)<<16) );
```

For I2C Hard bus:

- **Clk Speed:**
The Clk_Speed parameter sets the required I2C bus speed. Defined values are:
 - ADL_BUS_I2C_HARD_CLK_STD (standard I2C bus speed, 100 Kbit/s)
 - ADL_BUS_I2C_HARD_CLK_FAST (fast I2C bus speed, 400 Kbit/s)

For Parallel bus:

- **Data Order:**
ADL_BUS_PARA_DATA_DIRECT_ORDER,
to send data on direct order
ADL_BUS_PARA_DATA_REVERSE_ORDER,
to send data on reverse order

ADL User Guide for Open AT® OS v3.13

API

- **LCD_EN signal polarity (only for LCD_EN chip select):**
ADL_BUS_PARA_LCDEN_POL_LOW
data is sampled on the rising edge from low state to high state of LCD_EN.
ADL_BUS_PARA_LCDEN_POL_HIGH
data is sampled on the falling edge from high state to low state of LCD_EN.

- **LCD_EN Address Setup Time (only for LCD_EN chip select):**
 This is the time interval between the setting of an address for the Parallel bus and the activation of the LCD_EN pin. It is the T1 time in the figure below.
 The allowed values are from 0 to 31 (using bits 0 to 4).
 The resulting time interval is:
*For P32X3 product: (X * 38.5) ns ;*
*For P32X6 product: (1 + 2 X) * 19 ns.*

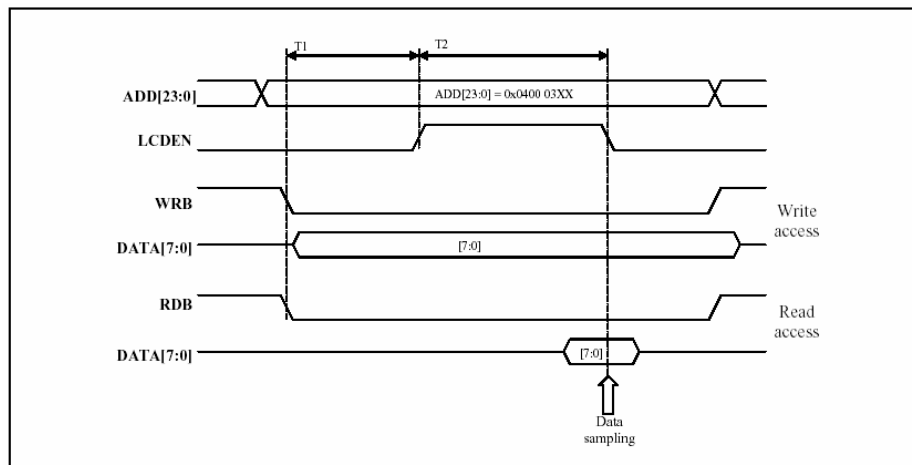


Figure 4: LCD_EN Address Setup chronogram

- **LCD_EN Signal Pulse Duration (only for LCD_EN chip select):**
 This is the time interval during which the LCD_EN pin is valid. It is the T2 time in the figure above.
 The allowed values are from 0 to 31 (using bits 5 to 10).
 The resulting time interval is:
*For P32X3 product: (X + 1.5) * 38.5 ns ;*
*For P32X6 product: (1 + 2 * (X + 1)) * 19 ns.*
(Warning, for the P32X6 product, the 0 value is considered as 32).

ADL User Guide for Open AT® OS v3.13

API

- **CS_USER number of wait states (only for CS_USER chip select):**
This is the time interval during which the data is valid on the bus, using the defined values:

- ADL_BUS_PARA_CSUSR_0_WAIT_STATE (62 ns)
- ADL_BUS_PARA_CSUSR_1_WAIT_STATE (100 ns)
- ADL_BUS_PARA_CSUSR_2_WAIT_STATE (138 ns)
- ADL_BUS_PARA_CSUSR_3_WAIT_STATE (176 ns)

- **Returned values**

A positive or null bus handle on success.

ADL_RET_ERR_PARAM if one parameter has an incorrect value

ADL_RET_ERR_ALREADY_SUBSCRIBED if requested bus and address is already subscribed

For other negative errors, please refer to the BUS API chapter of the Open AT® Basic Development Guide.

- **Remark**

If one or more IOs are required to open a bus, these IOs must not be subscribed by any application. On the bus unsubscribe operation, the IOs can be subscribed again.

3.8.3 The `adl_busUnsubscribe` function

This function unsubscribes from a previously subscribed bus type

- **Prototype**

```
s8      adl_busUnsubscribe ( u8      Handle );
```

- **Parameters**

Handle:

Handle previously returned by `adl_busSubscribe` function.

- **Returned values**

- OK on success.
- ADL_RET_ERR_UNKNOWN_HDL if the provided handle is unknown.
- For other negative errors, please refer to the BUS API chapter of the Open AT® Basic Development Guide.

3.8.4 The `adl_busRead` function

This function reads data from a previously subscribed bus type

- **Prototype**

```
s8      adl_busRead      (u8      Handle,
                        adl_busAccess_t *pAccessMode,
                        u32      DataLen,
                        void *    Data );
```

- **Parameters**

Handle:

Handle previously returned by `adl_busSubscribe` function.

pAccessMode:

Bus access mode, defined according to the following type:

```
typedef struct
{
    u32 Address;
    u32 Opcode;
    u8  OpcodeLength;
    u8  AddressLength;
    u8  AccessSize; (reserved for future products)
} adl_busAccess_t;
```

This parameter is processed differently according to bus type:

- **For SPI bus:**

For Q24X3 and P32X3 products:

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

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

the **OpcodeLength** is the sum of **Opcode** and **Address** lengths in bits (if **OpcodeLength** is 0, nothing is sent; if **OpcodeLength** < 9, just **Opcode** is sent; if 8 < **OpcodeLength** < 25, **Opcode** then **Address** are sent),

the **AddressLength** parameter is not used.

ADL User Guide for Open AT® OS v3.13

API

For Q24X6, Q2400 P32X6, Q24 Classic, Q24 Plus, Q24 Extended and Q24 Auto products:

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

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

- **For I2C Soft bus:**
Not used, this parameter should be NULL.
- **For I2C hard bus:**
Not used, this parameter should be NULL.
- **For Parallel bus:**
Only the **Address** parameter is used.
This parameter is used to set the A2 pin value ; it can be set to the following values:
WM_BUS_PARA_ADDRESS_A2_SET, to set the A2 pin;
WM_BUS_PARA_ADDRESS_A2_RESET, to reset the A2 pin

DataLen:
Number of bytes to read from the bus.

Data:
Buffer to which the read bytes are to be copied.

- **Returned values**
 - OK on success.
 - ADL_RET_ERR_UNKNOWN_HDL if the handle provided is unknown,
 - ADL_RET_ERR_PARAM if a parameter has an incorrect value,
 - For other negative errors, please refer to the BUS API chapter of the Open AT® Basic Development Guide.

3.8.5 The adl_busWrite function

This function writes on a previously subscribed bus.

- **Prototype**

```
s8      adl_busWrite      ( u8      Handle,
                           adl_busAccess_t * pAccessMode,
                           u32      DataLen,
                           void *   Data );
```

- **Parameters**

Handle:

Handle previously returned by adl_busSubscribe function.

pAccessMode:

Bus access mode, defined with the following type:

```
typedef struct
{
    u32 Address;
    u32 Opcode;
    u8 OpcodeLength;
    u8 AddressLength;
    u8 AccessSize; (reserved for future products)
} adl_busAccess_t;
```

This parameter is processed differently according to bus type:

- **For SPI bus:**

- For Q24X3 and P32X3 products:

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

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

the **OpcodeLength** is the sum of **Opcode** and **Address** lengths in bits
(if **OpcodeLength** is 0, nothing is sent;
if **OpcodeLength** < 9, just **Opcode** is sent;
if 8 < **OpcodeLength** < 25, **Opcode** then **Address** are sent),

the **AddressLength** parameter is not used.

For Q24X6, Q2400 P32X6, Q24 Classic, Q24 Plus, Q24 Extended and Q24 Auto products:

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

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

ADL User Guide for Open AT® OS v3.13

API

- **For I2C Soft bus:**
Not used, this parameter should be NULL.
- **For I2C hard bus:**
Not used, this parameter should be NULL.
- **For Parallel bus:**
Only the **Address** parameter is used.
This parameter is used to set the A2 pin value; it can be set to following values:
WM_BUS_PARA_ADDRESS_A2_SET, to set the A2 pin;
WM_BUS_PARA_ADDRESS_A2_RESET, to reset the A2 pin

DataLen:

Number of bytes to write on the bus.

2

Data:

Data buffer to write on the bus.

- **Returned values**

OK on success.

ADL_RET_ERR_UNKNOWN_HDL if the handle provided is unknown,

ADL_RET_ERR_PARAM if a parameter has an incorrect value,

For other negative errors, please refer to the BUS API chapter of the Open AT® Basic Development Guide.

3.9 Errors management

3.9.1 Required Header File

The header file for the error functions is:
adl_errors.h

3.9.2 The adl_errSubscribe function

This function subscribes to the management service and gives an error handler: this allows the application to handle errors generated by ADL or by the `adl_errHalt` function. Errors generated by the Wavecom Core Software cannot be handled by such an error handler.

- **Prototype**

```
s8      adl_errSubscribe      ( adl_errHdlr_f  Handler );
```

- **Parameters**

Handler:

Error Handler, defined on following type:

```
typedef bool ( * adl_errHdlr_f ) ( u16 ErrorID, ascii * ErrorStr );
```

An error is described by an Id and a string (associated text), that are sent as parameters to the `adl_errHalt` function.

If the error is processed and filtered the handler should return FALSE. The return value TRUE will cause the product to execute a fatal error reset with a backtrace.

A backtrace is composed of the message provided, and a call stack "footprint" taken at the function call time. It is readable by the Target Monitoring Tool (Please refer to the Tools Manual for more information).

Note that **ErrorIDs** below 0x0100 are for internal purposes so the application should only use **ErrorIDs** above 0x0100.

ADL may generates errors which will be handled by an error handler:

ErrorID	ADL function	Cause
ADL_ERR_MEM_GET	adl_memGet	The product ran out of heap memory, or the heap memory is composed of free blocks smaller than the required size.
ADL_ERR_MEM_RELEASE	adl_memRelease	The pointer provided was not provided by the <code>adl_memGet</code> function, or it was already released.

ADL User Guide for Open AT® OS v3.13

API

ErrorID	ADL function	Cause
ADL_ERR_IO_ALLOCATE	adl_ioSubscribe	Abnormal error on Gpio subscription: should be reported to Wavecom support.
ADL_ERR_IO_RELEASE	adl_ioUnsubscribe	Abnormal error on Gpio unsubscription: should be reported to Wavecom support.
ADL_ERR_IO_READ	adl_ioRead	Abnormal error on Gpio read: should be reported to Wavecom support.
ADL_ERR_IO_WRITE	adl_ioWrite	Abnormal error on Gpio write: should be reported to Wavecom support.
ADL_ERR_FLH_READ	adl_flhRead	Abnormal error on Flash object read: should be reported to Wavecom support.
ADL_ERR_FLH_DELETE	adl_flhErase	Abnormal error on Flash object erasure: should be reported to Wavecom support.

- **Returned values**

- OK on success.
- ADL_RET_ERR_PARAM if the parameter has an incorrect value
- ADL_RET_ERR_ALREADY_SUBSCRIBED if the service is already subscribed

- **Returned values**

The reboot is performed once the handler has returned TRUE. In order to ensure the downloading of a new binary file after a fatal error has been detected, the Open AT® application software startup is performed after a 20-second delay.

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

Moreover, if the product reset is due to a fatal error (from Open AT® application, or from Wavecom Core Software), the `adl_main` function's `adlInitType` parameter will be set to the `ADL_INIT_REBOOT_FROM_EXCEPTION` value.

3.9.3 The `adl_errUnsubscribe` function

This function unsubscribes from Management service. Errors generated by ADL or by the `adl_errHalt` function will no longer be handled by the error handler.

- **Prototype**

```
s8 adl_errUnsubscribe (adl_errHdlr_f Handler);
```

- **Parameters**

Handler:

Handler returned by `adl_errSubscribe` function

- **Returned values**

- OK on success.
- ADL_RET_ERR_PARAM if the parameter has an incorrect value
- ADL_RET_ERR_UNKNOWN_HDL if the handler provided is unknown
- ADL_RET_ERR_NOT_SUBSCRIBED if the service is not subscribed

3.9.4 The `adl_errHalt` function

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.

- **Prototype**

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

ErrorID:
Error ID

ErrorString:

Error string to be provided to the error handler and to be stored in the resulting backtrace if a fatal error is required.

Please note that only the string address is stored in the backtrace, so this parameter must not be a pointer on a RAM buffer, but a constant string pointer. Moreover, the string will only be correctly displayed if the current application is still present in the module's flash memory. If the application is erased or modified, the string will not be correctly displayed when retrieving the backtraces.

3.9.5 The `adl_errEraseAllBacktraces` function

Backtraces (caused by the `adl_errHalt` function, ADL or the Wavecom Core Software) are stored in the product's 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 `adl_errEraseAllBacktraces` function allows this storage place to be freed and re-initialized.

- **Prototype**

`void adl_errEraseAllBacktraces (void);`

3.9.6 The `adl_errStartBacktraceAnalysis` function

In order to retrieve backtraces from product memory, a backtrace analysis process has to be started with the `adl_errStartBacktraceAnalysis` function.

- **Prototype**

```
s8      adl_errStartBacktraceAnalysis ( void );
```

- **Returned values**

- A positive or null handle on success. This handle must be used in the next `adl_errRetrieveNextBacktrace` function call. It will be valid until this function returns a `ADL_RET_ERR_DONE` code.
- `ADL_RET_ERR_ALREADY_SUBSCRIBED` if a backtrace analysis is already running.
- `ERROR` if an unexpected internal error occurred.

- **Note**

Only one analysis may be running at a time. The `adl_errStartBacktraceAnalysis` function will return the `ADL_RET_ERR_ALREADY_SUBSCRIBED` error code if it is called while an analysis is currently running.

3.9.7 The `adl_errGetAnalysisState` function

This function may be used in order to know the current backtrace analysis process state.

- **Prototype**

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

- **Returned values**

Current backtrace analysis state, which uses the type below:

```
typedef enum
{
    ADL_ERR_ANALYSIS_STATE_IDLE,           // No running analysis
    ADL_ERR_ANALYSIS_STATE_RUNNING       // An analysis is running
} adl_errAnalysisState_e;
```

3.9.8 The `adl_errRetrieveNextBacktrace` function

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

ADL User Guide for Open AT® OS v3.13

API

- **Prototype**

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

- **Parameters**

Handle:

Backtrace analysis handle, returned by the `adl_errStartBacktraceAnalysis` function.

BacktraceBuffer:

Buffer in which the next retrieved backtrace will be copied. This parameter may be set to `NULL` in order to know the required size of the next backtrace buffer.

Size:

Backtrace buffer size. If this size is not large enough, the `ADL_RET_ERR_PARAM` error code will be returned.

- **Returned values**

- OK if the next stored backtrace was successfully copied in the `BacktraceBuffer` parameter.
- The required size for the next backtrace buffer if the `BacktraceBuffer` parameter is set to `NULL`.
- `ADL_RET_ERR_PARAM` if the provided `Size` parameter is not large enough.
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the `adl_errStartBacktraceAnalysis` function was not called before.
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided `Handle` parameter is invalid.
- `ADL_RET_ERR_DONE` if the last backtrace buffer has already been retrieved. The `Handle` parameter will now be unsubscribed and not usable any more with the `adl_errRetrieveNextBacktrace` function. A new analysis has to be started with the `adl_errStartBacktraceAnalysis` function.

- **Note**

Once retrieved, the backtrace buffers may be stored (separately or concatenated), in order to be sent (using the application's protocol/bearer choice) to a remote server or PC. Once retrieved as one or several files on a PC, this(these) may be read using the Target Monitoring Tool and the Serial Link Manager in order to decode the backtrace buffer(s). Please refer to the Tools Manual in order to know how to process these files.

3.10 SIM Service

ADL provides this service to handle SIM and PIN code related events.

3.10.1 Required Header File

The header file for the SIM related functions is:

adl_sim.h

3.10.2 The adl_simSubscribe function

This function subscribes to the SIM service, in order to receive SIM and PIN code related events. This will allow PIN code (if provided) to be entered if necessary.

- **Prototype**

```
void adl_simSubscribe ( adl_simHdlr_f   SimHandler,  
                      ascii *        PinCode );
```

- **Parameters**

SimHandler:

SIM handler defined using the following type:

```
typedef void ( * adl_simHdlr_f ) ( u8 Event );
```

The events received by this handler are defined below.

Normal events:

```
ADL_SIM_EVENT_PIN_OK  
    if PIN code is all right  
ADL_SIM_EVENT_REMOVED  
    if SIM card is removed  
ADL_SIM_EVENT_INSERTED  
    if SIM card is inserted  
ADL_SIM_EVENT_FULL_INIT  
    when initialization is done
```

Error events:

```
ADL_SIM_EVENT_PIN_ERROR  
    if given PIN code is wrong  
ADL_SIM_EVENT_PIN_NO_ATTEMPT  
    if there is only one attempt left to entered the right PIN code  
ADL_SIM_EVENT_PIN_WAIT  
    if the argument PinCode is set to NULL
```

On the last three events, the service is waiting for the external application to enter the PIN code.

Please note that the deprecated ADL_SIM_EVENT_ERROR event has been removed since ADL version 3. This code was mentioned in the version 2 documentation, but was never generated by the SIM service.

PinCode:

This is a string containing the PIN code text to enter. If it is set to NULL or if the code provided is incorrect, the PIN code will have to be entered by the external application.

This argument is used only the first time the service is subscribed. It is ignored on all further subscriptions.

3.10.3 The adl_simUnsubscribe function

This function unsubscribes from SIM service. The handler provided will no longer receive SIM events.

• **Prototype**

```
void adl_simUnsubscribe ( adl_simHdlr_f Handler)
```

• **Parameters**

Handler:

Handler used with adl_SimSubscribe function.

3.10.4 The adl_simGetState function

This function gets the current SIM service state.

• **Prototype**

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

• **Returned values**

The returned value is the SIM service state, based on following type:

```
typedef enum
{
    ADL_SIM_STATE_INIT, // Service init state (PIN state not known yet)
    ADL_SIM_STATE_REMOVED, // SIM removed
    ADL_SIM_STATE_INSERTED, // SIM inserted (PIN state not known yet)
    ADL_SIM_STATE_FULL_INIT, // SIM Full Init done
    ADL_SIM_STATE_PIN_ERROR, // SIM error state
    ADL_SIM_STATE_PIN_OK, // PIN code OK, waiting for full init
    ADL_SIM_STATE_PIN_WAIT, // SIM inserted, PIN code not entered yet

    /* Always last State */
    ADL_SIM_STATE_LAST
} adl_simState_e;
```


3.11 SMS Service

ADL provides this service to handle SMS events, and to send SMS to the network.

3.11.1 Required Header File

The header file for the SMS related functions is:

`adl_sms.h`

3.11.2 The `adl_smsSubscribe` function

This function subscribes to the SMS service in order to receive SMSs from the network.

- **Prototype**

```
s8      adl_smsSubscribe ( adl_smsHdlr_f      SmsHandler,  
                          adl_smsCtrlHdlr_f  SmsCtrlHandler,  
                          u8                  Mode );
```

- **Parameters**

SmsHandler:

SMS handler defined using the following type:

```
typedef bool ( * adl_smsHdlr_f ) ( ascii * SmsTel,  
                                  ascii * SmsTimeLength,  
                                  ascii * SmsText );
```

This handler is called each time a SMS is received from the network.

SmsTel contains the originating telephone number of the SMS (in text mode), or NULL (in PDU mode).

SmsTimeLength contains the SMS time stamp (in text mode), or the PDU length (in PDU mode).

SmsText contains the SMS text (in text mode), or the SMS PDU (in PDU mode).

This handler returns TRUE if the SMS must be forwarded to the external application (it is then stored in SIM memory, and the external application is then notified by a "+CMTI" unsolicited indication).

It returns FALSE if the SMS should not be forwarded.

If the SMS service is subscribed several times, a received SMS will be forwarded to the external application only if each of the handlers return TRUE.

SmsCtrlHandler:

SMS event handler, defined using the following type:

```
typedef void ( * adl_smsCtrlHdlr_f ) ( u8 Event, u16 Nb );
```

This handler is notified by the following events during an SMS sending process.

ADL_SMS_EVENT_SENDING_OK

*the SMS was sent successfully, **Nb** parameter value is not relevant.*

ADL_SMS_EVENT_SENDING_ERROR

*An error occurred during SMS sending, **Nb** parameter contains the error number, according to "+CMS ERROR" value (cf. AT Commands Interface Guide).*

ADL_SMS_EVENT_SENDING_MR

*the SMS was sent successfully, **Nb** parameter contains the sent Message Reference value. A ADL_SMS_EVENT_SENDING_OK event will be received by the control handler.*

Mode:

Mode used for SMS reception from the following values:

ADL_SMS_MODE_PDU

SmsHandler will be called in PDU mode on each SMS reception.

ADL_SMS_MODE_TEXT

SmsHandler will be called in Text mode on each SMS reception.

• **Returned values**

- On success, this function returns a positive or null handle, requested for further SMS sending operations.
- ADL_RET_ERR_PARAM if a parameter has a wrong value.

3.11.3 The `adl_smsSend` function

This function sends a SMS to the network.

- **Prototype**

```
s8      adl_smsSend      ( u8      Handle,  
                          ascii *   SmsTel,  
                          ascii *   SmsText,  
                          u8      Mode );
```

- **Parameters**

Handle:

Handle returned by `adl_smsSubscribe` function.

SmsTel:

Telephone number to which the SMS is to be sent (in text mode), or NULL (in PDU mode).

SmsText:

SMS text (in text mode), or SMS PDU (in PDU mode).

Mode:

Mode used for SMS sending from the following values:

```
ADL_SMS_MODE_PDU  
    to send a SMS in PDU mode.  
ADL_SMS_MODE_TEXT  
    to send a SMS in Text mode.
```

- **Returned values**

- This function returns OK on success.
- `ADL_RET_ERR_PARAM` if a parameter has a wrong value.
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle provided is unknown.
- `ADL_RET_ERR_BAD_STATE` if the product is not ready to send a SMS (initialization not yet performed, or SMS sending already in progress)

3.11.4 The `adl_smsUnsubscribe` function

This function unsubscribes from the SMS service. The associated handler with the handle provided will no longer receive SMS events.

- **Prototype**

```
s8      adl_smsUnsubscribe ( u8      Handle )
```

- **Parameters**

Handle:

Handle returned by `adl_smsSubscribe` function.

- **Returned values**

- OK on success.
- `ADL_RET_ERR_UNKNOWN_HDL` if the handler provided is unknown.
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the service is not subscribed.
- `ADL_RET_ERR_BAD_STATE` if the service is processing a SMS

3.12 Call Service

ADL provides this service to handle call-related events, and to setup calls.

3.12.1 Required Header File

The header file for the call related functions is:

`adl_call.h`

3.12.2 The `adl_callSubscribe` function

This function subscribes to the call service in order to receive call-related events.

- **Prototype**

```
s8      adl_callSubscribe ( adl_callHdlr_f CallHandler );
```

- **Parameters**

CallHandler:

Call handler defined using the following type:

```
typedef s8 ( * adl_callHdlr_f ) ( u16 Event, u32 Call_ID );
```

The pairs events / call Id received by this handler are defined below ; each event is received according to an "event type", which can be :

- MO (*Mobile Originated call-related event*)
- MT (*Mobile Terminated call-related event*)
- CMD (*Incoming AT command-related event*)

Event / Call ID	Description	Type
ADL_CALL_EVENT_RING_VOICE / 0	<i>if voice phone call</i>	MT
ADL_CALL_EVENT_RING_DATA / 0	<i>if data phone call</i>	MT
ADL_CALL_EVENT_NEW_ID / X	<i>if wind: 5,X</i>	MO
ADL_CALL_EVENT_RELEASE_ID / X	<i>if wind: 6,X; on data call release, X is a logical OR between the Call ID and the ADL_CALL_DATA_FLAG constant</i>	MO MT
ADL_CALL_EVENT_ALERTING / 0	<i>if wind: 2</i>	MO
ADL_CALL_EVENT_NO_CARRIER / 0	<i>phone call failure, 'NO CARRIER'</i>	MO MT
ADL_CALL_EVENT_NO_ANSWER / 0	<i>phone call failure, no answer</i>	MO
ADL_CALL_EVENT_BUSY / 0	<i>phone call failure, busy</i>	MO

ADL User Guide for Open AT® OS v3.13

API

Event / Call ID	Description	Type
ADL_CALL_EVENT_SETUP_OK / Speed	<i>ok response after a call setup performed by the <code>adl_callSetup</code> function; in data call setup case, the connection <Speed> (in bits/second) is also provided.</i>	MO
ADL_CALL_EVENT_ANSWER_OK / Speed	<i>ok response after an <code>ADL_CALL_NO_FORWARD_ATA</code> request from a call handler; in data call answer case, the connection <Speed> (in bps) is also provided</i>	MT
ADL_CALL_EVENT_HANGUP_OK / Data	<i>ok response after a <code>ADL_CALL_NO_FORWARD_ATH</code> request, or a call hangup performed by the <code>adl_callHangup</code> function; on data call release, Data is the <code>ADL_CALL_DATA_FLAG</code> constant (0 on voice call release)</i>	MO MT
ADL_CALL_EVENT_SETUP_OK_FROM_EXT / Speed	<i>ok response after an 'ATD' command from the external application; in data call setup case, the connection <Speed> (in bits/second) is also provided.</i>	MO
ADL_CALL_EVENT_ANSWER_OK_FROM_EXT / Speed	<i>ok response after an 'ata' command from the external application; in data call answer case, the connection <Speed> (in bps) is also provided</i>	MT
ADL_CALL_EVENT_HANGUP_OK_FROM_EXT / Data	<i>ok response after an 'ATH' command from the external application; on data call release, Data is the <code>ADL_CALL_DATA_FLAG</code> constant (0 on voice call release)</i>	MO MT
ADL_CALL_EVENT_AUDIO_OPENED / 0	<i>if +WIND: 9</i>	MO MT
ADL_CALL_EVENT_ANSWER_OK_AUTO / Speed	<i>OK response after an auto-answer to an incoming call (ATS0 command was set to a non-zero value); in data call answer case, the connection <Speed> (in bps) is also provided</i>	MT
ADL_CALL_EVENT_RING_GPRS / 0	<i>if GPRS phone call</i>	MT

ADL User Guide for Open AT® OS v3.13

API

Event / Call ID	Description	Type
ADL_CALL_EVENT_SETUP_FROM_EXT / Mode	<i>if the external application has used the 'ATD' command to set up a call. Mode value depends on call type (Voice: 0, GSM Data: ADL_CALL_DATA_FLAG, GPRS session activation: binary OR between ADL_CALL_GPRS_FLAG constant and the activated CID). According to the notified handlers return values, call setup may be launched or not: if at least one handler returns the ADL_CALL_NO_FORWARD code (or higher), the command will reply "+CME ERROR: 600" to the external application; otherwise (if all handlers return ADL_CALL_FORWARD), the call setup is launched.</i>	CMD
ADL_CALL_EVENT_CIEV	<i>OK response after a call setup was performed</i>	MO
ADL_CALL_EVENT_SETUP_ERROR_NO_SIM / 0	<i>A call setup (from embedded or external application) has failed (no SIM card inserted)</i>	MO
ADL_CALL_EVENT_SETUP_ERROR_PIN_NOT_READY / 0	<i>A call setup (from embedded or external application) has failed (the PIN code is not entered)</i>	MO
ADL_CALL_EVENT_SETUP_ERROR / Error	<i>A call setup (from embedded or external application) has failed (the <Error> field is the returned +CME ERROR value; cf. AT Commands interface guide for more information)</i>	MO

The events returned by this handler are defined below:

Event	Description
ADL_CALL_FORWARD	<i>the call event shall be sent to the external application On unsolicited events, these will be forwarded to all opened ports. On responses events, these will be forwarded only on the port on which the request was executed.</i>
ADL_CALL_NO_FORWARD	<i>the call event shall not be sent to the external application</i>

Event	Description
ADL_CALL_NO_FORWARD_ATH	<i>the call event shall not be sent to the external application and the application shall terminate the call by sending an 'ATH' command.</i>
ADL_CALL_NO_FORWARD_ATA	<i>the call event shall not be sent to the external application and the application shall answer the call by sending an 'ATA' command.</i>

- **Returned values**

- OK on success
- ADL_RET_ERR_PARAM on parameter error

3.12.3 The `adl_callSetup` function

This function just runs the `adl_callSetupExt` one on the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` port (cf. `adl_callSetupExt` description for more information). 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® port.

3.12.4 The `adl_callSetupExt` function

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

- **Prototype**

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

- **Parameters**

PhoneNb:

Phone number to use to set up the call.

Mode:

Mode used to set up the call:

ADL_CALL_MODE_VOICE,
ADL_CALL_MODE_DATA

Port:

Port on which to run the call setup command. When setup return events are received in the Call event handler, if the application requires ADL to forward these events, they will be forwarded to this Port parameter value.

- **Returned values**

- OK on success
- ADL_RET_ERR_PARAM on parameter error (bad value, or unavailable port)

3.12.5 The `adl_callHangup` function

This function just runs the `adl_callHangupExt` one on the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` port (cf. `adl_callHangupExt` description for more information). Please note that events generated by the `adl_callHangup` will not be able to be forwarded to any external port, since the setup command was running on the Open AT® port.

3.12.6 The `adl_callHangupExt` function

This function hangs up the phone call.

- **Prototype**

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

- **Parameters**

Port:

Port on which to run the call hang-up command. When hang-up return events are received in the Call event handler, if the application requires ADL to forward these events, they will be forwarded to this Port parameter value.

- **Returned values**

- OK on success
- ADL_RET_ERR_PARAM on parameter error (unavailable port)

3.12.7 The `adl_callAnswer` function

This function just runs the `adl_callAnswerExt` one on the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` port (cf. `adl_callAnswerExt` description for more information). Please note that events generated by the `adl_callAnswer` will not be able to be forwarded to any external port, since the setup command was running on the Open AT® port.

3.12.8 The `adl_callAnswerExt` function

This function allows the application to answer a phone call out of the call events handler.

- **Prototype**

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

- **Parameters**

Port:

Port on which to run the call hang-up command. When hang-up return events are received in the Call event handler, if the application requires ADL to forward these events, they will be forwarded to this Port parameter value.

- **Returned values**

- OK on success
- ADL_RET_ERR_PARAM on parameter error (unavailable port)

3.12.9 The `adl_callUnsubscribe` function

This function unsubscribes from the Call service. The handler provided will no longer receive Call events.

- **Prototype**

```
s8 adl_callUnsubscribe ( adl_callHdlr_f Handler );
```

- **Parameters**

Handler:

Handler used with `adl_callSubscribe` function.

- **Returned values**

- OK on success
- ADL_RET_ERR_PARAM on parameter error
- ADL_RET_ERR_UNKNOWN_HDL if the handler provided is unknown
- ADL_RET_ERR_NOT_SUBSCRIBED if the service is not subscribed.

3.13 GPRS Service

ADL provides this service to handle GPRS related events and to set up, activate and deactivate PDP contexts.

3.13.1 Required Header File

The header file for the GPRS related functions is:
adl_gprs.h

3.13.2 The adl_gprsSubscribe function

This function subscribes to the GPRS service in order to receive GPRS related events.

- **Prototype**

```
s8      adl_gprsSubscribe ( adl_gprsHdlr_f GprsHandler );
```

- **Parameters**

GprsHandler:

GPRS handler defined using the following type:

```
typedef s8 (*adl_gprsHdlr_f)(u16 Event, u8 Cid);
```

The pairs events/Cid received by this handler are defined below:

Event / Call ID	Description
ADL_GPRS_EVENT_RING_GPRS	<i>If incoming PDP context activation is requested by the network</i>
ADL_GPRS_EVENT_NW_CONTEXT_DEACT / X	<i>If the network has forced the deactivation of the Cid X</i>
ADL_GPRS_EVENT_ME_CONTEXT_DEACT / X	<i>If the ME has forced the deactivation of the Cid X</i>
ADL_GPRS_EVENT_NW_DETACH	<i>If the network has forced the detachment of the ME</i>
ADL_GPRS_EVENT_ME_DETACH	<i>If the ME has forced a network detachment or lost the network</i>
ADL_GPRS_EVENT_NW_CLASS_B	<i>If the network has forced the ME on class B</i>
ADL_GPRS_EVENT_NW_CLASS_CG	<i>If the network has forced the ME on class CG</i>
ADL_GPRS_EVENT_NW_CLASS_CC	<i>If the network has forced the ME on class CC</i>
ADL_GPRS_EVENT_ME_CLASS_B	<i>If the ME has changed his class to class B</i>

ADL User Guide for Open AT® OS v3.13

API

Event / Call ID	Description
ADL_GPRS_EVENT_ME_CLASS_CG	<i>If the ME has changed his class to class CG</i>
ADL_GPRS_EVENT_ME_CLASS_CC	<i>If the ME has changed his class to class CC</i>
ADL_GPRS_EVENT_NO_CARRIER	<i>If the activation of the external application with 'ATD*99' (PPP dialing) did hang up.</i>
ADL_GPRS_EVENT_DEACTIVATE_OK / X	<i>If the deactivation requested with adl_gprsDeact() function did succeed on the Cid X</i>
ADL_GPRS_EVENT_DEACTIVATE_OK_FROM_EXT / X	<i>If the deactivation requested by the external application succeed on the Cid X</i>
ADL_GPRS_EVENT_ANSWER_OK	<i>If the acceptance of the incoming PDP activation with adl_gprsAct() did succeed</i>
ADL_GPRS_EVENT_ANSWER_OK_FROM_EXT	<i>If the acceptance of the incoming PDP activation by the external application did succeed</i>
ADL_GPRS_EVENT_ACTIVATE_OK / X	<i>If the activation requested with adl_gprsAct() on the Cid X did succeed</i>
ADL_GPRS_EVENT_GPRS_DIAL_OK_FROM_EXT / X	<i>If the activation requested by the external application with 'ATD*99' (PPP dialing) did succeed on the Cid X</i>
ADL_GPRS_EVENT_ACTIVATE_OK_FROM_EXT / X	<i>If the activation requested by the external application on the Cid X did succeed</i>
ADL_GPRS_EVENT_HANGUP_OK_FROM_EXT	<i>If the rejection of the incoming PDP activation by the external application did succeed</i>
ADL_GPRS_EVENT_DEACTIVATE_KO / X	<i>If the deactivation requested with adl_gprsDeact() on the Cid X did fail</i>
ADL_GPRS_EVENT_DEACTIVATE_KO_FROM_EXT / X	<i>If the deactivation requested by the external application on the Cid X did fail</i>
ADL_GPRS_EVENT_ACTIVATE_KO_FROM_EXT / X	<i>If the activation requested by the external application on the Cid X did fail</i>

ADL User Guide for Open AT® OS v3.13

API

Event / Call ID	Description
ADL_GPRS_EVENT_ACTIVATE_KO / X	<i>If the activation requested with <code>adl_gprsAct()</code> on the Cid X did fail</i>
ADL_GPRS_EVENT_ANSWER_OK_AUTO	<i>If the incoming PDP context activation was automatically accepted by the ME</i>
ADL_GPRS_EVENT_SETUP_OK / X	<i>If the set up of the Cid X with <code>adl_gprsSetup()</code> did succeed</i>
ADL_GPRS_EVENT_SETUP_KO / X	<i>If the set up of the Cid X with <code>adl_gprsSetup()</code> did fail</i>
ADL_GPRS_EVENT_ME_ATTACH	<i>If the ME has forced a network attachment</i>
ADL_GPRS_EVENT_ME_UNREG	<i>If the ME is not registered</i>
ADL_GPRS_EVENT_ME_UNREG_SEARCHING	<i>If the ME is not registered but is searching a new operator to register to.</i>

Note: If Cid X is not defined, the value ADL_CID_NOT_EXIST will be used as X.

The possible return values for this handler are defined below:

Event	Description
ADL_GPRS_FORWARD	<i>the event shall be sent to the external application. On unsolicited events, these will be forwarded to all opened ports. On responses events, these will be forwarded only on the port on which the request was executed.</i>
ADL_GPRS_NO_FORWARD	<i>the event shall not be sent to the external application</i>
ADL_GPRS_NO_FORWARD_ATH	<i>the event shall not be sent to the external application and the application shall terminate the incoming activation request by sending an 'ATH' command.</i>
ADL_GPRS_NO_FORWARD_ATA	<i>the event shall not be sent to the external application and the application shall accept the incoming activation request by sending an 'ATA' command.</i>

- **Returned values for `adl_gprsSubscribe`**

This function returns OK on success, or a negative error value.

Possible error values are:

Error value	Description
ADL_RET_ERR_PARAM	<i>In case of parameter error</i>

3.13.3 The `adl_gprsSetup` function

This function just runs the `adl_gprsSetupExt` one on the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` port (cf. `adl_gprsSetupExt` description for more information). Please note that events generated by the `adl_gprsSetup` will not be able to be forwarded to any external port, since the setup command was running on the Open AT® port.

3.13.4 The `adl_gprsSetupExt` function

This function sets up a PDP context identified by its CID with some specific parameters.

- **Prototype**

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

- **Parameters**

Cid:

The Cid of the PDP context to set up (integer value between 1 and 4).

Params:

Structure containing the parameters to set up using the following type:

```
typedef struct
{
    ascii* APN;      ascii* Login;
    ascii* Password;
    ascii* FixedIP;
    bool   HeaderCompression;
    bool   DataCompression;
} adl_gprsSetupParams_t;
```

- APN: Address of the Provider GPRS Gateway (GGSN) maximum 100 bytes string
- Login: GPRS account login maximum 50 bytes string

ADL User Guide for Open AT® OS v3.13

API

- Password:
GPRS account password
maximum 50 bytes string
- FixedIP:
Optional fixed IP address of the MS (used only if not set to NULL)
maximum 15 bytes string
- HeaderCompression:
PDP header compression option (enabled if set to TRUE)
- DataCompression:
PDP data compression option (enabled if set to TRUE)

Port:

Port on which to run the PDP context setup command. When setup return events are received in the GPRS event handler, if the application requires ADL to forward these events, they will be forwarded to this Port parameter value.

- **Returned values**

This function returns OK on success, or a negative error value.
Possible error values are:

Error value	Description
ADL_RET_ERR_PARAM	<i>In case of parameter error: bad Cid value or unavailable port.</i>
ADL_RET_ERR_PIN_KO	<i>If the PIN is not entered, or if the "+WIND:4" indication has not occurred yet.</i>
ADL_GPRS_CID_NOT_DEFINED	<i>in case of problem to set up the Cid (the CID is already activated)</i>
ADL_NO_GPRS_SERVICE	<i>f the GPRS service is not supported by the product.</i>
ADL_RET_ERR_BAD_STATE	<i>The service is still processing another GPRS API; application should wait for the corresponding event (indication of end of processing) in the GPRS handler before calling this function.</i>

3.13.5 The `adl_gprsAct` function

This function just runs the `adl_gprsActExt` one on the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` port (cf. `adl_gprsActExt` description for more information). Please note that events generated by the `adl_gprsAct` will not be able to be forwarded to any external port, since the setup command was running on the Open AT® port.

3.13.6 The `adl_gprsActExt` function

This function activates a specific PDP context identified by its Cid.

- **Prototype**

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

- **Parameters**

Cid:

The Cid of the PDP context to activate (integer value between 1 and 4).

Port:

Port on which to run the PDP context activation command. When activation return events are received in the GPRS event handler, if the application requires ADL to forward these events, they will be forwarded to this Port parameter value.

- **Returned values**

This function returns OK on success, or a negative error value.

Possible error values are:

Error value	Description
ADL_RET_ERR_PARAM	<i>in case of parameters error: bad Cid value or unavailable port</i>
ADL_RET_ERR_PIN_KO	<i>If the PIN is not entered, or if the "+WIND:4" indication has not occurred yet.</i>
ADL_GPRS_CID_NOT_DEFINED	<i>in case of problem to set up the Cid (the CID is already activated)</i>
ADL_NO_GPRS_SERVICE	<i>f the GPRS service is not supported by the product.</i>
ADL_RET_ERR_BAD_STATE	<i>The service is still processing another GPRS API; application should wait for the corresponding event (indication of end of processing) in the GPRS handler before calling this function.</i>

Important Note: This function must be called before opening the GPRS FCM Flows.

3.13.7 The `adl_gprsDeact` function

This function just runs the `adl_gprsDeactExt` one on the `ADL_PORT_OPEN_AT_VIRTUAL_BASE` port (cf. `adl_gprsDeactExt` description for more information). Please note that events generated by the `adl_gprsDeact` will not be able to be forwarded to any external port, since the setup command was running on the Open AT® port.

3.13.8 The `adl_gprsDeactExt` function

This function deactivates a specific PDP context identified by its Cid.

- Prototype**

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

- Parameters**

Cid:

The Cid of the PDP context to deactivate (integer value between 1 and 4).

Port:

Port on which to run the PDP context deactivation command. When deactivation return events are received in the GPRS event handler, if the application requires ADL to forward these events, they will be forwarded to this Port parameter value.

- Returned values**

This function returns OK on success, or a negative error value.

Possible error values are:

Error value	Description
ADL_RET_ERR_PARAM	<i>in case of parameters error: bad Cid value or unavailable port.</i>
ADL_RET_ERR_PIN_KO	<i>If the PIN is not entered, or if the "+WIND:4" indication has not occurred yet.</i>
ADL_GPRS_CID_NOT_DEFINED	<i>in case of problem to set up the Cid (the CID is already activated)</i>
ADL_NO_GPRS_SERVICE	<i>f the GPRS service is not supported by the product.</i>
ADL_RET_ERR_BAD_STATE	<i>The service is still processing another GPRS API; application should wait for the corresponding event (indication of end of processing) in the GPRS handler before calling this function.</i>

IMPORTANT NOTE: if the GPRS flow is running, please do wait for the ADL_FCM_EVENT_FLOW_CLOSED event before calling the adl_gprsDeact function, in order to prevent module lock.

3.13.9 The adl_gprsGetCidInformations function

This function gets information about a specific activated PDP context identified by its Cid.

- **Prototype**

```
s8 adl_gprsGetCidInformations ( u8          Cid,  
                               adl_gprsInfosCid_t * Infos );
```

- **Parameters**

Cid:

The Cid of the PDP context (integer value between 1 and 4).

Infos:

Structure containing the information of the activated PDP context using the following type:

```
typedef struct  
{  
    u32 LocalIP; // Local IP address of the MS  
    u32 DNS1;    // First DNS IP address  
    u32 DNS2;    // Second DNS IP address  
    u32 Gateway; // Gateway IP address  
}adl_gprsInfosCid_t;
```

This parameter fields will be set only if the GPRS session is activated; otherwise, they all will be set to 0.

- **Returned values**

This function returns OK on success, or a negative error value.
Possible error values are:

Error value	Description
ADL_RET_ERR_PARAM	<i>in case of parameters error: bad Cid value</i>
ADL_RET_ERR_PIN_KO	<i>If the PIN is not entered, or if the "+WIND:4" indication has not occurred yet.</i>
ADL_GPRS_CID_NOT_DEFINED	<i>in case of problem to set up the Cid (the CID is already activated)</i>
ADL_NO_GPRS_SERVICE	<i>f the GPRS service is not supported by the product.</i>
ADL_RET_ERR_BAD_STATE	<i>The service is still processing another GPRS API; application should wait for the corresponding event (indication of end of processing) in the GPRS handler before calling this function.</i>

3.13.10 The `adl_gprsUnsubscribe` function

This function unsubscribes from the GPRS service. The handler provided will not receive GPRS events any more.

- **Prototype**

```
s8 adl_gprsUnsubscribe ( adl_gprsHdlr_f Handler );
```

- **Parameters**

Handler:

Handler used with `adl_gprsSubscribe` function.

- **Returned values**

This function returns OK on success, or a negative error value.
Possible error values are:

Error value	Description
ADL_RET_ERR_PARAM	<i>in case of parameters error</i>
ADL_RET_ERR_UNKNOWN_HDL	<i>if the provided handler is unknown</i>
ADL_RET_ERR_NOT_SUBSCRIBED	<i>if the service is not subscribed</i>

3.13.11 The `adl_gprsIsAnIPAddress` function

This function checks if the provided string is a valid IP address. Valid IP address strings are those based on the "a.b.c.d" format, where a, b, c & d are integer values between 0 and 255.

- **Prototype**

```
bool adl_gprsIsAnIPAddress ( ascii * AddressStr );
```

- **Parameters**

AddressStr:

IP address string to check.

- **Returned values**

This function returns TRUE if the string provided is a valid IP address string, and FALSE otherwise.

NULL & empty string ("") are not considered as a valid IP address.

3.13.12 Example

This example simply demonstrates how to use the GPRS service in a nominal case (error cases are not handled).

Full examples using the GPRS service are also available on the SDK (Ping_GPRS sample).

```
// Global variables & constants
adl_gprsSetupParams_t MyGprsSetup;
adl_gprsInfosCid_t   InfosCid;

// GPRS event Handler
s8 MyGprsEventHandler ( u16 Event, u8 CID )
{

    // Trace event
    TRACE (( 1, "Received GPRS event %d/%d", Event, CID ));

    // Switch on event
    switch ( Event )
    {
        case ADL_GPRS_EVENT_SETUP_OK :
            TRACE (( 1, "PDP Ctxt Cid %d Setup OK", CID ));
            // Activate the session
            adl_gprsAct ( 1 );
            break;

        case ADL_GPRS_EVENT_ACTIVATE_OK :
            TRACE (( 1, "PDP Ctxt %d Activation OK", CID ));
            // Get context information
            adl_gprsGetCidInformations ( 1, &InfosCid );
            // De-activate the session
            adl_gprsDeAct ( 1 );
        }
        break;

        case ADL_GPRS_EVENT_DEACTIVATE_OK :
            TRACE (( 1, " PDP Ctxt %d De-activation OK", CID ));
            // Un-subscribe from GPRS event handler
            adl_gprsUnsubscribe ( MyGprsEventHandler );
            break;
    }

    // Forward event
    return ADL_GPRS_FORWARD;
}
```

ADL User Guide for Open AT® OS v3.13

API

```
// Somewhere in the application code, used as an event handler
void MyFunction ( void )

    // Subscribe to an A&D cell
    MyGprsSetup.APN = "myapn";
    MyGprsSetup.Login = "login";
    MyGprsSetup.Password = "password";
    MyGprsSetup.FixedIP = NULL;
    MyGprsSetup.HeaderCompression = FALSE;
    MyGprsSetup.DataCompression = FALSE;

    // Subscribe to GPRS event handler
    adl_gprsSubscribe ( MyGprsEventHandler );

    // Set up the GPRS context
    adl_gprsSetup ( 1, MyGprsSetup );
}
```

3.14 Application Safe Mode Service

By default, the +WOPEN and +WDWL commands cannot be filtered by any embedded application. This service allows one application to get these command events, in order to prevent any external application stopping or erasing the current embedded one.

3.14.1 Required Header File

The header file for the Application safe mode service is:
adl_safe.h

3.14.2 The adl_safeSubscribe function

This function subscribes to the Application safe mode service in order to receive +WOPEN and +WDWL command events.

- **Prototype**

```
s8      adl_safeSubscribe ( u16      WDWLopt,  
                           u16      WOPENopt,  
                           adl_safeHdlr_f SafeHandler );
```

- **Parameters**

WDWLopt:

Additional options for +WDWL command subscription. This command is at least subscribed in ACTION and READ mode. Please see adl_atCmdSubscribe API for more details on these options.

WOPENopt:

Additional options for +WOPEN command subscription. This command is at least subscribed in READ, TEST and PARAM mode, with at least one mandatory parameter. Please see adl_atCmdSubscribe API for more details on these options.

SafeHandler:

Application safe mode handler defined using the following type:

```
typedef bool (*adl_safeHdlr_f) ( adl_safeCmdType_e CmdType,  
                                adl_atCmdPreParser_t * paras );
```

The CmdType events received by this handler are defined below:

```
typedef enum
{
    ADL_SAFE_CMD_WDWL,                // AT+WDWL command
    ADL_SAFE_CMD_WDWL_READ,          // AT+WDWL? command
    ADL_SAFE_CMD_WDWL_OTHER,         // WDWL other syntax

    ADL_SAFE_CMD_WOPEN_STOP,         // AT+WOPEN=0 command
    ADL_SAFE_CMD_WOPEN_START,        // AT+WOPEN=1 command
    ADL_SAFE_CMD_WOPEN_GET_VERSION,  // AT+WOPEN=2 command
    ADL_SAFE_CMD_WOPEN_ERASE_OBJ,    // AT+WOPEN=3 command
    ADL_SAFE_CMD_WOPEN_ERASE_APP,    // AT+WOPEN=4 command
    ADL_SAFE_CMD_WOPEN_SUSPEND_APP,  // AT+WOPEN=5 command
    ADL_SAFE_CMD_WOPEN_AD_GET_SIZE,  // AT+WOPEN=6 command
    ADL_SAFE_CMD_WOPEN_AD_SET_SIZE,  // AT+WOPEN=6,<size> command
    ADL_SAFE_CMD_WOPEN_READ,         // AT+WOPEN? command
    ADL_SAFE_CMD_WOPEN_TEST,         // AT+WOPEN=? command
    ADL_SAFE_CMD_WOPEN_OTHER         // WOPEN other syntax
} adl_safeCmdType_e;
```

The `paras` received structure contains the same parameters as if the commands were subscribed with `adl_atCmdSubscribe` API.

If the Handler returns `FALSE`, the command will not be forwarded to the Wavecom core software.

If the Handler returns `TRUE`, the command will be processed by the Wavecom core software, which will send responses to the external application.

- **Returned values**

- OK on success.
- `ADL_RET_ERR_PARAM` if the parameters have an incorrect value
- `ADL_RET_ERR_ALREADY_SUBSCRIBED` if the service is already subscribed

3.14.3 The `adl_safeUnsubscribe` function

This function unsubscribes from Application safe mode service. The +WDWL and +WOPEN commands are no longer filtered and always processed by the Wavecom core software.

- **Prototype**

```
s8 adl_safeUnsubscribe ( adl_safeHdlr_f Handler);
```

- **Parameters**

Handler:

Handler used with `adl_safeSubscribe` function.

- **Returned values**

- OK on success.
- `ADL_RET_ERR_PARAM` if the parameter has an incorrect value
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided handler is unknown
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the service is not subscribed

3.14.4 The `adl_safeRunCommand` function

This function allows to run +WDWL or +WOPEN command with any standard syntax, and to get its answers.

- **Prototype**

```
s8      adl_safeRunCommand ( adl_safeCmdType_e CmdType,  
                             adl_atRspHandler_t RspHandler );
```

- **Parameters**

CmdType:

Command type to run; please refer to `adl_safeSubscribe` description. `ADL_SAFE_CMD_WDWL_OTHER` and `ADL_SAFE_CMD_WOPEN_OTHER` values are not allowed.

The `ADL_SAFE_CMD_WOPEN_SUSPEND_APP` may be used to suspend the Open AT® application task. The execution may be resumed using the `AT+WOPENRES` command, or by sending a signal on the hardware Interrupt product pin (The `INTERRUPT` feature has to be enabled on the product: please refer to the `AT+WFM` command). Open AT® application running in Remote Task Environment cannot be suspended (the function has no effect). Please note that the current Open AT® application process is suspended immediately on the `adl_safeRunCommand` process; if there is any code after this function call, it will be executed only once the process is resumed.

RspHandler:

Response handler to get ran commands' results. All responses are subscribed; the command will be executed on the Open AT® virtual port. Instead of providing a response handler, a port identifier may be specified (using `adl_port_e` type): the command will be executed on this port, and the resulting responses sent back on this port too.

- **Returned values**

- OK on success.
- `ADL_RET_ERR_PARAM` if the parameter has an incorrect value

3.15 AT Strings Service

This service provides APIs to process AT standard response strings.

3.15.1 Required Header File

The header file for the AT strings service is:

adl_str.h

3.15.2 The adl_strID_e type

This type defines all pre-defined AT strings by this service, defined below:

```
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;
```

3.15.3 The `adl_strGetID` function

This function returns the ID of the response string provided.

- **Prototype**

```
adl_strID_e adl_strGetID ( ascii * rsp );
```

- **Parameters**

rsp:

String to parse to get the ID.

- **Returned values**

- ADL_STR_NO_STRING if the string is unknown.
- Id of the string otherwise.

3.15.4 The `adl_strGetIDExt` function

This function returns the ID of the response string provided, with an optional argument and its type.

- **Prototype**

```
adl_strID_e adl_strGetIDExt (  ascii * rsp
                               void *  arg
                               u8 *   argtype );
```

- **Parameters**

rsp:

String to parse to get the ID.

arg:

Parsed first argument; not used if set to NULL.

argtype:

Type of the parsed argument:

if argtype is ADL_STR_ARG_TYPE_ASCII, arg is an `ascii * string`;

if argtype is ADL_STR_ARG_TYPE_U32, arg is an `u32 * integer`.

- **Returned values**

- ADL_STR_NO_STRING if the string is unknown.
- Id of the string otherwise.

3.15.5 The `adl_strIsTerminalResponse` function

This function checks whether the response ID provided is a terminal response. A terminal response is the last response that a response handler will receive from a sent command.

- **Prototype**

```
bool adl_strIsTerminalResponse ( adl_strID_e RspID );
```

- **Parameters**

RspID:

Response ID to check.

- **Returned values**

- TRUE if the provided response ID is a terminal one.
- FALSE otherwise.

3.15.6 The `adl_strGetResponse` function

This function provides the standard response string from its ID.

- **Prototype**

```
ascii *adl_strGetResponse ( adl_strID_e RspID );
```

- **Parameters**

RspID:

Response ID from which to get the string.

- **Returned values**

- Standard response string on success;
- NULL if the ID does not exist.

IMPORTANT WARNING:

The returned pointer memory is allocated by this function, but its ownership is transferred to the embedded application; i.e. the embedded application will have to release the returned pointer.

3.15.7 The `adl_strGetResponseExt` function

This function provides a standard response string from its ID, with the argument provided.

- **Prototype**

```
ascii * adl_strGetResponseExt ( adl_strID_e RspID,  
                               u32          arg );
```

- **Parameters**

RspID:

Response ID from which to get the string.

arg:

Response argument to copy in the response string; according to response ID, this argument should be an `u32` integer value, or an `ascii * string`.

- **Returned values**

Standard response string on success;
NULL if the ID does not exist.

IMPORTANT WARNING:

The returned pointer memory is allocated by this function, but its ownership is transferred to the embedded application; i.e. the embedded application will have to release the returned pointer.

3.16 Application & Data storage Service

This service provides APIs to use the Application & Data storage volume. This volume may be used to store data, or ".dwl" files (Wavecom OS updates, new Open AT® applications or E2P configuration files) in order to be later installed on the product.

The default storage size is 768 Kbytes; it may be configured with the AT+WOPEN command (Please refer to the AT commands interface guide for more information) .

This storage size has to be set to the maximum (about 1.2 Mbytes) in order to have enough place to store a Wavecom OS update.

Warning: any A&D size change will result in formatting of this area (some seconds after start-up; all A&D cells data will be erased).

Legal mention:

The Download Over The Air feature enables the Wavecom firmware to be remotely updated.

The downloading and firmware updating processes have to be activated and managed by an appropriate Open AT® based application to be developed by the customer. The security of the whole process (request for update, authentication, encryption, etc) has to be managed by the customer under his own responsibility. Wavecom shall not be liable for any issue related to any use by customer of the Download Over The Air feature.

WAVECOM AGREES AND THE CUSTOMER ACKNOWLEDGES THAT THE SDK Open AT® IS PROVIDED "AS IS" BY WAVECOM WITHOUT ANY WARRANTY OR GUARANTEE OF ANY KIND.

3.16.1 Required Header File

The header file for the Application & Data storage service is:

```
adl_ad.h
```

3.16.2 The adl_adSubscribe function

This function subscribes to the required A&D space cell identifier.

- **Prototype**

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

- **Parameters**

CellID:

A&D space cell identifier to subscribe to; this cell may already exist or not. If the cell does not exist, the given size is allocated.

Size:

New cell size in bytes (this parameter is ignored if the cell already exists). It may be set to **ADL_AD_SIZE_UNDEF** for a variable size. In this case, new cells subscription will fail until the undefined size cell is finalised.

Total used size in flash will be data size + header size ; header size is variable (with an average value of 16 bytes).

When subscribing, the size is rounded to the next multiple of 4 .

• **Returned values**

- A positive or null value on success:
 - The A&D cell handle on success, to be used on further A&D API functions calls,
- A negative error value:
 - **ADL_RET_ERR_ALREADY_SUBSCRIBED** if the cell is already subscribed;
 - **ADL_AD_RET_ERR_OVERFLOW** if there is not enough allocated space,
 - **ADL_AD_RET_ERR_NOT_AVAILABLE** if there is no A&D space available on the product,
 - **ADL_RET_ERR_PARAM** if the CellId parameter is 0xFFFFFFFF (this value should not be used as an A&D Cell ID),
 - **ADL_RET_ERR_BAD_STATE** (when subscribing an undefined size cell) if another undefined size cell is already subscribed and not finalized.

3.16.3 The adl_adUnsubscribe function

This function unsubscribes from the given A&D cell handle.

• **Prototype**

```
s32 adl_adUnsubscribe ( s32 CellHandle );
```

• **Parameters**

CellHandle:

A&D cell handle returned by adl_adSubscribe function.

• **Returned values**

- OK on success;
- **ADL_RET_ERR_UNKNOWN_HDL** if the handle was not subscribed.

3.16.4 The `adl_adEventSubscribe` function

This function allows the application to provide ADL with an event handler to be notified with A&D service related events.

- **Prototype**

```
s32 adl_adEventSubscribe ( adl_adEventHdlr_f Handler );
```

- **Parameters**

Handler:

Call-back function provided by the application. Please refer to next chapter for more information.

- **Returned values**

- A positive or null value on success:
 - A&D event handle, to be used in further A&D API functions calls,
- A negative error value:
 - `ADL_RET_ERR_PARAM` if the `Handler` parameter is invalid,
 - `ADL_RET_ERR_NO_MORE_HANDLES` if the A&D event service has been subscribed more than 128 times.

- **Notes**

In order to format or re-compact the A&D storage volume, the `adl_adEventSubscribe` function has to be called before the `adl_adFormat` or the `adl_adRecompact` functions.

3.16.5 The `adl_adEventHdlr_f` call-back type

This call-back function has to be provided to ADL through the `adl_adEventSubscribe` interface, in order to receive A&D related events.

- **Prototype**

```
typedef void (*adl_adEventHdlr_f) ( adl_adEvent_e Event,  
u32 Progress );
```

- **Parameters**

Event:

Event is the received event identifier. The events (defined in the `adl_adEvent_e` type) are described in the table below.

ADL User Guide for Open AT® OS v3.13

API

Event	Meaning
ADL_AD_EVENT_FORMAT_INIT	The <code>ad1_adFormat</code> function has been called by an application (a format process has just been requested).
ADL_AD_EVENT_FORMAT_PROGRESS	The format process is on going. Several "progress" events should be received until the process is completed.
ADL_AD_EVENT_FORMAT_DONE	The format process is over. The A&D storage area is now usable again. All cells have been erased, and the whole storage place is available.
ADL_AD_EVENT_RECOMPACT_INIT	The <code>ad1_adRecompact</code> function has been called by an application (a re-compaction process has been requested).
ADL_AD_EVENT_RECOMPACT_PROGRESS	The re-compaction process is on going. Several "progress" events should be received until the process is complete.
ADL_AD_EVENT_RECOMPACT_DONE	The re-compaction process is over: the A&D storage area is now usable again. The space previously used by deleted cells is now free.
ADL_AD_EVENT_INSTALL	The <code>ad1_adInstall</code> function has been called by an application (an install process has just been required and the wireless CPU® is going to reset).

Progress:

On `ADL_AD_EVENT_FORMAT_PROGRESS` & `ADL_AD_EVENT_RECOMPACT_PROGRESS` events reception, this parameter is the process progress ratio (considered as a percentage).

On `ADL_AD_EVENT_FORMAT_DONE` & `ADL_AD_EVENT_RECOMPACT_DONE` events reception, this parameter is set to 100%.

Otherwise, this parameter is set to 0.

3.16.6 The `adl_adEventUnsubscribe` function

This function allows the Open AT® application to unsubscribe from the A&D events notification.

- **Prototype**

```
s32 adl_adEventUnsubscribe ( s32 EventHandle );
```

- **Parameters**

EventHandle:

Handle previously returned by the `adl_adEventSubscribe` function.

- **Returned values**

- OK on success,
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle is unknown,
- `ADL_RET_ERR_NOT_SUBSCRIBED` if no A&D event handler has been subscribed,
- `ADL_RET_ERR_BAD_STATE` if a format or re-compaction process is currently running with this event handle.

3.16.7 The `adl_adWrite` function

This function writes data at the end of the given A&D cell.

- **Prototype**

```
s32 adl_adWrite ( s32 CellHandle  
                 u32 Size  
                 void * Data );
```

- **Parameters**

CellHandle:

A&D cell handle returned by `adl_adSubscribe` function.

Size:

Data buffer size in bytes.

Data:

Data buffer.

- **Returned values**

- OK on success,
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle was not subscribed,
- `ADL_RET_ERR_PARAM` on parameter error,
- `ADL_RET_ERR_BAD_STATE` if the cell is finalized,
- `ADL_AD_RET_ERR_OVERFLOW` if the write operation exceeds the cell size.

3.16.8 The `adl_adInfo` function

This function provides information on the requested A&D cell.

- **Prototype**

```
s32 adl_adInfo ( s32 CellHandle  
                adl_adInfo_t * Info );
```

- **Parameters**

CellHandle:

A&D cell handle returned by the `adl_adSubscribe` function.

Info:

Information structure on requested cell, based on following type:

```
typedef struct  
{  
    u32 identifier; // identifier  
    u32 size; // entry size  
    void *data; // pointer to stored data  
    u32 remaining; // remaining writable space unless finalized  
    bool finalised; // TRUE if entry is finalized  
}adl_adInfo_t;
```

- **Returned values**

- OK on success ;
- `ADL_RET_ERR_PARAM` on parameter error ;
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle was not subscribed.
- `ADL_RET_ERR_BAD_STATE` if the required cell is a not finalized undefined size cell.

3.16.9 The `adl_adFinalise` function

This function sets the provided A&D cell in read-only (finalized) mode. The cell content can no longer be modified.

- **Prototype**

```
s32 adl_adFinalise ( s32 CellHandle );
```

- **Parameters**

CellHandle:

A&D cell handle returned by the `adl_adSubscribe` function.

- **Returned values**

- OK on success;
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle was not subscribed;
- `ADL_RET_ERR_BAD_STATE` if the cell was already finalized.

3.16.10 The `adl_adDelete` function

This function deletes the A&D provided cell. The used space and the ID will be available on the next re-compaction process.

- **Prototype**

```
s32 adl_adDelete ( s32 CellHandle );
```

- **Parameters**

CellHandle:

A&D cell handle returned by the `adl_adSubscribe` function.

- **Returned values**

- OK on success;
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle was not subscribed.

Note: calling `adl_adDelete` will unsubscribe the allocated handle.

3.16.11 The `adl_adInstall` function

This function installs the content of the requested cell, if it is a `.DWL` file. This file should be an Open AT® application, an EEPROM configuration file, an XModem downloader binary file, or a Wavecom OS binary file.

WARNING: This API resets the product on success.

- **Prototype**

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

- **Parameters**

Handle:

A&D cell handle returned by the `adl_adSubscribe` function.

- **Returned values**

- **Product resets on success.** The parameter of the `adl_main` function is then set to `ADL_INIT_DOWNLOAD_SUCCESS`, or `ADL_INIT_DOWNLOAD_ERROR`, according to `.DWL` file update success or not. Before the product reset, all subscribed event handlers (if any) will receive the `ADL_AD_EVENT_INSTALL` event, in order to let them perform the last operations.
- `ADL_INIT_DOWNLOAD_ERROR`, according to the `.DWL` file update success or not.
- `ADL_RET_ERR_BAD_STATE` if the cell is not finalized;
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle was not subscribed.

3.16.12 The `adl_adRecompact` function

This function starts the re-compaction process, which will release the deleted cells spaces and IDs.

Warning: if some A&D cells are deleted, if this recompaction process is not performed regularly, these deleted cells used space will not be freed.

- **Prototype**

```
s32 adl_adRecompact ( s32 EventHandle );
```

- **Parameters**

EventHandle:

Event handle previously returned by the `adl_adEventSubscribe` function. The associated handler will receive the re-compaction process events sequence.

- **Returned values**

- OK on success. Event handlers will receive the following event sequence:
 - `ADL_AD_EVENT_RECOMPACT_INIT` just after the process is launched,
 - `ADL_AD_EVENT_RECOMPACT_PROGRESS` several times, indicating process progression,
 - `ADL_AD_EVENT_RECOMPACT_DONE` when the process is complete.
- `ADL_RET_ERR_BAD_STATE` if a re-compaction or format process is currently running,
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle is unknown,
- `ADL_RET_ERR_NOT_SUBSCRIBED` if no A&D event handler has been subscribed,
- `ADL_AD_RET_ERR_NOT_AVAILABLE` if there is no A&D space available on the product.

3.16.13 The `adl_adGetState` function

This function provides an information structure on the current A&D volume state.

- **Prototype**

```
s32 adl_adGetState ( adl_adState_t * State );
```

- **Parameters**

State:

A&D volume information structure, based on following type:

```
typedef struct
{
    u32 freemem;           // Space free memory size
    u32 deletedmem;       // Deleted memory size
    u32 totalmem;         // Total memory
    u16 numobjects;       // Number of allocated objects
    u16 numdeleted;       // Number of deleted objects
    u8  pad;              // not used
} adl_adState_t;
```

- **Returned values**

- OK on success;
- `ADL_AD_RET_ERR_NOT_AVAILABLE` if there is no A&D space available on the product;
- `ADL_AD_RET_ERR_NEED_RECOMPACT` if a power down or a reset occurred when a re-compaction process was running: the application has to launch the `adl_adRecompact` function before using any other A&D service function;
- `ADL_RET_ERR_PARAM` on parameter error.

3.16.14 The `adl_adGetCellList` function

This function provides the list of the current allocated cells.

- **Prototype**

```
s32 adl_adGetCellList ( wm_lst_t * CellList );
```

- **Parameters**

CellList:

Return allocated cell list. The list elements are the cell identifiers and are based on u32 type.

The list is ordered by cell id values, from the lowest to the greatest.

WARNING: the list used memory is allocated by the `adl_adGetCellList` function and **has to be released by the application**.

- **Returned values**

- OK on success;
- `ADL_AD_RET_ERR_NOT_AVAILABLE` if there is no A&D space available on the product;
- `ADL_RET_ERR_PARAM` on parameter error.

Note :

- The number of elements in the returned list is limited by `ADL_AD_MAX_CELL_RETRIEVE`.
- If the number of cell IDs to get is superior to `ADL_AD_MAX_CELL_RETRIEVE`, use `adl_adFindInit()` and `adl_adFindNext()` functions (please refer to sections 3.16.16 and 3.16.17).

3.16.15 The `adl_adFormat` function

This function allows the A&D storage volume to be completely re-initialized. It is allowed only if there are currently no subscribed cells, or if there is no currently running re-compaction or format process.

Important warning:

All the A&D storage cells will be erased by this operation. The A&D storage format process may take up to several seconds.

- **Prototype**

```
s32 adl_adFormat (s32 EventHandle );
```

- **Parameters**

EventHandle:

Event handle previously returned by the `adl_adEventSubscribe` function. The associated handler will receive the format process events sequence.

• **Returned values**

- OK on success. Event handlers will receive the following event sequence:
 - ADL_AD_EVENT_FORMAT_INIT just after the process is launched,
 - ADL_AD_EVENT_FORMAT_PROGRESS several times, indicating process progression,
 - ADL_AD_EVENT_FORMAT_DONE once the process is complete,
- ADL_RET_ERR_UNKNOWN_HDL if the handle is unknown,
- ADL_RET_ERR_NOT_SUBSCRIBED if no A&D event handler has been subscribed,
- ADL_AD_RET_ERR_NOT_AVAILABLE if there is no A&D space available on the product,
- ADL_RET_ERR_BAD_STATE if there is at least one currently subscribed cell, or if a re-compaction or format process is already running.

3.16.16 The adl_adFindInit function

• **Prototype :**

```
s32 adl_adFindInit ( u32          MinCellId,  
                   u32          MaxCellId,  
                   adl_adBrowse_t * BrowseInfo );
```

• **Parameters :**

MinCellId :

Minimum cell value for wanted cell identifiers

MaxCellId :

Maximum cell value for wanted cell identifiers

BrowseInfo :

Returned browse information, to be used with the `adl_adFindNext()` function (see section 3.16.17).

This parameter is based on following type:

```
typedef struct  
{  
    u32  hidden[4]; // memory space necessary for cell information  
}adl_adBrowse_t;
```

• **Returned values :**

- OK on success
- ADL_AD_RET_ERR_NOT_AVAILABLE if A&D space is not available
- ADL_RET_ERR_PARAM on parameter error

3.16.17 The `adl_adFindNext` function

This function performs a cell ID search on the browse informations provided by the `adl_adFindInit()` function.

- **Prototype**

```
s32 adl_adFindNext (  adl_adBrowse_t *  BrowseInfo,  
                    u32 *              CellID );
```

- **Parameters**

BrowseInfo:

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

CellID:

Next found cell ID.

- **Returned values**

- **OK on success**
- **ADL_RET_ERR_PARAM** on parameter error
- **ADL_AD_RET_REACHED_END** no more elements to enumerate

3.16.18 Example

This example demonstrates how to use the A&D service in a nominal case (error cases not handled).

Complete examples using the A&D service are also available on the SDK (DTL Application_Download sample, generic Download library sample).

```
// Global variables & constants

// Cell & event handles
s32 MyADCellHandle;
s32 MyADEventHandle;

// Info & state structure
adl_adInfo_t Info;
adl_adState_t State;

// A&D event handler
void MyADEventHandler ( adl_adEvent_e Event, u32 Progress )
{
    // Check event
    switch ( Event )
    {
        case ADL_AD_EVENT_RECOMPACT_DONE :
        case ADL_AD_EVENT_FORMAT_DONE :
            // The process is over
            TRACE (( 1, "Format/Recompact process over..." ));
            break;
    }
}

...

// Somewhere in the application code, used as an event handler
void MyFunction ( void )
{
    // Local variables
    u8 DataBuffer [ 10 ];

    // Get state
    adl_adGetState ( &State );

    // Subscribe to the A&D event service
    MyADEventHandle = adl_adEventSubscribe ( MyADEventHandler );

    // Subscribe to an A&D cell
    MyADCellHandle = adl_adSubscribe ( 0x00000000, 20 );
}
```

```
// Write data buffer
wm_memset ( DataBuffer, 10, 0 );
adl_adWrite ( MyADCellHandle, 10, DataBuffer );

// Get info
adl_adInfo ( MyADCellHandle, &Info );

// Install the cell (will fail, not finalized)
adl_adInstall ( MyADCellHandle );

// Finalize the cell
adl_adFinalise ( MyADCellHandle );

// Delete the cell
adl_adDelete ( MyADCellHandle );

// Launch the re-compaction process
adl_adRecompact ( MyADEventHandle );

// Launch the format process
// (will fail, re-compaction process is running)

adl_adFormat ( MyADEventHandle );

// Unsubscribe from the A&D event service
// (will fail, re-compaction process is running)
adl_adEventUnsubscribe ( MyADEventHandler );
}
```

3.17 GPS Service

ADL applications may use this service to access the GPS device information on Q2501 products.

Note: the product uses the module's second UART to access the GPS component. This will lock some GPIOs, which will not be available for allocation by the application; please refer to §2.5 for more information.

3.17.1 Required Header File

The header file for the GPS service is:

```
adl_gps.h
```

3.17.2 GPS Data structures

3.17.2.1 Position

GPS Position data is stored in the following structure:

```
typedef struct
{
    ascii UTC_time [_S_UTC_TIME];           // hhmmss.sss
    ascii date [_S_DATE];                   // ddmmyy
    ascii latitude [_S_POSITION];           // ddmm.mmmmm
    ascii latitude_Indicator[_S_INDICATOR]; // N - S
    ascii longitude [_S_POSITION];          // dddmm.mmmmm
    ascii longitude_Indicator[_S_INDICATOR]; // E - W
    ascii status[_S_INDICATOR];
    ascii P_Fix[_S_INDICATOR];
    ascii sat_used [_S_SAT];                 // Satellites used
    ascii HDOP [_S_HDOP];                   // Horizontal Dilution of
                                           // Precision
    ascii altitude [_S_ALTITUDE];           // MSL Altitude
    ascii altitude_Unit[_S_INDICATOR];
    ascii geoid_Sep [_S_GEOID_SEP];         // geoid correction
    ascii geoid_Sep_Unit[_S_INDICATOR];
    ascii Age_Dif_Cor [_S_AGE_DIF_COR];     // Age of Differential
                                           // correction
    ascii Dif_Ref_ID [_S_DIF_REF_ID];       // Diff Ref station ID
    ascii magneticVariation[_S_COURSE];     // magnetic variation: not
                                           // available for sirf
                                           // technology
} adl_gpsPosition_t;
```

All fields are ascii zero terminated strings containing GPS information.

3.17.2.2 Speed

GPS Speed data is stored in the following structure:

```
typedef struct
{
    ascii course [_S_COURSE];           // Degrees from true North
    ascii speed_knots [_S_SPEED];       // Speed in knots
    ascii speed_km_p_hour [_S_SPEED];  // Speed in km/h
} adl_gpsSpeed_t;
```

All fields are ascii zero terminated strings containing GPS information.

3.17.2.3 Satellite View

GPS satellite view data is stored in the following structure:

```
typedef struct
{
    u8 id;           // range 1 to 32
    u8 elevation;   // maximum 90
    u32 azimuth;    // range 0 to 359
    s8 SNR ;        // range 0 to 99, -1 when not tracking
} adl_gpsSatellite_t;
```

All fields are integers containing GPS information about the current satellite.

```
typedef struct
{
    u8 NB_Msg ;           // Number of messages
    u8 MSG_Number ;      // Message Number
    u8 Sat_view ;        // Satellites in view
    adl_gpsSatellite_t sat [_NB_SAT_MAX]; // array for informations about
                                           // differents satellites
} adl_gpsSatView_t;
```

The different fields contain information about the current satellite view. Each satellite's information details are contained in the "sat" field.

3.17.3 The `adl_gpsSubscribe` function

This function subscribes to the GPS service in order to receive GPS related events.

- **Prototype**

```
s8      adl_gpsSubscribe ( adl_gpsHdlr_f   GpsHandler
                          u32             PollingTime );
```

- **Parameters**

GpsHandler:

GPS events handler defined using the following type:

```
typedef bool (*adl_gpsHdlr_f) ( adl_gpsEvent_e Event,
                                adl_gpsData_t* GpsData );
```

The events received by this handler are defined below:

ADL_GPS_EVENT_RESETING_HARDWARE

*If the ADL GPS service needs to reset the product, in order to enable the GPS device internal mode. The handler may refuse this reset by returning FALSE. If at least one handler refuses the reset, the service goes to ADL_GPS_STATE_EXT_MODE state. The **GpsData** parameter is set to NULL.*

ADL_GPS_EVENT_EXT_MODE

*If the at least one Handler refused the ADL_GPS_EVENT_RESETING_HARDWARE event, the service entered in ADL_GPS_STATE_EXT_MODE state, and will be available on next product reset. The **GpsData** parameter is set to NULL. Handler's returned value is not relevant.*

ADL_GPS_EVENT_IDLE

*If the service entered the ADL_GPS_STATE_IDLE state: the service is ready to read GPS data. The **GpsData** parameter is set to NULL. Handler's returned value is not relevant.*

ADL_GPS_EVENT_POLLING_DATA

*If a Polling Time was required on subscription. The **GpsData** contains all GPS data read from the GPS device. Handler's returned value is not relevant.*

The **GpsData** parameter is based on the following type:

```
typedef struct
{
    adl_gpsPosition_t   Position; // Current GPS position
    adl_gpsSpeed_t      Speed;    // Current GPS speed
    adl_gpsSatView_t    SatView;  // Current GPS satellite view
} adl_gpsData_t;
```

Position:

Current GPS position data; please refer to GPS service data structures in § 3.17.2

Speed:

Current GPS speed data; please refer to GPS service data structures in § 3.17.2

SatView:

Current GPS satellite view data; please refer to GPS service data structures in § 3.17.2

PollingTime:

Time interval (in seconds) between each GPS data polling event (ADL_GPS_EVENT_POLLING_DATA) reception by the GPS handler.

• **Returned values**

- This function returns a positive or null handle on success;
- ADL_RET_ERR_PARAM on parameter error,
- ADL_RET_ERR_NO_MORE_HANDLES if there are no more free handles,
- ADL_GPS_RET_ERR_NO_Q25_PRODUCT if the current product is not a Q2501 product.

3.17.4 The adl_gpsUnsubscribe function

This function un-subscribes from the GPS service. The corresponding GPS handler will no longer receive any GPS events.

• **Prototype**

```
s8 adl_gpsUnsubscribe ( u8 Handle );
```

• **Parameters**

Handle:

The handle returned by the adl_gpsSubscribe function.

• **Returned values**

- This function returns 0 on success,
- ADL_RET_ERR_NOT_SUBSCRIBED if the GPS service was not subscribed,
- ADL_RET_ERR_UNKNOWN_HDL if the handle provided is not a valid one,
- ADL_RET_ERR_BAD_STATE if the service is in INIT state.

3.17.5 The `adl_gpsGetState` function

This function returns the current GPS service state.

- **Prototype**

```
adl_gpsState_e adl_gpsGetState ( void );
```

- **Returned values**

The current GPS service state, based on following type:

```
typedef enum
{
    ADL_GPS_STATE_INIT, // Service initialization state
    ADL_GPS_STATE_NO_Q25, // Not a Q25 product
    ADL_GPS_STATE_RESETING_HARDWARE, // Trying to reset product after have
                                     set the GPS internal mode
    ADL_GPS_STATE_EXT_MODE, // Reset refused: will be on internal mode on
                             next product start-up
    ADL_GPS_STATE_IDLE // GPS driver in IDLE mode, ready to read data
} adl_gpsState_e;
```

3.17.6 The `adl_gpsGetPosition` function

This function gets the current position read from the GPS device.

- **Prototype**

```
s8 adl_gpsGetPosition ( u8 Handle, adl_gpsPosition_t * Position );
```

- **Parameters**

Handle:

The handle returned by the `adl_gpsSubscribe` function.

Position:

Position data read from the GPS device. Please refer to GPS service data structures in § 3.17.2

- **Returned values**

- This function returns OK on success.
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the GPS service was not subscribed,
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle provided is not a valid one,
- `ADL_RET_ERR_BAD_STATE` if the GPS service is out of IDLE state.

3.17.7 The `adl_gpsGetSpeed` function

This function gets the current speed read from the GPS device.

- **Prototype**

```
s8 adl_gpsGetSpeed ( u8 Handle, adl_gpsSpeed_t * Speed );
```

- **Parameters**

Handle:

The handle returned by the `adl_gpsSubscribe` function.

Speed:

Speed data read from the GPS device. Please refer to GPS service data structures in § 3.17.2

- **Returned values**

- This function returns OK on success.
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the GPS service was not subscribed,
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle provided is not a valid one,
- `ADL_RET_ERR_BAD_STATE` if the GPS service is out of IDLE state.

3.17.8 The `adl_gpsGetSatView` function

This function gets the current satellite view read from the GPS device.

- **Prototype**

```
s8 adl_gpsGetSatView ( u8 Handle, adl_gpsSatView_t * SatView );
```

- **Parameters**

Handle:

The handle returned by the `adl_gpsSubscribe` function.

SatView:

SatView data read from the GPS device. Please refer to GPS service data structures in § 3.17.2

- **Returned values**

- This function returns OK on success.
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the GPS service was not subscribed,
- `ADL_RET_ERR_UNKNOWN_HDL` if the handle provided is not a valid one,
- `ADL_RET_ERR_BAD_STATE` if the GPS service is out of IDLE state.

3.18 AT/FCM IO Ports Service

ADL applications may use this service to be informed about the product AT/FCM IO ports states.

3.18.1 Required Header File

The header file for the AT/FCM IO Ports service is:
adl_port.h

3.18.2 AT/FCM IO Ports

AT Commands and FCM services can be used to send and receive AT Commands or data blocks, to or from one of the product ports. These ports are linked either to product physical serial ports (as UART1 / UART2 / USB ports), or virtual ports (as Open AT® virtual AT port, GSM CSD call data port, GPRS session data port or Bluetooth virtual ports).

AT/FCM IO Ports are identified by the type below:

```
typedef enum
{
    ADL_PORT_NONE,
    ADL_PORT_UART1,
    ADL_PORT_UART2,
    ADL_PORT_USB,

    ADL_PORT_UART1_VIRTUAL_BASE    = 0x10,
    ADL_PORT_UART2_VIRTUAL_BASE    = 0x20,
    ADL_PORT_USB_VIRTUAL_BASE      = 0x30,
    ADL_PORT_BLUETOOTH_VIRTUAL_BASE = 0x40,
    ADL_PORT_GSM_BASE              = 0x50,
    ADL_PORT_GPRS_BASE             = 0x60,
    ADL_PORT_OPEN_AT_VIRTUAL_BASE  = 0x80
} adl_port_e;
```

The available ports are described below:

- ADL_PORT_NONE
Not usable
- ADL_PORT_UART1
*Product physical UART 1
Please refer to the AT+WMFM command documentation to know how to open/close this product port.*
- ADL_PORT_UART2
*Product physical UART 2
Please refer to the AT+WMFM command documentation to know how to open/close this product port.*

- **ADL_PORT_USB**
Product physical USB port (reserved for future products)
- **ADL_PORT_UART1_VIRTUAL_BASE**
*Base ID for 27.010 protocol logical channels on UART 1
Please refer to AT+CMUX command & 27.010 protocol documentation to know how to open/close such a logical channel.*
- **ADL_PORT_UART2_VIRTUAL_BASE**
*Base ID for 27.010 protocol logical channels on UART 2
Please refer to AT+CMUX command & 27.010 protocol documentation to know how to open/close such a logical channel.*
- **ADL_PORT_USB_VIRTUAL_BASE**
Base ID for 27.010 protocol logical channels on USB link (reserved for future products)
- **ADL_PORT_BLUETOOTH_VIRTUAL_BASE**
*Base ID for connected Bluetooth peripheral virtual port.
ONLY USABLE WITH THE FCM SERVICE
Please refer to the Bluetooth AT commands documentation to know how to connect, and how to open/close such a virtual port.*
- **ADL_PORT_GSM_BASE**
*Virtual Port ID for GSM CSD data call flow
ONLY USABLE WITH THE FCM SERVICE
Please note that this port will be considered as always available (no OPEN/CLOSE events for this port; adl_portIsAvailable function will always return TRUE)*
- **ADL_PORT_GPRS_BASE**
*Virtual Port ID for GPRS data session flow
ONLY USABLE WITH THE FCM SERVICE
Please note that this port will be considered as always available (no OPEN/CLOSE events for this port; adl_portIsAvailable function will always return TRUE) if the GPRS feature is supported on the current product.*
- **ADL_PORT_OPEN_AT_VIRTUAL_BASE**
*Base ID for AT commands contexts dedicated to Open AT® applications
ONLY USABLE WITH THE AT COMMANDS SERVICE
This port is always available, and is opened immediately at the product's start-up. This is the default port on which the AT commands sent by the AT Command service are executed.*

3.18.3 Ports test macros

Some ports & events test macros are provided. These macros are defined below.

- **ADL_PORT_IS_A_SIGNAL_CHANGE_EVENT(_e)**
Returns TRUE if the event "_e" is a signal change one, FALSE otherwise.
- **ADL_PORT_GET_PHYSICAL_BASE(_port)**
Extracts the physical port identifier part of the "_port" provided.

ADL User Guide for Open AT® OS v3.13

API

E.g. if used on a 27.010 virtual port identifier based on the UART 2, this macro will return `ADL_PORT_UART2`.

- **ADL_PORT_IS_A_PHYSICAL_PORT(_port)**
Returns TRUE if the “_port” provided is a physical output based one (E.g. UART1, UART2 or 27.010 logical ports), FALSE otherwise.
- **ADL_PORT_IS_A_PHYSICAL_OR_BT_PORT(_port)**
Returns TRUE is the “_port” provided is a physical output or a bluetooth based one, FALSE otherwise.
- **ADL_PORT_IS_AN_FCM_PORT(_port)**
Returns TRUE if the “_port” provided is able to handle the FCM service (i.e. all ports except the Open AT® virtual base ones), FALSE otherwise.
- **ADL_PORT_IS_AN_AT_PORT(_port)**
Returns TRUE if the “_port” provided is able to handle AT commands services (i.e. all ports except the GSM & GPRS virtual base ones), FALSE otherwise.

3.18.4 The `adl_portSubscribe` function

This function subscribes to the AT/FCM IO Ports service in order to receive specific port-related events.

- **Prototype**

```
s8      adl_portSubscribe ( adl_portHdlr_f PortHandler );
```

- **Parameters**

PortHandler:

Port-related events handler defined using the following type:

```
typedef void (*adl_portHdlr_f) ( adl_portEvent_e Event,
                                adl_port_e Port,
                                u8 State );
```

The events received by this handler are defined below:

ADL_PORT_EVENT_OPENED

*Notifies the ADL application that the specified **Port** is now opened. According to its type, it may now be used with either the AT Commands service or FCM service.*

ADL_PORT_EVENT_CLOSED

*Notifies the ADL application that the specified **Port** is now closed. It is no longer usable with either the AT Commands service or FCM service.*

ADL_PORT_EVENT_DSR_STATE_CHANGE

*Notifies the ADL application that the specified **Port** DSR signal state has changed to the new **State** value (0/1). This event will be received by all*

ADL User Guide for Open AT® OS v3.13

API

*subscribers that have started a polling process on the specified **Port** DSR signal with the `adl_portStartSignalPolling` function.*

ADL_PORT_EVENT_CTS_STATE_CHANGE

*Informs the ADL application that the specified **Port** CTS signal state has changed to the new **State** value (0/1). This event will be received by all subscribers that have started a polling process on the specified **Port** CTS signal with the `adl_portStartSignalPolling` function.*

The handler **Port** parameter uses the `adl_port_e` type described above. The handler **State** parameter is set only for the `ADL_PORT_EVENT_XXX_STATE_CHANGE` events.

- **Returned values**

- A positive or null handle on success;
- `ADL_RET_ERR_PARAM` on parameter error,
- `ADL_RET_ERR_NO_MORE_HANDLES` if there are no more free handles (the service is able to process up 127 subscriptions).

3.18.5 The `adl_portUnsubscribe` function

This function unsubscribes from the AT/FCM IO Ports service. The related handler will no longer receive port-related events. If a signal polling process was started only for this handle, it will be automatically stopped.

- **Prototype**

```
s8      adl_portUnsubscribe ( u8 Handle );
```

- **Parameters**

Handle:

Handle previously returned by the `adl_portSubscribe` function.

- **Returned values**

- OK on success;
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided handle is unknown ;
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the service is not subscribed.

3.18.6 The `adl_portIsAvailable` function

This function checks if the required port is currently opened or not.

- **Prototype**

```
bool    adl_portIsAvailable ( adl_port_e Port );
```

- **Parameters**

Port:

Port from which to require the current state.

- **Returned values**

- TRUE if the port is currently opened;
- FALSE if the port is closed, or if it does not exists.

- **Notes**

- The function will always return TRUE on the ADL_PORT_GSM_BASE port;
- The function will always return TRUE on the ADL_PORT_GPRS_BASE port if the GPRS feature is enabled (always FALSE otherwise).

3.18.7 The `adl_portGetSignalState` function

This function returns the required port signal state.

- **Prototype**

```
s8      adl_portGetSignalState ( adl_port_e Port,  
                                adl_portSignal_e Signal );
```

- **Parameters**

Port:

Port from which to require the current signal state. Only physical output related ports (UARTX & USB ports, used as physical ports, or with the 27.010 protocol) may be used with this function.

Signal:

Signal from which to query the current state, based on the following type:

```
typedef enum  
{  
    ADL_PORT_SIGNAL_CTS,  
    ADL_PORT_SIGNAL_DSR,  
  
    ADL_PORT_SIGNAL_LAST  
} adl_portSignal_e;
```

Signals are detailed below:

ADL_PORT_SIGNAL_CTS

Required port CTS input signal : physical pin for a physical port (UARTX), emulated logical signal for a 27.010 logical port.

ADL_PORT_SIGNAL_DSR

Required port DSR input signal : physical pin for a physical port (UARTX), emulated logical signal for a 27.010 logical port.

• **Returned values**

- The signal state (0/1) on success;
- ADL_RET_ERR_PARAM on parameter error;
- ADL_RET_ERR_BAD_STATE if the required port is not opened.

3.18.8 The adl_portStartSignalPolling function

This function starts a polling process on a required port signal for the provided subscribed handle.

Only one polling process can run at a time. A polling process is defined on one port, for one or several of this port's signals.

It means that this function may be called several times on the same port in order to monitor several signals; the polling time interval is set up by the first function call (polling time parameters are ignored or further calls). If the function is called several times on the same port & signal, additional calls will be ignored.

Once a polling process is started on a port's signal, this is monitored: each time this signal state changes, a ADL_PORT_EVENT_XXX_STATE_CHANGE event is sent to all the handlers which have required a polling process on it.

Whatever the number of requested signals and subscribers to this port polling process, a single cyclic timer will be internally used for this one.

• **Prototype**

```
s8      adl_portStartSignalPolling (u8 Handle,
                                   adl_port_e Port,
                                   adl_portSignal_e Signal,
                                   u8 PollingTimerType,
                                   u32 PollingTimerValue );
```


• **Parameters**

Handle:

Handle previously returned by the adl_portSubscribe function.

Port:

Port on which to run the polling process. Only physical output related ports (UARTX & USB ones, used as physical ports, or with the 27.010 protocol) may be used with this function.

Signal:

Signal to monitor during the polling process. See the adl_portGetSignalState function for information about the available signals.

PollingTimerType:

PollingTimerValue parameter value's unit. The allowed values are defined below:

Timer type	Timer unit
ADL_TMR_TYPE_100MS	PollingTimerValue is in 100 ms steps
ADL_TMR_TYPE_TICK	PollingTimerValue is in 18.5 ms tick steps

This parameter is ignored on additional function calls on the same port.

PollingTimerValue:

Polling time interval (uses the PollingTimerType parameter's value unit).

This parameter is ignored on additional function calls on the same port.

• **Returned values**

- OK on success ;
- ADL_RET_ERR_PARAM on parameter error;
- ADL_RET_ERR_UNKNOWN_HDL if the provided handle is unknown;
- ADL_RET_ERR_NOT_SUBSCRIBED if the service is not subscribed;
- ADL_RET_ERR_BAD_STATE if the required port is not opened;
- ADL_RET_ERR_ALREADY_SUBSCRIBED if a polling process is already running on another port.

3.18.9 The `adl_portStopSignalPolling` function

This function stops a running polling process on a required port signal for the provided subscribed handle.

The associated handler will no longer receive the `ADL_PORT_EVENT_XXX_STATE_CHANGE` events related to this signal port.

The internal polling process cyclic timer will be stopped as soon as the last subscriber to the current running polling process has called this function.

- **Prototype**

```
s8      adl_portStopSignalPolling ( u8 Handle,  
                                   adl_port_e Port,  
                                   adl_portSignal_e Signal );
```

- **Parameters**

Handle:

Handle previously returned by the `adl_portSubscribe` function.

Port:

Port on which the polling process to stop is running.

Signal:

Signal on which the polling process to stop is running.

- **Returned values**

- OK on success ;
- `ADL_RET_ERR_PARAM` on parameter error;
- `ADL_RET_ERR_UNKNOWN_HDL` if the provided handle is unknown;
- `ADL_RET_ERR_NOT_SUBSCRIBED` if the service is not subscribed;
- `ADL_RET_ERR_BAD_STATE` if the required port is not opened;
- `ADL_RET_ERR_BAD_HDL` if there is no running polling process for this Handle / Port / Signal combination.

3.19 RTC Service

ADL provides an RTC service to access to the module's inner RTC, and to process time related data.

3.19.1 Required Header File

The header file for the RTC functions is:

```
adl_rtc.h
```

3.19.2 RTC service types

3.19.2.1 The `adl_rtcTime_t` structure

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

```
typedef struct
{
    u8 Year;           // Year (Two digits)
    u8 Month;         // Month (1-12)
    u8 Day;           // Day of the month (1-31)
    u8 Hour;          // Hour (0-23)
    u8 Minute;        // Minute (0-59)
    u8 Second;        // Second (0-59)
    u16 SecondFracPart; // Second fractional part
} adl_rtcTime_t;
```

Years are cyclically provided on two digits, without any century information.

Second fractional part step is the `ADL_RTC_SECOND_FRACPART_STEP` constant. This field's most significant bit is not used (values are in the [0 – 0x7FFF] range).

3.19.2.2 The `adl_rtcTimeStamp_t` structure

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

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

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

Second fractional part step is the `ADL_RTC_SECOND_FRACPART_STEP` constant. This field's most significant bit is not used (values are in the [0 – 0x7FFF] range).

3.19.2.3 Constants

RTC service constants are defined below.

Constant	Value	Use
ADL_RTC_SECOND_FRACPART_STEP	30.5	Second fractional part step value (in μ s); The real value is $1/2^{15}$
ADL_RTC_DAY_SECONDS	24x60x60	Seconds count in a day
ADL_RTC_HOUR_SECONDS	60x60	Seconds count in an hour
ADL_RTC_MINUTE_SECONDS	60	Seconds count in a minute
ADL_RTC_MS_US	1000	μ seconds count in a millisecond

3.19.2.4 Macros

RTC service macros are defined below.

Macro	Parameter	Use
ADL_RTC_GET_TIMESTAMP_SECONDS(_t)	adl_rtcTimeStamp_t structure	Timestamp seconds part (0-59)
ADL_RTC_GET_TIMESTAMP_MINUTES(_t)	adl_rtcTimeStamp_t structure	Timestamp minutes part (0-59)
ADL_RTC_GET_TIMESTAMP_HOURS(_t)	adl_rtcTimeStamp_t structure	Timestamp hours part (0-23)
ADL_RTC_GET_TIMESTAMP_DAYS(_t)	adl_rtcTimeStamp_t structure	Timestamp days part
ADL_RTC_GET_TIMESTAMP_MS(_t)	adl_rtcTimeStamp_t structure	Timestamp milliseconds part (0-999)
ADL_RTC_GET_TIMESTAMP_US(_t)	adl_rtcTimeStamp_t structure	Timestamp microseconds part (0-999)

These macros may be used in order to extract duration parts from a given timestamp; the logical equations below are always true:

```

_t.TimeStamp == ADL_RTC_GET_TIMESTAMP_SECONDS(_t) + ADL_RTC_GET_TIMESTAMP_MINUTES(_t) *
                ADL_RTC_MINUTE_SECONDS + ADL_RTC_GET_TIMESTAMP_HOURS(_t) *
                ADL_RTC_HOUR_SECONDS +
                ADL_RTC_GET_TIMESTAMP_DAYS(_t) * ADL_RTC_DAY_SECONDS

_t.SecondFracPart * ADL_RTC_SECOND_FRACPART_STEP ==
                ADL_RTC_GET_TIMESTAMP_MS(_t) * ADL_RTC_MS_US +
                ADL_RTC_GET_TIMESTAMP_US(_t)

```

3.19.3 The `adl_rtcGetTime` function

This function retrieves the current RTC time structure.

- **Prototype**
`s32 adl_rtcGetTime (adl_rtcTime_t * TimeStructure);`
- **Parameters**
TimeStructure:
Retrieved current time structure.
- **Returned values**
 - OK on success.
 - `ADL_RET_ERR_PARAM` if the parameter is incorrect.

3.19.4 The `adl_rtcConvertTime` function

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

- **Prototype**
`s32 adl_rtcConvertTime (adl_rtcTime_t * TimeStructure,
adl_rtcTimeStamp_t * TimeStamp,
adl_rtcConvert_e Conversion);`
- **Parameters**
TimeStructure:
Input / output RTC time structure
TimeStamp:
Input / output timestamp structure
Conversion:
Conversion mode, using the type below:

```
typedef enum  
{  
    ADL_RTC_CONVERT_TO_TIMESTAMP,  
    ADL_RTC_CONVERT_FROM_TIMESTAMP  
} adl_rtcConvert_e;
```

ADL_RTC_CONVERT_TO_TIMESTAMP

This mode allows the `TimeStructure` parameter to be converted to a `TimeStamp` parameter. Since RTC structure years are only available on two digits, real years will be considered from 1970 to 2069.

ADL_RTC_CONVERT_FROM_TIMESTAMP

This mode allows the TimeStamp parameter to be converted to a TimeStructure parameter. Since RTC structure years are only available on two digits, timestamps greater or equal to 2070 year will lead to a conversion error.

• **Returned values**

- OK on success.
- ERROR if conversion failed (internal error).
- ADL_RET_ERR_PARAM if one parameter value is incorrect.
- ADL_RET_ERR_OVERFLOW if a "From Timestamp" conversion is required on a year greater or equal to 2070.

3.19.5 The adl_rtcDiffTime function

This function allows the difference between two timestamp structures to be reckoned.

• **Prototype**

```
s32 adl_rtcDiffTime (    adl_rtcTimeStamp_t *   TimeStamp1,
                        adl_rtcTimeStamp_t *   TimeStamp2,
                        adl_rtcTimeStamp_t *   Result );
```

• **Parameters**

TimeStamp1:
First timestamp

TimeStamp2:
Second timestamp

Result:
Time difference between the two timestamps provided

• **Returned values**

- 0 on success, and if TimeStamp1 equals to TimeStamp2.
- 1 on success, and if TimeStamp1 is greater than TimeStamp2.
- -1 on success, and if TimeStamp2 is greater than TimeStamp1.
- ADL_RET_ERR_PARAM if one parameter value is incorrect.

3.20 DAC Service

3.20.1 Required Header File

The header file for the functions dealing with the DAC interface is:
adl_dac.h

3.20.2 The adl_dacSubscribe function

This function subscribes to one of the module DAC block interfaces.

- **Prototype**

```
s32 adl_dacSubscribe ( adl_dacChannel_e Channel,
                    adl_dacParam_t * Parameters )
```

- **Parameters**

Channel:

The DAC channel identifier to be subscribed, using the type below:

```
typedef enum
{
    ADL_DAC_CHANNEL_1,
    ADL_DAC_NUMBER_OF_CHANNEL,
    ADL_DAC_CHANNEL_PAD = 0x7fffffff
} adl_dacChannel_e;
```

Channel identifiers depend on the current module type (please refer to the module Product Technical Specification document for more information):

Module type	Channel identifier	Output DAC PIN name	Output DAC PIN number
Q2501	ADL_DAC_CHANNEL_1	AUXDAC	31

Parameters:

DAC channel initialization parameters, using the type below:

```
typedef struct {
    u32 InitialValue;
} adl_dacParam_t;
```

InitialValue:

Initial value to be written on the DAC just after this has been opened. Significant bits and output voltage depends on the module type (please refer to the module Product Technical Specification document for more information).

Module type	Significant bits	Max. output voltage
Q2501	8 less significant bits	2.64 V (for 0xFF value)

- **Returned values**

- A positive or null value on success:
 - DAC service handle, to be used with further DAC service functions calls.
- A negative error value otherwise:
 - ADL_RET_ERR_ALREADY_SUBSCRIBED if the required channel has already been subscribed.
 - ADL_RET_ERR_NO_MORE_HANDLES if there are no more free DAC handles.
 - ADL_RET_ERR_NOT_SUPPORTED if the current module does not support the DAC service.
 - ADL_RET_ERR_PARAM on parameter error.

- **Notes**

The DAC service is only available on the Q2501 product.

3.20.3 The `adl_dacUnsubscribe` function

This function un-subscribes from a previously subscribed DAC block.

- **Prototype**

```
s32 adl_dacUnsubscribe ( s32 Handle )
```

- **Parameters**

Handle:

DAC service handle previously returned by the `adl_dacSubscribe` function.

- **Returned values**

- OK on success
- ADL_RET_ERR_UNKNOWN_HDL if the handle provided is unknown

3.20.4 The `adl_dacWrite` function

This function allows the output value of a subscribed DAC block to be set.

- **Prototype**

```
s32 adl_dacWrite ( s32 Handle,  
                 u32 Value )
```

- **Parameters**

Handle:

DAC service handle previously returned by the `adl_dacSubscribe` function.

Value:

Value to be written on the DAC output. Significant bits and output voltage depend on module type (please refer to the module Product Technical Specification document for more information).

Module type	Significant bits	Max. output voltage
Q2501	8 less significant bits	2.64 V (for 0xFF value)

• **Returned values**

- OK on success
- ADL_RET_ERR_PARAM on parameter error.

3.20.5 Example

This example just demonstrates how to use the DAC service in a nominal case (error cases not handled).

A full example using the DAC service is also available on the SDK (ADL generic DAC sample).

```
// Global variable
s32 MyDACHandle;

...

// Somewhere in the application code, used as an event handler
void MyFunction ( void )
{
    // Initialization structure
    adl_dacParam_t InitStruct = { 0 };

    // Subscribe to the DAC service
    MyDACHandle = adl_dacSubscribe ( ADL_DAC_CHANNEL_1, &InitStruct );

    // Write a value on the DAC block
    adl_dacWrite ( MyDACHandle, 80 );

    ...

    // Write another value on the DAC block
    adl_dacWrite ( MyDACHandle, 190 );

    ...

    // Unsubscribe from the DAC service
    adl_dacUnsubscribe ( MyDACHandle );
}
```

4 Error codes

4.1 General error codes

Error code	Error value	Description
OK	0	No error response
ERROR	-1	general error code
ADL_RET_ERR_PARAM	-2	parameter error
ADL_RET_ERR_UNKNOWN_HDL	-3	unknown handler / handle error
ADL_RET_ERR_ALREADY_SUBSCRIBED	-4	service already subscribed
ADL_RET_ERR_NOT_SUBSCRIBED	-5	service not subscribed
ADL_RET_ERR_FATAL	-6	fatal error
ADL_RET_ERR_BAD_HDL	-7	Bad handle
ADL_RET_ERR_BAD_STATE	-8	Bad state
ADL_RET_ERR_PIN_KO	-9	Bad PIN state
ADL_RET_ERR_NO_MORE_HANDLES	-10	The service subscription maximum capacity is reached
ADL_RET_ERR_DONE	-11	The required iterative process is now terminated
ADL_RET_ERR_OVERFLOW	-12	The required operation has exceeded the function capabilities
ADL_RET_ERR_NOT_SUPPORTED	-13	An option, required by the function, is not enabled on the Wireless CPU®: the function is not supported in this configuration
ADL_RET_ERR_SPECIFIC_BASE	-20	Beginning of specific errors range

4.2 Specific FCM service error codes

Error code	Error value
ADL_FCM_RET_ERROR_GSM_GPRS_ALREADY_OPENED	ADL_RET_ERR_SPECIFIC_BASE
ADL_FCM_RET_ERR_WAIT_RESUME	ADL_RET_ERR_SPECIFIC_BASE-1
ADL_FCM_RET_OK_WAIT_RESUME	OK+1
ADL_FCM_RET_BUFFER_EMPTY	OK+2
ADL_FCM_RET_BUFFER_NOT_EMPTY	OK+3

4.3 Specific flash service error codes

Error code	Error value
ADL_FLH_RET_ERR_OBJ_NOT_EXIST	ADL_RET_ERR_SPECIFIC_BASE
ADL_FLH_RET_ERR_MEM_FULL	ADL_RET_ERR_SPECIFIC_BASE-1
ADL_FLH_RET_ERR_NO_ENOUGH_IDS	ADL_RET_ERR_SPECIFIC_BASE-2
ADL_FLH_RET_ERR_ID_OUT_OF_RANGE	ADL_RET_ERR_SPECIFIC_BASE-3

4.4 Specific GPRS service error codes

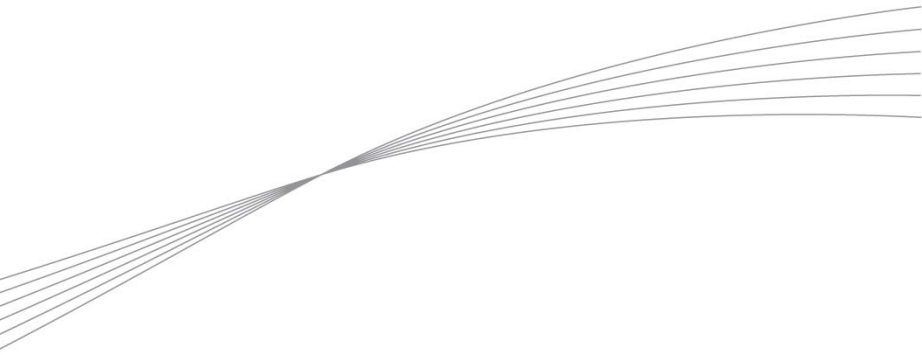
Error code	Error value
ADL_GPRS_CID_NOT_DEFINED	-3
ADL_NO_GPRS_SERVICE	-4
ADL_CID_NOT_EXIST	5

4.5 Specific GPS service error codes

Error code	Error value
ADL_GPS_RET_ERR_NO_Q25_PRODUCT	ADL_RET_ERR_SPECIFIC_BASE

4.6 Specific A&D storage service error codes

Error code	Error value
ADL_AD_RET_ERR_NOT_AVAILABLE	ADL_RET_ERR_SPECIFIC_BASE
ADL_AD_RET_ERR_OVERFLOW	ADL_RET_ERR_SPECIFIC_BASE - 1
ADL_AD_RET_ERROR	ADL_RET_ERR_SPECIFIC_BASE - 2
ADL_AD_RET_ERR_NEED_RECOMPACT	ADL_RET_ERR_SPECIFIC_BASE - 3



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