ADLIB WIN Interface Library Software

User's Manual for ADAC Data Acquisition Boards

the smart approach to instrumentation ™

IOtech, Inc.
25971 Cannon Road
Cleveland, OH 44146-1833
Phone: (440) 439-4091
Fax: (440) 439-4093
E-mail (sales): sales@iotech.com
E-mail (post-sales): productsupport@iotech.com
Internet: www.iotech.com

ADLIB WIN

Interface Library Software
for ADAC Data Acquisition Boards

p/n 1107-0902 Rev 1.0

DAQ Board API Functions
Version 3.30 03/07/2002

1. INTRODUCTION

1.1 OVERVIEW

ADLIB WIN is a set of sophisticated, high level, dynamically linked library (DLL) data acquisition subroutines for programmers involved in the developing of process and/or data acquisition applications. The functions supplied with ADLIB provide an easy to use interface to ADAC PC series data acquisition products, shielding the programmer from both the complexity of low level DAQ board programming and the complicated DMA and interrupt handling mechanisms of the Windows environment. ADLIB supports DMA, Interrupt and Software Polled data transfer methods for acquiring data.

ADLIB WIN removes the complex task of allocating suitable DMA, and Interrupt buffers by automatically allocating buffer(s) when a DAQ board's subsystem is initialized. The application programmer can then retrieve a pointer to buffer and/or set up a windows message handler routine that will be called when each buffer has been completed. ADLIB allows N buffers of size M for each DAQ board's logical devices and the only limitation on quantity and size of the buffer(s) is the amount of available system memory.

ADLIB WIN buffer pointers are not just pointers to a DAQ board's subsystem data, they point to a structure that contains the buffer type, size, linear data address, physical data address, current sample count, hardware trigger points, completion flags, status flags and error flags.

ADLIB WIN provides support for an *unlimited number* of DAQ boards, the only restrictions being available memory, DMA channels and Interrupt levels in the host system.

ADLIB WIN provides a *multitasking environment* through which multiple logical devices can run concurrent and report directly back to the user application when the requested ADLIB service has completed. ADLIB services can be either a single cycle or continuous cycle operation.

ADLIB WIN provides extensive error reporting through each library call and buffer structure. In addition, ADLIB employs error control functions that can enable or disable certain types of error conditions.

The ADLIB WIN Series of drivers consists of the following:

ADLIB WDM – used with ADAC PCI series of DAQ boards for Windows 98/ME/2000; based on Microsoft's Windows Driver Model, also includes support for ADAC PCI Series boards for Windows NT 4.0.

ADLIB WIN 95 – used with ADAC ISA series of DAQ boards for Windows 95/98

ADLIB WIN NT – used with ADAC ISA series of DAQ boards for Windows NT 4.0

ADLIB WIN 16 – used with ADAC ISA series of DAQ boards for Windows 3.1

Throughout the manual, unless otherwise specified, ADLIB WIN refers to all of the above packages.

1.2 SYSTEM REQUIREMENTS

ADLIB WIN runs on the IBM family of personal computers. Specifically IBM PC/386/486/Pentium and compatibles. Minimum system requirements are:

- MS-DOS/PC-DOS, Version 3.1 or higher.
- 4M Ram, 8Mb recommended.
- Microsoft Windows 3.1 running in 386 Enhanced mode, Microsoft Windows for WorkGroups 3.1 running in 386 Enhanced mode, Microsoft Windows 95/98/ME, or Windows 2000 or Windows NT 4.00
- 3.5” (1.44Mb) floppy diskette drive.
- One fixed disk drive.

It is recommended that 16Mb of RAM be available and a numeric coprocessor be installed where applicable. ADLIB WIN does not require the coprocessor, but most data acquisition, analysis and control applications will benefit from its availability.

1.3 APPLICATION LANGUAGE REQUIREMENTS

The ADLIB-WIN API interface is provided through the Windows dynamic link library (DLL) interface specification. Example programs are provided for each independent language, with various acquisition methods demonstrated.

Languages supported are:

Microsoft Windows Languages

Visual Basic	16 or 32 bit
Visual C/C++	16 or 32 bit

2. INSTALLATION

2.1 INSTALLATION FROM WINDOWS for PCI CARD SUPPORT

If you are installing **ADLIB WDM** for a **PCI** based product, follow the installatiion instructions on the CD provided with the PCI board.

If you are using ADLIB WDM with Windows NT 4.0, you must do the following:

1. Manually copy the “aPcixx.sys” file from the ADAC CD’s NT4 directory to your \WinNT\system32\drivers directory.

2. From the ADAC CD’s NT4 directory, run the “aPci55xx.reg” file by double clicking on it, this will update your registry to look for ADAC PCI boards.

3. Reboot your PC to load the ADAC drivers.

2.2 INSTALLATION FROM WINDOWS for ISA CARD SUPPORT

The installation instructions below are only for older **ISA** based products.

To Start Setup:

1. Insert Disk 1 in Drive A.
2. From the file or Start menu, choose Run.
3. Type **a:setup**
4. follow the setup instruction on the screen.

Note that the use of the Setup programs default installation directory name **adlib** is highly recommended and will ease the first time installation because the directory search paths specified in the ADLIB configuration file (.CON) and example programs have been set to match the default directory structure.

After setup has completed:

1) For **ADLIB WIN95** installs, the following files have been added to your win95 directory.

adacdm32.VXD	DMA and Interrupt support Vxd services
adlcore.dll	ADLIB interface DLL for C/C++
adlgrm.dll	ADLIB global resource manager
adlvb32.dll	ADLIB interface DLL for Visual Basic
msghoo32.oca	Visual Basic message handler
msghoo32.ocx	Visual Basic message handler

The following changes have been made to your system.ini file in the [386Enh] section.

```
device = *vdmad                 (If not already present.)
device = adacdm32.vxd
ADACDMABUFFERSIZE=256
```

Once the system.ini file has been updated, restart Windows 95 to load the ADAC virtual DMA device driver.

For **ADLIB WIN NT** installs, the following files have been added to your WINNT\SYSTEM32 and WINNT\SYSTEM32\DRIVERS directory.

C:\WINNT\SYSTEM32	C:\WINNT\SYSTEM\DRIVERS
adlcore.dll	a*.SYS (Low level board drivers)
adlgrm.dll	(as selected from INSTALL)
adlvb32.dll	
msghoo32.oca	
msghoo32.ocx	

Once the installation has completed, use Explorer to edit the newly installed boards registry installation file (boardname.reg) located in the adlib\alwinNT\DRIVERS\(boardname) directory.

Edit the following fields and save the file.

IOPortRange 0_low = dword:00000120	(Board Address)
DMA0 = dword:00000005	(DMA Channel)
IRQ0 = dword:00000005	(IRQ Level)

Double click the file to install the boards registry settings.

For **ADLIB WIN16** installs, the following files have been added to your windows directory.

adacdm16.386	DMA support Vxd services
adlcore.dll	ADLIB interface DLL
adlgrm.dll	ADLIB global resource manager

The following changes have been made to your system.ini file in the [386Enh] section.

device = *vdmad	(If not already present.)
device = adacdm16.386	
ADACDMABUFFERSIZE=256	

Once the system.ini file has been updated, restart Windows to load the ADAC virtual DMA device driver.

2. If the default installation directory "**adlib**" was **NOT** used, the example programs Configuration file ADLWIN16.con, ADLWIN32.con, or ADLWINNT.con will require changes. Edit the file and set the following entries to the appropriate directory locations.

BoardInifile =	i.e. c:\mydir\alwin16\5800mfhr\examples\msc\adlai01\adlib.ini
BoardCapsfile =	i.e. c:\mydir\alwin16\5800mfhr\5801mf.cap
DeviceDrvPath =	i.e. c:\mydir\alwin16\5800mfhr\

3. Check our web site for any updates to ADLIB that may not have been included on your release.

The directory contains individual file updates that have not been included in your release.

2.3 SUMMARY OF DISK FILES FOR ADLIB WDM PCI CARD SUPPORT

\adlib (Root Directory)

license.txt	

\alWdm

readme.txt	

\alWdm\msc

adlerr.h	ADLIB core error codes.
adlib.h	ADLIB include file.
drverr.h	ADLIB driver specific error codes.
adlcore.lib	ADLIB import library.

\alWdm\vbasic

adlerr.bas	ADLIB core error codes.
adlib.bas	ADLIB include file.
drverr.bas	ADLIB driver specific error codes.

\alWdm\drivers\ BOARD_TYPE

These directories contain the board specific drivers (*.dll) and capabilities (*.cap) file(s).

\alWdm\drivers\BOARD_TYPE\examples\msc

\Adlai01	examines the BUFFER_DONE message response handler
\Adlai02	examines the User polling of the BUFFER_DONE flag.
\Adlai03	examines the Thermocouple temperature conversion routine
\Adlai04	examines the ABOUT Trigger mode.
\Adlai05	examines the BURST/SCAN MODE option.
\Adlai06	examines the Software Transfer method with ABOUT Trigger mode.
\Adlai07	examines the Software Transfer method.
\Adlda01	examines the BUFFER_DONE message response handler
\Adlda02	examines the User polling of the BUFFER_DONE flag.
\Adlda03	examines the Software Transfer method.
\Adldio01	examines the Software Transfer Digital Input/Output functions.
\Adldio02	examines the Digital Input/Output Handshaking functions.
\Adldio03	examines the Digital Input/Output 16 bit PortResolution functions.

\alWdm\drivers\ BOARD_TYPE \examples\vbasic

Adlai01.frm	examines the BUFFER_DONE message response handler
Adlai02.frm	examines the User polling of the BUFFER_DONE flag.
Adlai03.frm	examines the Thermocouple temperature conversion routine
Adlai04.frm	examines the ABOUT Trigger mode.
Adlai05.frm	examines the BURST/SCAN MODE option.
Adlai06.frm	examines the Software Transfer method with ABOUT Trigger mode.
Adlai07.frm	examines the Software Transfer method.
Adlda01.frm	examines the Analog Output function.
\Adlda01.frm	examines the BUFFER_DONE message response handler
\Adlda02.frm	examines the User polling of the BUFFER_DONE flag.
\Adlda03.frm	examines the Software Transfer method.
Adldio01.frm	examines the Software Transfer Digital Input/Output functions.
Adldio02.frm	examines the Digital Input/Output Handshaking functions.
Adldio03.frm	examines the Digital Input/Output 16 bit PortResolution functions.

2.4 SUMMARY OF DISK FILES FOR ADLIB WIN95

\adlib (Root Directory)

adlibwin.hlp	
license.txt	

\alwin95

readme.txt	

\alwin95\msc

adlerr.h	ADLIB core error codes.
adlib.h	ADLIB include file.
drverr.h	ADLIB driver specific error codes.
adlcore.lib	ADLIB import library.

\alwin95\vbasic

adlerr.bas	ADLIB core error codes.
adlib.bas	ADLIB include file.
drverr.bas	ADLIB driver specific error codes.

\alwin95\drivers\ BOARD_TYPE

These directories contain the board specific drivers (*.dll) and capabilities (*.cap) file(s).

\alwin95\drivers\BOARD_TYPE\examples\msc

\Adlai01	examines the BUFFER_DONE message response handler
\Adlai02	examines the User polling of the BUFFER_DONE flag.
\Adlai03	examines the Thermocouple temperature conversion routine
\Adlai04	examines the ABOUT Trigger mode.
\Adlai05	examines the BURST/SCAN MODE option.
\Adlai06	examines the Software Transfer method with ABOUT Trigger mode.
\Adlai07	examines the Software Transfer method.
\Adlda01	examines the Analog Output function.
\Adldio01	examines the Software Transfer Digital Input/Output functions.
\Adldio02	examines the Digital Input/Output Handshaking functions.
\Adldio03	examines the Digital Input/Output 16 bit PortResolution functions.

\alwin95\drivers\ BOARD_TYPE \examples\vbasic

Adlai01.frm	examines the BUFFER_DONE message response handler
Adlai02.frm	examines the User polling of the BUFFER_DONE flag.
Adlai03.frm	examines the Thermocouple temperature conversion routine
Adlai04.frm	examines the ABOUT Trigger mode.
Adlai05.frm	examines the BURST/SCAN MODE option.
Adlai06.frm	examines the Software Transfer method with ABOUT Trigger mode.
Adlai07.frm	examines the Software Transfer method.
Adlda01.frm	examines the Analog Output function.
Adldio01.frm	examines the Software Transfer Digital Input/Output functions.
Adldio02.frm	examines the Digital Input/Output Handshaking functions.
Adldio03.frm	examines the Digital Input/Output 16 bit PortResolution functions.

2.5 SUMMARY OF DISK FILES FOR ADLIB WINNT

\adlib (Root Directory)

license.txt	

\alwinNT

readme.txt	

\alwinNT\msc

adlerr.h	ADLIB core error codes.
adlib.h	ADLIB include file.
drverr.h	ADLIB driver specific error codes.
adlcore.lib	ADLIB import library.

\alwinNT\vbasic

adlerr.bas	ADLIB core error codes.
adlib.bas	ADLIB include file.
drverr.bas	ADLIB driver specific error codes.

\alwinNT\drivers\ BOARD_TYPE

These directories contain the board specific drivers (*.dll) and capabilities (*.cap) file(s).

\alwinNT\drivers\BOARD_TYPE\examples\msc

\Adlai01	examines the BUFFER_DONE message response handler
\Adlai02	examines the User polling of the BUFFER_DONE flag.
\Adlai03	examines the Thermocouple temperature conversion routine
\Adlai04	examines the ABOUT Trigger mode.
\Adlai05	examines the BURST/SCAN MODE option.
\Adlai06	examines the Software Transfer method with ABOUT Trigger mode.
\Adlai07	examines the Software Transfer method.
\Adlda01	examines the Analog Output function.
\Adldio01	examines the Software Transfer Digital Input/Output functions.
\Adldio02	examines the Digital Input/Output Handshaking functions.
\Adldio03	examines the Digital Input/Output 16 bit PortResolution functions.

\alwinNT\drivers\ BOARD_TYPE \examples\vbasic

Adlai01.frm	examines the BUFFER_DONE message response handler
Adlai02.frm	examines the User polling of the BUFFER_DONE flag.
Adlai03.frm	examines the Thermocouple temperature conversion routine
Adlai04.frm	examines the ABOUT Trigger mode.
Adlai05.frm	examines the BURST/SCAN MODE option.
Adlai06.frm	examines the Software Transfer method with ABOUT Trigger mode.
Adlai07.frm	examines the Software Transfer method.
Adlda01.frm	examines the Analog Output function.
Adldio01.frm	examines the Software Transfer Digital Input/Output functions.
Adldio02.frm	examines the Digital Input/Output Handshaking functions.
Adldio03.frm	examines the Digital Input/Output 16 bit PortResolution functions.

2.6 SUMMARY OF DISK FILES FOR ADLIB WIN16

\adlib (Root Directory)

adlibwin.hlp	
license.txt	

\alwin16

readme.txt	

\alwin16\msc

adlerr.h	ADLIB core error codes.
adlib.h	ADLIB include file.
drverr.h	ADLIB driver specific error codes.
adlcore.lib	ADLIB import library.

\alwin16\vbasic

adlerr.bas	ADLIB core error codes.
adlib.bas	ADLIB include file.
drverr.bas	ADLIB driver specific error codes.

\alwin16\drivers\ BOARD_TYPE

These directories contain the board specific drivers (*.dll) and capabilities (*.cap) file(s).

\alwin16\drivers\BOARD_TYPE\examples\msc

\Adlai01	examines the BUFFER_DONE message response handler
\Adlai02	examines the User polling of the BUFFER_DONE flag.
\Adlai03	examines the Thermocouple temperature conversion routine
\Adlai04	examines the ABOUT Trigger mode.
\Adlai05	examines the BURST/SCAN MODE option.
\Adlai06	examines the Software Transfer method with ABOUT Trigger mode.
\Adlai07	examines the Software Transfer method.
\Adlda01	examines the Analog Output function.
\Adldio01	examines the Software Transfer Digital Input/Output functions.
\Adldio02	examines the Digital Input/Output Handshaking functions.
\Adldio03	examines the Digital Input/Output 16 bit PortResolution functions.

\alwin16\drivers\ BOARD_TYPE \examples\vbasic

Adlai01.frm	examines the BUFFER_DONE message response handler
Adlai02 .frm	examines the User polling of the BUFFER_DONE flag.
Adlai03.frm	examines the Thermocouple temperature conversion routine
Adlai04.frm	examines the ABOUT Trigger mode.
Adlai05.frm	examines the BURST/SCAN MODE option.
Adlai06.frm	examines the Software Transfer method with ABOUT Trigger mode.
Adlai07.frm	examines the Software Transfer method.
Adlda01.frm	examines the Analog Output function.
Adldio01.frm	examines the Software Transfer Digital Input/Output functions.
Adldio02.frm	examines the Digital Input/Output Handshaking functions.
Adldio03.frm	examines the Digital Input/Output 16 bit PortResolution functions.

3. CONFIGURATION, CAPABILITIES AND INITIALIZATION FILES

Beginning with ADLIB version 3.00, a configuration utility program ADACCONFIG.EXE is provided to setup and configure your ADLIB boards within the PC. The utility provides all the necessary information to create both the Configuration (.CON) and Initialization (.INI) files. The following sections provide the necessary detailed information on the .con and .ini files for manual configuration on these files and are not required reading when using the ADACCONFIG utility. The ADACCONFIG.EXE is located in your main ADLIB directory or form the programs menu group chosen when ADLIB was installed.

3.1 CONFIGURATION (.CON) FILE

A Configuration file contains settings that apply for all applications running in the environment using the ADLIB WIN Library. There is only one Configuration file per system. This file contains "static" system-wide configuration information about all the boards and drivers installed in the system. This file is read when the environment is loaded by the user's application. The Configuration file has a .CON filename extension.

Contents of the Configuration file include:

- The environment specific string name.
- An assigned DAQ board designator number for each installed board (this is the board's "ID number").
- The board specific string name.
- I/O or memory base address of each board.
- Path and filename of the capabilities file for each DAQ board (from the capabilities file, ADLIB WIN knows the type of the board, and everything else about it).
- Path to directory containing board driver, for each unique type of board (the capabilities file specifies the name of the file, but not the path).
- All available device subsections supported by each DAQ board.

3.2 CAPABILITIES (.CAP) FILE

A Capabilities file contains all the possible programmable options for each ADLIB function call for a particular DAQ board. There is one Capabilities file per unique DAQ board, ADLIB uses this information to validate all application program function call parameters, and when initializing a logical device from the initialization file described below. The Capabilities file has a .CAP filename extension.

Contents include:

- Device capabilities: what features and options the DAQ board supports
- How the board's subsections are divided up into logical hardware devices (such as ADC0, DAC0, DAC1, DOT0, DIN3, CTR0, etc.)

3.3 INITIALIZATION (.INI) FILE

The initialization file specifies how the programmable features of boards and logical devices are to be initialized during the ADLIB device initialization function call. There is only one .INI file per application, and the name and location is specified in the .CON file. This file is read during the allocation of a logical device.

The initialization file reduces the amount of user programming required by initially setting the ADLIB logical device subsystem database (LDSD) to the desired user application defaults. This feature allows only the bare minimum amount of ADLIB calls to start a logical device. The Initialization file has an .INI filename extension.

Contents include:

One or more named sections, each corresponding to a Logical Device, including information for data transfer method, number of channels enabled, channel gain settings, sampling mode, sampling rate, notification method, and logical device option information.

If an .INI parameter is not specified, but the capabilities file indicated support for the option, the parameter will default to the first option for that entry in the capabilities file.

4. APPLICATION DEVELOPMENT

4.1 OVERVIEW

The ADLIB WIN driver is an extensive set of both function and structure based library routines that provide an easy to use, hardware independent, software interface to ADAC products. Developers are shielded from the time consuming task of designing and interfacing to DAQ hardware in any environment. This allows the programmer to spend less time in the development, design and coding of an application program.

4.2 OPERATING MODES

All ADLIB DAQ board acquisitions specify operation mode of the hardware and software. The modes include the Software Cycles, Data Transfer and Notification Methods defined below.

Cycle Modes:

AL_SINGLE_CYCLE — In Single Cycle mode, after a device is started, it transfers a specified number of samples between the device and one or more buffers in the processor's memory and then stops. After each buffer is filled, the calling application program is "notified" that a transfer has completed.

AL_CONTINUOUS_CYCLE — In Continuous Cycle mode, after you start a device, it continues transferring buffers in circular fashion. If you have more than one buffer defined, it fills up the first, moves to the next, the next, etc., until it fills the last one. Then the device "wraps around" and starts filling the first buffer again. After each buffer is filled, the calling application program is notified that a transfer has completed.

Data Transfer Methods:

AL_DTM_SOFTWARE — The Software Transfer Method has two distinct modes of operation, depending on the availability of a hardware FIFO. The following descriptions assume a hardware clocking source is selected.

a) For DAQ boards without a FIFO, the driver must sit in a tight loop and look for the data ready flag (in a board register) and collect each sample individually. When all buffers have been completed, the driver returns control to the calling application.

b) When a FIFO is available, the driver collects all samples available in the FIFO each time the AL_GetDoneBufferPtr() function is called, returning a pointer to the buffer only when it has filled with the specified number of samples. It's the applications responsibility to call the AL_GetDoneBufferPtr() function in a timely manner, otherwise the FIFO will overflow, generating an error condition. In this case a pointer to the incomplete buffer will be returned with the appropriate error flags set in the buffer structure itself.

Data Transfer Methods *continued*

AL_DTM_DMA	The DMA Transfer Method is asynchronous, meaning the transfers happen automatically in the background without software intervention while the application program is doing something else. The application program calls AL_GetDoneBufferPtr() function to determine if an input buffer is ready for processing, or if an output buffer is ready to accept new data.
AL_DTM_IRQ	The interrupt (IRQ) Transfer Method is asynchronous, meaning the transfers happen automatically in the background while the application program is doing something else. The application program calls AL_GetDoneBufferPtr() function to determine if a input buffer is ready for processing, or if an output buffer is ready to accept new data.

Notification Methods:

AL_CHECK_BUFFER	This is a "synchronous" method where the application calls a GetDoneBuffPtr() function to determine whether or not the buffer transfer is complete. If any buffers have been completed, a pointer to the first buffer completed is returned. The retrieval of buffers is always done in a first completed, first returned order. Once the application has processed the data in an input buffer, it must call a function to clear the Buffer Done flag -- this tells the Library functions that it is OK to reuse the buffer. If the Library gets to the point where it needs to use the buffer again, and the Buffer Done flag hasn't cleared, the Library will flag an error condition.
AL_POSTMESSAGE	This is an "asynchronous" method where the library sends a message to the application when the buffer is done. The application sets up this message handler function through the use of standard library routines. That function would then process the data in the buffers. Again, the application is responsible for clearing the Buffer Done flag as described above. See below for Postmessage handler examples.

C/C++ POSTMESSAGE HANDLER EXAMPLE

```
long OnBufferHandler(WPARAM wParam, LPARAM lParam)
{
float EngUnits[NUM_CHANNELS];
long lBuffNum;

/* Get the pointer to the buffer structure */
lpbuff = (LPBUFFSTRUCT)lParam;
/* the buffer number is wParam */
lBuffNum = wParam;

/* Check buffer errors flags */
if  (lpbuff->lErrorFlags != 0)
   OnBufferError(lpbuff);

/* Check buffer status flags */
if  (lpbuff->dwStatusFlags != BUFFER_FULL)
   OnBufferError(lpbuff);

/* Convert to Eng. units */
AL_DemuxData(lhldAdc0, lpbuff, EngUnits, 8);

/* Clear the buffer done flags */
AL_ClearBufferDoneFlag(lhldAdc0, lpbuff->dwBuffNum);

return(0);
}
```

Visual Basic for Windows Postmessage Handler Example

```
Global lpDataBuffStat As DATABUFFSTATUS
Global DataBuff(2048) As Integer
Global EngUnits(2048) As Single

Sub MsgBlaster1_Message (MsgVal As Integer, wParam As Integer, lParam As Long, ReturnVal As Long)

  lBuffNum& = wParam

  ‘ Get the buffer status
  errnum& = AL_GetBufferStatus(gblLhld&, lpDataBuffStat, DONE_BUFFER)
  If errnum& < 0 Then End

  ' Check the status and error flags
  If lpDataBuffStat.lStatusFlags <> BUFFER_FULL Then OnBufferError(lBuffNum&)
  If lpDataBuffStat.lErrorFlags <> 0 Then OnBufferError(lBuffNum&)

  errnum& = AL_CopyBuffer(gblLhld&, lBuffNum&, DataBuff(0), 0, 10)
  If errnum& < 0 Then End
  errnum& = AL_DemuxDataSet(gblLhld&, lBuffNum&, EngUnits(0), 0, 1)
  If errnum& < 0 Then End
  errnum& = AL_ClearBufferDoneFlag(gblLhld&, lBuffNum&)
  If errnum& < 0 Then End

 ReturnVal = 0

End Sub
```

4.3 LOGICAL DEVICES

The Interface Library defines a “Logical Device Subsystem” as being a software representation of a functionally distinct subset of the hardware on a DAQ board. Each logical device deals with either Input or Output data, but not both.

Typical Logical Devices on DAQ boards include:

- ADC section (with N channels).
- DAC section (with N channels).
- Digital Input section.
- Digital Output section.
- Counter/Timers

You can also allocate multiple Logical Devices for the same piece of hardware. Since a logical device is a “software representation” of a subsection of a board, you can have more than one representation of the same subsection of a board. In other words, you can assign more than one logical device to the same piece of hardware.

It might be convenient, for example, to assign more than one logical device to an ADC subsection on a board. Each logical device could be configured entirely differently (for things like channel/gain, sample rate, buffer size, etc.) and an application program could switch between the two devices. To do this, the app would have to separately allocate the logical devices, and do the “set” functions to configure each device. Then, the first device would be initialized and started. To switch to another logical device attached to the same physical hardware, the app would stop the running device and initialize and start another.

5. PROGRAM DEVELOPMENT

5.1 INCLUDING ADLIB WIN IN C\C++

The sections presented below describe the mechanics of incorporating the ADLIB WIN library in a Windows based application program. To add the ADLIB library into the Windows Application Programming Interface (API) requires the inclusion of the following header files in your program.

1. "adlib.h" ADLIB API
2. "adlerr.h" ADLIB error codes
3. "drverr.h" ADLIB hardware generic error codes

5.1.1 Steps To Programming In C\C++

```
LHLD lhldAdc0;              /* Define a handle variable for the ADC device. */
LPBUFFSTRUCT lpbuff;        /* Define a pointer to an ADLIB data buffer structure */
ERRNUM errnum;              /* Define an error number variable. */

/* Load and initialize the environment. */
errnum = AL_LoadEnvironment("c:\adlib\alwin16\5800mfhr\examples\adlai01\adlwin16.con");

/* Allocate the ADC0 subsystem on board number 0. Reads in the current */
/* initialization file(.INI) and sets the LDSD as defined. */
lhldAdc0 = AL_AllocateDevice("ADC0", 0);
```

/* Make any changes to the LDSD before initializing the device. All LDSD parameters are either set to their initialization file (.INI) defaults when specified, otherwise each undefined parameter defaults to the first or lowest option available in the DAQ board's capability file (.CAP). If the parameter can't be found in capabilities file the option is defined as unsupported. */

```
errnum = AL_SetCycleMode(lhldAdc0, AL_SINGLE_CYCLE);
errnum = AL_SetTriggerMode(lhldAdc0,  "POST_TRIG");
errmum = AL_SetNumOfBuffers(lhldAdc0, 1);
errmum = AL_SetBufferSize(lhldAdc0, 1000);
        .
        .
/* Initializes the hardware device to the current LDSD settings. */
errnum = AL_InitDevice(lhldAdc0);

/* Start the logical device. */
errnum = AL_StartDevice(lhldAdc0);

lpbuff  = AL_GetDoneBuffPtr(lhldAdc0);
if  (NULL != lpbuff)
{
   if  (lpbuff->lErrorFlag == 0)
      DoSomethingWithBuffer(lpbuff);
   else
      HandleERROR();
}
AL_StopDevice(lhldAdc0);
AL_ReleaseDevice(lhldAdc0);
AL_ReleaseEnvironment();
```

5.2 INCLUDING ADLIB WIN IN VISUAL BASIC FOR WINDOWS

The sections presented below describe the mechanics of incorporating the ADLIB WIN library in a Visual Basic for Windows based application program. To include the ADLIB library into the Windows Application Programming Interface(API) add the following basic header files in your program MYPROG .MAK application.

1. adlib.bas ADLIB API
2. adlerr.bas ADLIB error codes
3. drverr.bas ADLIB hardware generic error codes

5.2.1 Steps to Programming in Visual Basic For Windows

```
Global gblLhld As Long                          /* Define a handle variable for the ADC device. */
Global lpDataBuffStat LPDATABUFFSTATUS          /* Define a pointer to an ADLIB data buffer
                                                   structure */

/* Load and initialize the environment. */
  errnum& = AL_LoadEnvironment("c:\adlib\alwin16\examples\vbasic\adlwin16.con")
  If errnum& < 0 Then
    MsgBox "AL_LoadEnvironment Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If

/* Allocate the ADC0 subsystem on board number 0. Reads in the current */
/* initialization file(.INI) and sets the LDSD as defined. */
  gblLhld& = AL_AllocateDevice("ADC0", BoardNum&)
  If gblLhld& < 0 Then
    MsgBox "AL_AllocateDevice Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If

/* Make any changes to the LDSD before initializing the device. All LDSD parameters are either set to their
initialization file (.INI) defaults when specified, otherwise each undefined parameter defaults to the first or
lowest option available in the DAQ board's capability file (.CAP). If the parameter can't be found in the
capabilities file the option is defined as unsupported. */

errnum& = AL_SetTriggerMode(gblLhld&, "ABOUT_TRIG")
  If errnum& < 0 Then
    MsgBox "AL_SetTriggerMode Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
    End
  End If

errnum& = AL_SetBufferSize(gblLhld&, 256)
  If errnum& < 0 Then
    MsgBox "AL_SetBufferSize Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
    End
  End If

/* Initializes the hardware device to the current LDSD settings. */
  errnum& = AL_InitDevice(gblLhld&)
  If errnum& < 0 Then
    MsgBox "AL_InitDevice Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If
```

```
/* Start the logical device. */
  errnum& = AL_StartDevice(gblLhld&)
  If errnum& < 0 Then
    MsgBox "AL_StartDevice Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
    End
  End If

  ' Collect the data and display

  errnum& = AL_GetBufferStatus(gblLhld&, lpDataBuffStat, DONE_BUFFER)
  If errnum& = 1 Then 'The buffer is avaliable
    ' Check the status and error flags
    If lpDataBuffStat.lStatusFlags <> BUFFER_FULL Then
      Msg = "Incomplete Buffer Status = "
      Msg = Msg & lpDataBuffStat.lStatusFlags
      MsgBox Msg, MB_ICONSTOP, "ADLIB Buffer Status"
      Form_Unload (1)
    End If
    If lpDataBuffStat.lErrorFlags <> 0 Then
      Msg = "Buffer Error = "
      Msg = Msg & lpDataBuffStat.lStatusFlags
      MsgBox "Buffer Error", MB_ICONSTOP, "ADLIB Buffer Error"
      Form_Unload (1)
    End If

    errnum& = AL_CopyBuffer(gblLhld&, lpDataBuffStat.lBuffNum, DataBuff(0), 0, 2048)
    If errnum& < 0 Then
    MsgBox "AL_CopyBuffer Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
    End If
    errnum& = AL_DemuxDataSet(gblLhld&, lpDataBuffStat.lBuffNum, EngUnits(0), 0, 1)
    If errnum& < 0 Then
    MsgBox "AL_DemuxDataSet Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
    End If
    errnum& = AL_ClearBufferDoneFlag(gblLhld&, lpDataBuffStat.lBuffNum)
    If errnum& < 0 Then
    MsgBox "AL_ClearBufferDoneFlag Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
    End If
    lblShowData.Caption = Format$(DataBuff(0), "0")   ' print the counts
    lblShowVolts.Caption = Format$(EngUnits(0), "0.000") + " Volts"   ' print the voltage
  ElseIf errnum& < 0 Then 'An error occured in the AL_GetBufferStatus function call
    MsgBox "AL_GetBufferStatus Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If

  errnum& = AL_StopDevice(gblLhld&)
  If errnum& < 0 Then
    MsgBox "AL_StopDevice Error", MB_ICONSTOP, "ADLIB Error"
  End If
```

```
errnum& = AL_ReleaseDevice(gblLhld&)
If errnum& < 0 Then
  MsgBox "AL_ReleaseDevice Error", MB_ICONSTOP, "ADLIB Error"
End If

errnum& = AL_ReleaseEnvironment()
If errnum& < 0 Then
  MsgBox "AL_ReleaseEnvironment Error", MB_ICONSTOP, "ADLIB Error"
End If
```

5.3 EXAMPLE PROGRAMS

Included with ADLIB WIN is an extensive set of programming examples that provide conceptual interpretations of interfacing to ADLIB to assist in the development design of the user application. Each example program is designed to exercise a specific area of the library to simplify the understanding of the example's specific concept. See also, the comments provided at the beginning of each example program file for detailed descriptions.

The following is the list of example programs:

Example programs are also provided for each specific board type, and are not necessarily generic to all boards. The examples are located in each board (NAME) directory.

\Adlai01	examines the Buffer BUFFER_DONE message response handler
\Adlai02	examines the User polling of the BUFFER_DONE flag.
\Adlai03	examines the Thermocouple temperature conversion routine
\Adlai04	examines the ABOUT Trigger mode.
\Adlai05	examines the BURST/SCAN MODE option.
\Adlai06	examines the Software Transfer method with ABOUT Trigger mode.
\Adlai07	examines the Software Transfer method.
\Adlda01	examines the Analog Output function.
\Adldio01	examines the Software Transfer Digital Input/Output functions.
\Adldio02	examines the Digital Input/Output Handshaking functions.
\Adldio03	examines the Digital Input/Output 16 bit PortResolution functions.

6. MEMORY BUFFER ALLOCATION

6.1 OVERVIEW

ADLIB WIN removes the complexity of allocating suitable DMA and Interrupt buffers by automatically allocating buffer(s) when a DAQ board's subsystem is initialized. The application programmer can then retrieve a pointer to buffer and/or set up an ADLIB buffer message handler that will be called when each buffer has been completed. ADLIB allows N buffers of size M for each DAQ board's logical device, the only limitation on quantity and size of the buffer(s) is the amount of available host system memory.

6.2 DMA SYSTEM BUFFER INSTALLATION

All buffers allocated on behalf of ADLIB for DMA Acquisition come from the available ADLIB system buffer allocated on system start up by the adacdm16.386 (WIN 3.1) or adacdm32.VXD (WIN95) virtual device driver. ADLIB NT and WDM DMA buffers are allocated during your applications device initialization, not on system Start-Up

To install adacdm16.386 or adacdm32.VXD place the following entries in the [386Enh] section of the system.ini file located in your windows/win95 directory. Note that the ADLIB setup program automatically installs adacdm16.386 or adacdm32.VXD and sets the entries accordingly.

```
device = *vdmad
device = adacdm16.386          or adacdm32.VXD for WIN95
ADACDMABUFFERSIZE = 512
```

The ADACDMABUFFERSIZE entry is specified in kilobytes.

6.3 INTERRUPT and POLLING SYSTEM BUFFERS

All buffers allocated on behalf of ADLIB for Interrupt or Polling Acquisition come from the available Windows/Win95 memory heap. The buffers are page locked into upper memory (above 1 meg.) for their duration.

6.4 THE BUFFER STRUCTURE

ADLIB WIN buffer pointers are not just pointers to a DAQ board's subsystem data, but are pointers to a structure that contains the buffer type, size, linear data address, physical data address, current sample count, hardware trigger points, completion flags, status flags, and error flags.

Some of these fields remain constant for the duration of the acquisition. Others contain relevant information about the current condition of the buffer, and error conditions that occur in the hardware device's subsystem during the current acquisition.

The three structure variables that provide state/status/error information are defined as follows:

DWORD dwDoneFlag — The Done Flag specifies the current state of a buffer. This flag can be set to any of the three following conditions as defined in the ADLIB include file.

BUFFER_IDLE = 0	The buffer is available for use, but acquisition has not started on this buffer.
BUFFER_DONE = 1	The buffer has been filled with samples and is available for use in the user application.
BUFFER_INUSE = 2	The buffer is currently being filled with samples.

DWORD dwStatusFlag The Status Flag specifies the completion status of a buffer. This flag can be set to any of the four following conditions as defined in the ADLIB include file.

BUFFER_EMPTY = 1	No samples are available in the buffer.
BUFFER_INCOMPLETE = 2	The output buffer has not read completely or and input has not filled to capacity.
BUFFER_COMPLETE = 4	The output buffer has been read completely.
BUFFER_FULL = 8	The input buffer has filled to capacity.

Beginning with Revision 3.0 of ADLIB, the upper word of the dw status flag may contain the hardware status register state at the time of an error condition as described in the lErrorFlags below.

long lErrorFlags The Error Flags specify the condition which has stopped a buffer from successfully completing. Although an error may be reported, this does not indicate the BUFFER_FULL flag is not set. This would be the case if an error was detected and enough samples were available to fill the buffer to capacity. One or more of these flags may also be set together. If any of these flags have set, checking the current hardware error conditions will usually reveal the actual condition that caused the acquisition runtime error. This flag can be set to any of the four following conditions as defined in the ADLIB include file.

BUFFER_STOPPED = 1	An error occurred and the buffer(s) are no longer being serviced.
BUFFER_OVERRUN = 2	An error occurred when attempting to place the next sample beyond the bounds of the buffer.
BUFFER_UNDERRUN = 4	An error occurred and the buffer is incomplete.
BUFFER_NEXTBUSY = 8	When attempting to access the next buffer, the current state of the dwDoneFlag indicated the buffer was not yet available for use. To release control of a buffer call the function AL_ClearBufferDoneFlag(LHLD, lBufferNum)

The Buffer structure is defined as follows in ADLIB include file.

C\C++ adlib.h file

```
typedef struct tagDATABUFFER
{
 DWORD          dwBufferType;

 DWORD          dwBufferSize; .          /* Size of each buffer in samples. */

 WORD huge *    hpwBufferLinear; .       /* App linear buffer address. */

 LPVOID         lpvDataBuffPhysical;     /* Physical address for DMA buffer. Set to NULL when */
                                         /* non DMA transfer mode. This address should not be */
                                         /* used by a user App. */

 DWORD          dwBufferWrite;           /* Number of data samples in the buffer. */
 DWORD          dwBufferRead;            /* Number of data samples obtained from the buffer. */

 WORD huge *    hpwTrigPoint;            /* App linear address of the trigger point. */
 WORD huge *    hpwTrigPointStart;       /* App linear starting address of the pre-trigger data. */
```

```
 DWORD      dwDoneFlag;          /* Buffer complete flag */
 DWORD      dwStatusFlags;       /* Current status */

 long       lErrorFlags;         /* Buffer errors */

 DWORD      dwBuffNum;           /* Buffer number */
}DATABUFFER;
typedef DATABUFFER FAR * LPBUFFERSTRUCT;
```

In addition to the above structure that provides pointers to the data, a status structure is available. This structure was originally created to provided support in Visual Basic, but can also be used in the C\C++ ADLIB API. See the ADLIB function AL_GetBufferStatus for details.

```
typedef struct tagDATABUFFSTAT
{
 DWORD      dwBufferType;        /* Data Buffer type. */

 DWORD      dwBufferSize;        /* Size of each buffer in bytes. */

 DWORD      dwBufferWrite;       /* Number of data samples in the buffer. */
 DWORD      dwBufferRead;        /* Number of data samples obtained from the buffer. */

 DWORD      dwTrigPoint;         /* App linear address of the trigger point. */
 DWORD      dwTrigPointStart;    /* App linear starting address of the */
                                 /* pre-trigger data. */

 DWORD      dwDoneFlag;          /* Buffer complete flag */
 DWORD      dwStatusFlags;       /* Current status */
 long       lErrorFlags;         /* Buffer errors */

 DWORD      dwBuffNum;           /* Buffer number ID */

}DATABUFFSTAT;
typedef DATABUFFSTAT FAR * LPDATABUFFSTATUS;
```

Visual Basic for Windows adlib.bas file

```
Type DATABUFFSTATUS

   lBufferType As Long        ' Data Buffer type.

   lBufferSize As Long        ' Size of each buffer in bytes.

   lBufferWrite As Long       ' Number of data samples in the buffer.
   lBufferRead As Long        ' Number of data samples obtained from the buffer.

   lTrigPoint As Long         ' App buffer position number of the trigger point.
   lTrigPointStart As Long    ' App buffer position number of the pre-trigger data.

   lDoneFlag As Long          ' Buffer complete flag
   lStatusFlags As Long       ' Current status
   lErrorFlags As Long        ' Buffer errors

   lBuffNum As Long           ' Buffer number ID
End Type
```

6.5 BUFFER NOTIFICATIONS

6.5.1 C\C++

The user application has three choices of obtaining a pointer to a buffer that has been completed (BUFFER_DONE).

1. The first method involves calling the ADLIB function AL_GetDoneBuffPtr(LHLD), which will return a pointer to the next completed buffer when available, otherwise this function returns NULL.

2. The second method uses Windows Post-message handler routine installed by the user application. Each time a buffer is completed the user routine will be called, passing a pointer to the buffer in the 1Param parameter.

3. The third method involves getting a pointer to a specific buffer with the ADLIB function AL_GetBuffPtr(LHLD, BufferNumber). Once obtained, this pointer is valid until a device is either released or re-initialized. The user application can then monitor the actual buffer status to determine when the buffer is complete, n samples are available, or an error occurred. The buffer still needs to be cleared with the AL_ClearBufferDoneFlag(LHLD) when running in a continuous mode of operation, otherwise an error will be indicated in the buffer structure when the driver attempts to re-use the buffer.

Example:

```
long FAR PASCAL_ export AdlibBufferProc (hwnd, message, wParam, 1Param)
        {
        LPBUFFSTRUT 1pbuff;

        lpbuff = (LPBUFFSTRUCT) 1Param;
        /*do something with the buffer*/
        }
```

See the example programs for a functional example..

6.5.2 Visual Basic for Windows

1. The first method involves calling the ADLIB function AL_GetBufferStatus(LHLD, lpDataBuffStat As DATABUFFSTATUS, lBuffNum) with the lBuffNum set to the predefined Global constant DONE_BUFFER. Specifying DONE_BUFFER fills lpDataBuffStat with the current status on the next buffer in sequence from 0 thru n that has been completed. The programmer must check the lpDataBuffStat status flags for a BUFFER_FULL flag. Once the buffer is full the AL_CopyBuffer function can be called to retrieve the data. *See example program adlai02.*

2. The second method uses Windows Post-message handler routine installed by the user application. Each time a buffer is completed the user routine will be called, passing the completed buffer number in the wParam parameter. The programmer then uses the buffer number to check the status and copy the data from the internal ADLIB buffer. *See example program adlai01.*

Example: ↓

```
  Sub MsgBlaster1_Message (MsgVal As Integer, wParam As Integer, lParam As Long, ReturnVal As
Long)

  ' Collect the data and display

  errnum& = AL_GetBufferStatus(gblLhld&, lpDataBuffStat, DONE_BUFFER)
  If errnum& < 0 Then
    MsgBox "AL_GetBufferStatus Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If

  ' Check the status and error flags
  If lpDataBuffStat.lStatusFlags <> BUFFER_FULL Then
    Msg = "Incomplete Buffer Status = "
    Msg = Msg & lpDataBuffStat.lStatusFlags
    MsgBox Msg, MB_ICONSTOP, "ADLIB Buffer Status"
    Form_Unload (1)
  End If
  If lpDataBuffStat.lErrorFlags <> 0 Then
    Msg = "Buffer Error = "
    Msg = Msg & lpDataBuffStat.lStatusFlags
    MsgBox "Buffer Error", MB_ICONSTOP, "ADLIB Buffer Error"
    Form_Unload (1)
  End If

  errnum& = AL_CopyBuffer(gblLhld&, wParam, DataBuff(0), 0, 2048)
  If errnum& < 0 Then
    MsgBox "AL_CopyBuffer Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If
  errnum& = AL_DemuxDataSet(gblLhld&, wParam, EngUnits(0), 0, 1)
  If errnum& < 0 Then
    MsgBox "AL_DemuxDataSet Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If
  errnum& = AL_ClearBufferDoneFlag(gblLhld&, wParam)
  If errnum& < 0 Then
    MsgBox "AL_ClearBufferDoneFlag Error", MB_ICONSTOP, "ADLIB Error"
    Form_Unload (1)
  End If

 End Sub
```

7. DYNAMIC LINKING TO ADLIB WIN

7.1 IMPLICIT LINKING C\C++

ADLIB WIN provides an **import library adlcore.lib** created from its DLL interface libraries. The import library contains the necessary entry points to ADLIB that the linker uses to resolve external DLL references in the user's application code. To include the import library in your application, add **adlcore.lib** to the list of libraries supplied to the linker.

7.2 EXPLICIT LINKING C\C++

The ADLIB DLL adlcore.dll can also be linked to an application by specifying the name of the ADLIB function in the IMPORTS section of your applications module definition (.DEF) file. The file adlib.h contains all function prototypes exported by ADLIB. To include an ADLIB call, proceed the ADLIB function name with ADLCORE as shown below in your .DEF file IMPORTS section.

```
IMPORTS     ADLCORE.AL_LoadEnvironment
            ADLCORE.AL_ReleaseEnvironment
                        .
                        .
                        .
```

To link with another compiler, such as Borland C++, use the LoadLibrary & GetProcAddress functions along with the ALIB imports.

7.3 VISUAL BASIC FOR WINDOWS

All available ADLIB function calls have been predefined in the file Adlib.bas file. Each function has been declared with the necessary DLL Library and function parameter type definitions. Adding the Adlib.bas file to your programs .mak file provides all necessary interfaces to ADLIB.

8. ERROR HANDLING

8.1 OVERVIEW

All Library functions return 32-bit signed numbers (long). All the positive numbers, 0 through 2,147,483,647 inclusive (0 - 0x7FFFFFFF), are non-error return values from the function.

All the negative numbers are reserved for Error codes. The first 64K error codes, -1 through -65536 inclusive (0xFFFFFFFF - 0xFFFF0000), is reserved for “universal” error codes. The rest, -65537 through -2,147,483,648 inclusive (0xFFFEFFFF - 0x80000000), are used for “special” error codes.

“Universal” error codes are those that are common to all versions of the Library, for all environments, and independent of any user code. “Special” error codes are for environment-specific messages, and for application-specific registered messages.

8.2 ERROR CONTROL CONDITIONS

Several functions are available within the ADLIB library to control certain conditions under which an error is reported from the library. The following is a list of these functions, see the specific function's definition for complete details.

AL_SetErrOnReleaseRunning
AL_GetErrOnReleaseRunning
AL_SetErrOnReInitRunning
AL_GetErrOnReInitRunning

9. ENVIRONMENT CONFIGURATIONS

9.1 AL_LoadEnvironment

Prototype **C\C++**

ERRNUM AL_LoadEnvironment(LPSTR ***lpstrEnvInfoSource***);

Visual Basic for Windows
Function AL_LoadEnvironment(ByVal ***lpstrEnvInfoSource*** As String) As Long

LPSTR *lpstrEnvInfoSource* address of the ADLIB environment filename.

The **AL_LoadEnvironment** function loads the ADLIB environment configuration file. This file contains the environment specific settings and the board(s)' related initialization information required for the proper operation of the ADLIB.

Parameter	Description
lpstrEnvInfoSource	Points to a null-terminated string that names the complete drive, path, and filename of the ADLIB configuration file (adlib.con). If this parameter does not contain the full drive and path, ADLIB searches the Windows directory.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Comments:
The application must call this function before attempting to make any ADLIB API function calls.

9.2 AL_ReleaseEnvironment

Prototype **C\C++**

ERRNUM AL_ReleaseEnvironment(void);

Visual Basic for Windows
Function AL_ReleaseEnvironment() As Long

void
no parameters

The **AL_ReleaseEnvironment** function releases all previously allocated logical device subsystem databases(LDSD), board and environment structures, and associated memory.

Parameter	Description
void	Function takes no parameters.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Comments:
Once called, all internal memory and structure pointers previously returned from ADLIB are considered invalid. Further access to these pointers will address released system memory which may cause the system to crash.

9.3 AL_SetEnvString

Prototype **C\C++**

ERRNUM AL_SetEnvString(LPSTR ***lpstrEnv***);

Visual Basic for Windows
Function AL_SetEnvString(ByVal ***lpstrEnv*** As String) As Long

LPSTR *lpstrEnv* source address

The **AL_SetEnvString** function copies the *lpstrEnv* string to the ADLIB environment. The maximum string length allowed is 80 characters, including the NULL termination character. If the input string exceeds the 80 character limit the resulting string will be truncated and NULL padded.

Parameter	Description
lpstrEnv	Points to the user define environment description string.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

9.4 AL_GetEnvString

Prototype **C\C++**

ERRNUM AL_GetEnvString(LPSTR ***lpstr***, long ***lMaxLength***);

Visual Basic for Windows

Function AL_GetEnvString(ByVal ***lpstr*** As String,
ByVal ***lMaxLength*** As Long) As Long

LPSTR *lpstrEnv* destination address
long *lMaxLength* the count in bytes to be copied

The **AL_GetEnvString** function copies *lMaxLength* bytes for the ADLIB environment string to the destination string. If the size of the destination string length is less than the string to be copied, the resulting string will be truncated and NULL padded.

Parameter	Description
lpstrEnv	Points to the input destination string.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

10. BOARD FUNCTIONS

10.1 AL_GetBoardHardwareId

Prototype **C\C++**

BOARDHWID AL_GetBoardHardwareId(BRDID ***brdid***);

Visual Basic for Windows
Function AL_GetBoardHardwareId(ByVal ***brdid*** As Long) As Double

BRDID *brdid* ID number of the board

The **AL_GetBoardHardwareId** function retrieves the board's hardware register ID number.

Parameter	Description
brdid	Specifies the ID number of the board from which the hardware ID is to be retrieved.

Returns:
On success **BOARDHWID** contains the board's hardware register ID number, otherwise **BOARDHWID** contains the negative error code that occurred during the call.

Supported Boards:
5400 Series and 5800 Series
Pci55xx Series

10.2 AL_GetBoardHardwareVersion

Prototype **C\C++**

BOARDHWVER AL_GetBoardHardwareVersion(BRDID ***brdid***);

Visual Basic for Windows
Function AL_GetBoardHardwareVersion(ByVal ***brdid*** As Long) As Double

BRDID *brdid* ID number of the board

The **AL_GetBoardHardwareVersion** function retrieves the board's hardware register version number.

Parameter	Description
brdid	Specifies the ID number of the board from which the hardware version is to be retrieved.

Returns:
On success **BOARDHWVER** contains the board's hardware register version number, otherwise **BOARDHWVER** contains the negative error code that occurred during the call.

Supported Boards:
5400 Series and 5800 Series

10.3 AL_GetBoardDriverVersion

Prototype **C\C++**

DRIVERVER AL_GetBoardDriverVersion(BRDID ***brdid***);

Visual Basic for Windows
Function AL_GetBoardDriverVersion(ByVal ***brdid*** As Long) As Double

BRDID *brdid* ID number of the board

The **AL_GetBoardDriverVersion** function retrieves the board driver's software version number.

Parameter	Description
brdid	Specifies the ID number of the board from which the board driver's software version number is to be retrieved.

Returns:
On success **DRIVERVER** contains the board driver's software version number, otherwise **DRIVERVER** contains the negative error code that occurred during the call.

Supported Boards:
All

10.4 AL_SetBoardString

Prototype **C\C++**

ERRNUM AL_SetBoardString(BRDID ***brdid,*** LPSTR ***lpstrBrd***);

Visual Basic for Windows

Function AL_SetBoardString(ByVal ***brdid*** As Long,
ByVal ***lpstrBrd*** As String) As Long

BRDID *brdid* ID number of the board
LPSTR *lpstrBrd* source address

The **AL_SetBrdString** function copies the *lpstrBrd* string to the ADLIB board string. The maximum string length allowed is 80 characters, including the NULL termination character. If the input string exceeds the 80 character limit the resulting string will be truncated and NULL padded.

Parameter	Description
brdid	Specifies the ID number of the board from which the board driver's software version number is to be retrieved.
lpstrBrd	Points to the user define board description string.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Boards:
All

10.5 AL_GetBoardString

Protype **C\C++**

ERRNUM AL_GetBoardString(**BRDID** *brdid*, LPSTR ***lpstr***, long ***lMaxLength***);

Visual Basic for Windows
Function AL_GetBoardString(ByVal ***brdid*** As Long, ByVal ***lpstr*** As String,
ByVal ***lMaxLength*** As Long) As Long

BRDID *brdid*	ID number of the board
LPSTR *lpstrBrd*	destination address
long *lMaxLength*	the count in bytes to be copied

The **AL_GetBrdString** function copies *lMaxLength* bytes for the ADLIB board string to the destination string. If the size of the destination string length is less than the string to be copied, the resulting string will be truncated and NULL padded.

Parameter	Description
brdid	Specifies the ID number of the board from which the board driver's software version number is to be retrieved.
lpstrBrd	Points to the input destination string.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Boards:
All

10.6 AL_GetBoardName

Protype **C\C++**

ERRNUM AL_GetBoardString(**BRDID** *brdid*, LPSTR ***lpstr***, long ***lMaxLength***);

Visual Basic for Windows

Function AL_GetBoardString(ByVal ***brdid*** As Long, ByVal ***lpstr*** As String, ByVal ***lMaxLength*** As Long) As Long

BRDID *brdid*	ID number of the board
LPSTR *lpstrBrd*	destination address
long *lMaxLength*	the count in bytes to be copied

The **AL_GetBrdName** function copies *lMaxLength* bytes for the ADLIB board string to the destination string. If the size of the destination string length is less than the string to be copied, the resulting string will be truncated and NULL padded.

Parameter	Description
brdid	Specifies the ID number of the board from which the board driver's software version number is to be retrieved.
lpstrBrd	Points to the input destination string.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Boards:
All

11. LOGICAL DEVICE ALLOCATIONS

11.1 AL_AllocateDevice

Prototype **C\C++**

LHLD AL_AllocateDevice(LPSTR ***lpstrLogDev***, long ***lBoardId***);

Visual Basic for Windows
Function AL_AllocateDevice(ByVal ***lpstrLogDev*** As String,
ByVal ***lBoardId*** As Long) As Long

LPSTR *lpstrLogDev* address of the device subsystem string type
long *lBoardId* ID of the board

The **AL_AllocateDevice** function loads a device subsystem into the ADLIB logical device subsystem database (LDSD). If the ADLIB information file exist (myinfo.ini) as specified in the myenv.con file, then the LDSD is configured as defined. If the ADLIB information file does **NOT** exist or a configuration parameter is not indicated within the myinfo.ini file source, the parameter will be set to its associated capabilities file default(s) if available, otherwise the parameter is set to an ADLIB internal default. See the individual function listing for its default behavior. Multiple configurations for the same hardware device subsystem can also be allocated, but only one can be put into the running state at any time.

Parameter	Description
lpstrLogDev	Points to the logical device subsystem type to be allocated. The actual string is composed of the following type identifiers (see below) along with the ID number of the device subsystem to be allocated. The available types are board dependent and are specified in the [Board] entry section of the specific board's capability file. **ADC#** Specifies a specific analog input subsystem on the board to be allocated. **DAC#** Specifies a specific analog output subsystem on the board to be allocated. **DIN#** Specifies a specific digital input subsystem on the board to be allocated. **DOT#** Specifies a specific digital output subsystem on the board to be allocated. **COUNTER#** Specifies a specific COUNTER subsystem on the board to be allocated.
lBoardId	Specifies the specific board ID from which to logical device subsystem will be allocated.

Returns:
On success returns a handle to the logical device subsystem that the application uses in all subsequent library calls to the LDSD, otherwise LHLD contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
All

11.2 AL_ReleaseDevice

Prototype **C\C++**

ERRNUM AL_ReleaseDevice(LHLD ***lhld***);

Visual Basic for Windows
Function AL_ReleaseDevice(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD to be released

The **AL_ReleaseDevice** function releases a logical device subsystem database (LDSD) and all associated memory previously allocated to the device. Further calls to the LDSD will return an invalid LHLD error code, all previously returned pointers from the LDSD to the user application that provide access to the device or its data buffers are considered invalid (NULL) and must not be used by the application program.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be released.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Notes

If the device is currently running, an error flag setting determines whether to stop and then release, or to return a non-critical error and keep the device running. See the function AL_SetErrorOnReleaseRunning for more information.

Supported Logical Device Subsystems:
All

12. LOGICAL DEVICE STRING NAMES

12.1 AL_SetLdsString

Prototype **C\C++**

ERRNUM AL_SetLdsString(LHLD ***lhld***, LPSTR ***lpstrLds***);

Visual Basic for Windows
Function AL_SetLdsString(ByVal ***lhld*** As Long, ByVal ***lpstrLds*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrLds* source address

The **AL_SetLdsString** function copies the *lpstrLds* string to the ADLIB LDSD string. The maximum string length allowed is 80 characters, including the NULL termination character. If the input string exceeds the 80 character limit the resulting string will be truncated and NULL padded.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrEnv	Points to the user define Logical device description string.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
All

12.2 AL_GetLdsString

Prototype **C\C++**

ERRNUM AL_GetLdsString(LHLD ***lhld***, LPSTR ***lpstr***, long ***lMaxLength***);

Visual Basic for Windows
Function AL_GetLdsString(ByVal ***lhld*** As Long, ByVal ***lpstr*** As String,
ByVal ***lMaxLength*** As Long) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrLds* destination address
long *lMaxLength* the count in bytes to be copied

The **AL_GetLdsString** function copies *lMaxLength* bytes for the ADLIB LDSD string to the destination string. If the size of the destination string length is less than the string to be copied, the resulting string will be truncated and NULL padded.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrLds	Points to the input destination string.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
All

12.3 AL_GetMfgProductId

Prototype **C\C++**

PRODID AL_GetBoardHardwareVersion(LHLD ***lhld****);*

Visual Basic for Windows
Function AL_GetBoardHardwareVersion(ByVal ***lhld*** As Long) As Double

BRDID *brdid* ID number of the board

The **AL_GetMfgProductId** function retrieves the board's Product ID number.

Parameter	Description
brdid	Specifies the ID number of the board from which the hardware version is to be retrieved.

Returns:

On success **PRODID** contains the board's Product ID as specified in its capability file, otherwise **PRODID** contains the negative error code that occurred during the call.

Supported Boards:
All

13. LOGICAL DEVICE INITIALIZATIONS

13.1 AL_InitDevice

Prototype **C\C++**

ERRNUM AL_InitDevice(LHLD ***lhld***);

Visual Basic for Windows
Function AL_InitDevice(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD to be initialized

The **AL_InitDevice** function initializes the actual hardware device to the current ADLIB logical device subsystem database(LDSD) settings. This function must be called before attempting to call the AL_StartDevice function, failure to do so will result in an error. Once started, all ADLIB calls that affect the functional state of the hardware device are still allowed, but the new setting is only reflected in LDSD, and **NOT** in the actual hardware device itself. ADLIB functions that control the running state are reflected in both the LDSD and actual hardware device itself. To update the actual state of the hardware device with any changes made to the LDSD, AL_InitDevice function must be called once again. This allows changes to be made to the functionality of the LDSD while the hardware device is still running. Multiple configurations for the same hardware device subsystem can also be allocated with AL_AllocateDevice, but only one can be initialized and started at any given time.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be initialized.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Note

If the device is currently running, an error flag setting determines whether to stop and then reinitialize the hardware device or to return a non-critical error and keep the device running. See the function AL_ErrorOnInitRunning for more information.

Supported Logical Device Subsystems:
All

14. LOGICAL DEVICE START / STOP OPERATIONS

14.1 AL_StartDevice

Prototype **C\C++**

ERRNUM AL_StartDevice(LHLD ***lhld***);

Visual Basic for Windows
Function AL_StartDevice(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD to be started

The **AL_StartDevice** function puts a device in the run mode. Depending on the current ADLIB transfer mode this call may not return until completed. See AL_SetDataTransferMode for more details on ADLIB's foreground and background operations.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be started.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1,

14.2 AL_StopDevice

Prototype **C\C++**

ERRNUM AL_StopDevice(LHLD ***lhld***);

Visual Basic for Windows
Function AL_StopDevice(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD to be stopped

The **AL_StopDevice** function disables the current acquisition and all logical device operations are stopped.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be stopped.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

15. LOGICAL DEVICE CYCLE MODES

15.1 AL_SetCycleMode

Prototype **C\C++**

ERRNUM AL_SetCycleMode(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetCycleMode(ByVal ***lhld*** As Long, ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the cycle mode string

The **AL_SetCycleMode** function sets the software cycle operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired cycle mode string. The available settings are defined in the ADLIB include file as follows: • AL_SINGLE_CYCLE complete acquisition and stop • AL_CONTINUOUS_CYCLE repeat acquisition

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5500HR: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5500MF: ADC0, Options: SINGLE_CYCLE
5504DA: DAC0, DAC1, DAC2, DAC3, Options: SINGLE_CYCLE
5508LC: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5508LF: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5508SCi: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5508SHR: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5508TC/BG/RTD: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5516DMA: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5525/50MF: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5532TTL, DOT0, DOT1, DOT2, DOT3, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5532TTL: DIN0, DIN1, DIN2, DIN3, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5600 Series: DIN0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
5600 Series: DIN1, DOT0-3, Options: SINGLE_CYCLE
5800 Series: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
4012AD Series: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
4112AD Series: ADC0, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE
4608ACO: DOT0, Options: SINGLE_CYCLE
4608DCO: DOT0, Options: SINGLE_CYCLE
4616ACI: DIN0, DIN1, Options: SINGLE_CYCLE
4616CCI: DIN0, DIN1, Options: SINGLE_CYCLE
4616DCI: DIN0, DIN1, Options: SINGLE_CYCLE
4616HCO: DIN0, DIN1, Options: SINGLE_CYCLE
4616OII: DIN0, DIN1, Options: SINGLE_CYCLE
4616RLY: DOT0, Options: SINGLE_CYCLE
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3, Options: SINGLE_CYCLE
Pci55xx Series: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3, Options: SINGLE_CYCLE
Pci55xx Series: CTR0, CTR1, CTR2, CTR3, Options: SINGLE_CYCLE
Pci55xx Series: ADC0, DAC0, DAC1, Options: SINGLE_CYCLE or CONTINUOUS_CYCLE

15.2 AL_GetCycleMode

Prototype **C\C++**

ERRNUM AL_GetCycleMode(LHLD ***lhld***, LPSTR ***lpstrMode***, long ***lMaxLength***);

Visual Basic for Windows

Function AL_GetCycleMode(ByVal ***lhld*** As Long, ByVal ***lpstrMode*** As String, ByVal ***lMaxLength*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrMode*	destination address of the cycle mode string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetCycleMode** function retrieves the current software cycle operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and destination string *lpstrMode* is set to either AL_SINGLE_CYCLE or AL_CONTINUOUS_CYCLE as defined in the ADLIB include file, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0, DAC1, DAC2 and DAC3
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series: DIN0, DIN1, DOT0, DOT1, DOT2, DOT3
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
4608ACO: DOT0
4608DCO: DOT0
4616ACI: DIN0, DIN1
4616CCI: DIN0, DIN1
4616DCI: DIN0, DIN1
4616HCO: DIN0, DIN1
4616OII: DIN0, DIN1
4616RLY: DOT0
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3

Pci55xx Series: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
Pci55xx Series: CTR0, CTR1, CTR2, CTR3
Pci55xx Series: ADC0, DAC0, DAC1

16. LOGICAL DEVICE TRANSFER METHODS

16.1 AL_SetDataTransferMethod

Prototype **C\C++**

ERRNUM AL_SetDataTransferMethod(LHLD ***lhld***, LPSTR ***lpstrMethod***);

Visual Basic for Windows
Function AL_SetDataTransferMethod(ByVal ***lhld*** As Long,
ByVal ***lpstrMethod*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMethod* address of the transfer method string

The **AL_SetDataTransferMethod** function sets the hardware transfer operation method of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMethod	Points to the desired transfer mode string. The available settings are defined in the ADLIB include file as follows: • AL_DTM_SOFTWARE software data transfers • AL_DTM_DMA dma data transfers • AL_DTM_IRQ interrupt only data transfers • AL_DTM_C40PORT C40 port data transfers

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5500HR: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5500MF:ADC0, Options:	SOFTWARE
5504DA:DAC0, DAC1, DAC2,DAC3, Options:	SOFTWARE
5508LC: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5508LF: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5508SCi: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5508SHR: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5508TC/BG/RTD: ADC0, Options:	SOFTWARE or INTERRUPT
5516DMA: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5525/50MF: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
5532TTL: DIN0, DIN1, DIN2, DIN3, Options:	SOFTWARE or INTERRUPT
5532TTL: DOT0, DOT1, DOT2, DOT3, Options:	SOFTWARE or INTERRUPT
5600 Series: DIN0, Options:	SOFTWARE or INTERRUPT
5600 Series: DIN1, DOT0-3, Options:	SOFTWARE
5800 Series: ADC0, Options:	SOFTWARE, DMA or INTERRUPT
4012AD Series: ADC0, Options:	SOFTWARE
4112AD Series: ADC0, Options:	SOFTWARE
4608ACO: DOT0, Options:	SOFTWARE
4608DCO: DOT0, Options:	SOFTWARE
4616ACI: DIN0, DIN1, Options:	SOFTWARE
4616CCI: DIN0, DIN1, Options:	SOFTWARE
4616DCI: DIN0, DIN1, Options:	SOFTWARE
4616HCO: DIN0, DIN1, Options:	SOFTWARE
4616OII: DIN0, DIN1, Options:	SOFTWARE
4616RLY: DOT0, Options:	SOFTWARE
4632TTL: DIN0 to DIN3, DOT0 to DOT3, Options:	SOFTWARE
Pci55xx Series: CTR0, CTR1, CTR2, CTR3, Options:	SOFTWARE
Pci55xx Series: DIN0 to DIN3, DOT0 to DOT3, Options:	SOFTWARE
Pci55xx Series: ADC0, DAC0, DAC1, Options:	SOFTWARE or DMA

16.2 AL_GetDataTransferMethod

Prototype **C\C++**

ERRNUM AL_GetDataTransferMethod(LHLD ***lhld***, LPSTR ***lpstrMethod***, long *lMaxLen);*

Visual Basic for Windows
Function AL_GetDataTransferMethod(ByVal ***lhld*** As Long,
ByVal ***lpstrMethod*** As String,
ByVal ***lLength*** As Long) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* destination address of the transfer method string
long *lMaxLength* the count in bytes to be copied

The **AL_GetDataTransferMethod** function retrieves the data transfer method of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and destination string *lpstrMethod is* set to either AL_DTM_SOFTWARE, AL_DTM_DMA, AL_DTM_IRQ or AL_DTM_C40PORT as defined in the ADLIB include file, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0, DAC1, DAC2 and DAC3
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series: DIN0, DIN1, DOT0-3
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
4608ACO: DOT0
4608DCO: DOT0
4616ACI: DIN0, DIN1
4616CCI: DIN0, DIN1
4616DCI: DIN0, DIN1
4616HCO: DIN0, DIN1
4616OII: DIN0, DIN1
4616RLY: DOT0
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
Pci55xx Series: CTR0, CTR1, CTR2, CTR3
Pci55xx Series: DIN0 to DIN3, DOT0 to DOT3
Pci55xx Series: ADC0, DAC0, DAC1

17. LOGICAL DEVICE STATUS INFORMATION

17.1 AL_GetDeviceStatus

Prototype **C\C++**

STATUS AL_GetDeviceStatus(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetDeviceStatus(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetDeviceStatus** function returns the current hardware running status of a logical device subsystem.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success **STATUS** is set to either AL_RUNNING or AL_STOPPED, otherwise **STATUS** contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0, DAC0, DAC1, DIN0, DIN1, DIN2, DIN3, DOT1 and DOT3
5500HR: ADC0, DIN0, DIN1, DOT0 and DOT1
5500MF:ADC0, DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2 and DOT3
5504DA:DAC0, DAC1, DAC2 and DAC3
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0, DIN0, DIN1, DOT0 and DOT1
5508TC/BG/RTD: ADC0
5516DMA: ADC0, DIN0, DIN1, DOT0, DOT1
5525/50MF: ADC0, DAC0, DAC1, DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2 and DOT3
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series: DIN0, DIN1, DOT0-3
5800 Series: ADC0, DAC0, DAC1, DIN0, DIN1, DIN2, DIN3, DOT1 and DOT3
4012AD Series: ADC0
4112AD Series: ADC0
4608ACO: DOT0
4608DCO: DOT0
4616ACI: DIN0, DIN1
4616CCI: DIN0, DIN1
4616DCI: DIN0, DIN1
4616HCO: DIN0, DIN1
4616OII: DIN0, DIN1
4616RLY: DOT0
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
Pci55xx Series: CTR0, CTR1, CTR2, CTR3
Pci55xx Series: DIN0 to DIN3, DOT0 to DOT3
Pci55xx Series: ADC0, DAC0, DAC1

18. ANALOG OUTPUT

18.1 AL_SetDaOutput

Prototype **C\C++**

ERRNUM AL_SetDaOutput(LHLD ***lhld***, double ***dData***, long ***lUnits***);

Visual for Basic Windows
Function AL_SetDaOutput(ByVal ***lhld*** As Long, ByVal ***dData*** As Double,
ByVal ***lUnits*** As Long) As Long

LHLD *lhld* handle of the LDSD
double *dData* output data
long *l*Units specifies the output data units

The **AL_SetDaOutput** function immediately sets a hardware analog output port to the *dData* value.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dData	Specifies the data output value.
lUnits	Sets the units in which the above dData output parameter is specified. The available settings are defined as AL_VOLTS or AL_RAWDATA in the ADLIB include file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: DAC0 and DAC1
5504DA: DAC0, DAC1, DAC2 and DAC3
5525/50MF: DAC0 and DAC1
5800 Series: DAC0 and DAC1
4412DA/CL: DAC0, DAC1, DAC2 and DAC3
Pci55xx Series: DAC0, DAC1

19. DIGITAL INPUT/OUTPUT

19.1 AL_DigInput

Prototype **C\C++**

STATUS AL_API AL_DigInput(LHLD ***lhld***, long ***lChan***, long ***lMaskBits***);

Visual Basic for Windows
Function AL_DigInput(ByVal ***lhld*** As Long, ByVal ***lChan*** As Long,
ByVal ***lMaskBits*** As Long) As Long

LHLD *lhld*	handle of the LDSD
double *lChan*	specifies the input port channel number
long *lMaskBits*	specifies the input port masking bits

The **AL_DigInput** function immediately retrieves the input status of a digital input port.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lChan	Specifies the input port channel.
lMaskBits	Selects the bits that the function will operate on. All bits set in the mask will return their current state. Unselected mask bits will be set to zero.

Returns:
On success **STATUS** contains the current port data, otherwise ERRNUM contains the negative error code that occurred during the call.

Note:
If a device does not support the channel number parameter (lChan), set lChan to zero.

Supported Logical Device Subsystems:
5400 Series: DIN0, DIN1, DIN2 and DIN3
5500HR: DIN0 and DIN1
5500MF: DIN0, DIN1, DIN2 and DIN3
5508SHR: DIN0 and DIN1
5516DMA: DIN0 and DIN1
5525/50MF: DIN0, DIN1, DIN2 and DIN3
5532TTL: DIN0, DIN1, DIN2 and DIN3
5600 Series: DIN0, DIN1, DOT0-3
5800 Series: DIN0, DIN1, DIN2 and DIN3
4608ACO: DOT0
4608DCO: DOT0
4616ACI: DIN0, DIN1
4616CCI: DIN0, DIN1
4616DCI: DIN0, DIN1
4616HCO: DIN0, DIN1
4616OII: DIN0, DIN1
4616RLY: DOT0
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
Pci55xx Series: DIN0 to DIN3, DOT0 to DOT3

Supports Channel Number Parameter:
None

19.2 AL_DigOutput

Prototype **C\C++**

ERRNUM AL_API AL_DigOutput(LHLD ***lhld***, long ***lChan***, long ***lOutputData***, long ***lMaskBits***);

Visual Basic for Windows
Function AL_DigOutput(ByVal ***lhld*** As Long, ByVal ***lChan*** As Long,
ByVal ***lOutputData*** As Long,
ByVal ***lMaskBits*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lChan* specifies the output port channel number
long *lOutputData* specifies the data to be output
long *lMaskBits* specifies the output port masking bits

The **AL_DigOutput** function immediately sets a digital output port.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lChan	Specifies the output port channel.
lOutputData	Specifies the output data to be sent to the port.
lMaskBits	Selects the bits that the function will operate on. All bits set in the mask will be acted upon. Unselected bits in the mask will not change state.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the negative error code that occurred during the call.

Note:
If a device does not support the channel number parameter (lChan), set lChan to zero.

Supported Logical Device Subsystems:
5400 Series: DOT1 and DOT3
5500HR: DOT0 and DOT1
5500MF: DOT0, DOT1, DOT2 and DOT3
5508SHR: DOT0 and DOT1
5516DMA: DOT0 and DOT1
5525/50MF: DOT0, DOT1, DOT2 and DOT3
5532TTL: DOT0, DOT1, DOT2 and DOT3
5600 Series: DOT0-3
5800 Series: DOT1 and DOT3
4608ACO: DOT0
4608DCO: DOT0
4616RLY: DOT0
4616HCO: DOT0, DOT1
4632TTL: DOT0, DOT1, DOT2, DOT3
Pci55xx Series: DIN0 to DIN3, DOT0 to DOT3

Supports Channel Number Parameter:
None

19.3 AL_DigBitsTest

Prototype **C\C++**

STATUS AL_API AL_DigBitsTest(LHLD ***lhld***, long ***lChan***, long ***lTestBits***);

Visual Basic for Windows
Function AL_DigBitsTest(ByVal ***lhld*** As Long, ByVal ***lChan*** As Long,
ByVal ***lTestBits*** As Long) As Long

LHLD *lhld*	handle of the LDSD
double *lChan*	specifies the input port channel number
long *lTestBits*	specifies the input port bits to test

The **AL_DigBitsTest** function immediately retrieves input status of a digital input port.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lChan	Specifies the input port channel.
lTestBits	Selects the bits that the function will operate on. All bits set in the *lTestBits* will be checked for a true (1) condition.

Returns:
On success **STATUS** is set to one (1) if all of the *lTestBits* are true and set to zero (0) if any *lTestBits* are false, otherwise **STATUS** contains the negative error code that occurred during the call.

Note:
If a device does not support the channel number parameter (lChan), set lChan to zero.

Supported Logical Device Subsystems:
5400 Series: DIN0, DIN1, DIN2 and DIN3
5500HR: DIN0 and DIN1
5500MF: DIN0, DIN1, DIN2 and DIN3
5508SHR: DIN0 and DIN1
5516DMA: DIN0 and DIN1
5525/50MF: DIN0, DIN1, DIN2 and DIN3
5532TTL: DIN0, DIN1, DIN2 and DIN3
5600 Series: DIN0, DIN1, DOT0-3
5800 Series: DIN0, DIN1, DIN2 and DIN3
4608ACO: DOT0
4608DCO: DOT0
4616ACI: DIN0, DIN1
4616CCI: DIN0, DIN1
4616DCI: DIN0, DIN1
4616HCO: DIN0, DIN1
4616OII: DIN0, DIN1
4616RLY: DOT0
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
Pci55xx Series: DIN0 to DIN3, DOT0 to DOT3

Supports Channel Number Parameter:
None

19.4 AL_SetPortResolution

Prototype **C\C++**

ERRNUM AL_API AL_SetPortResolution(LHLD ***lhld***, LPSTR ***lpstrPortRes***);

Visual Basic for Windows

Function AL_SetPortResolution (ByVal ***lhld*** As Long,
ByVal ***lpstrPortRes*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrPortRes* specifies the input or output port resolution.

The **AL_SetPortResolution** function specifies whether input or output port is read or written in 4, 8 or 16 bit transfers.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
LpstrPortRes	Specifies 4, 8 or 16 bit port resolution.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the negative error code that occurred during the call.

Supported Logical Device Subsystems:

5532TTL: DIN0 and DIN2	either 8 or 16 bit
5600 Series/ACI, ADO, DCI, DCO, CCI, TTL: DIN0 and DIN2	either 8 or 16 bit
4616ACI: DIN0	either 8 or 16 bit
4616DCI: DIN0	either 8 or 16 bit
4616CCI: DIN0	either 8 or 16 bit
4616OII: DIN0	either 8 or 16 bit
4616HCO: DIN0	either 8 or 16 bit
4632TTL: DIN0, DIN2, DOT0, DOT2	either 8 or 16 bit

19.5 AL_SetPortMask

Prototype **C\C++**

ERRNUM AL_API AL_SetPortMask(LHLD ***lhld***, long ***lPortMask***);

Visual Basic for Windows
Function AL_SetPortMask (ByVal ***lhld*** As Long,
ByVal ***lPortMask*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lPortMask* specifies the input or output port masking bits.

The **AL_SetPortMask** function specifies the input or output port bit(s) that are to be acted upon during **interrupt** or **DMA** background operation.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lPortMask	Specifies the bit(s) to be acted upon.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the negative error code that occurred during the call.

Supported Logical Device Subsystems:
5532TTL:
5600 Series ACI, DCI, CCI, TTL: DIN0

19.6 AL_SetPortStruct

Prototype **C\C++**

ERRNUM AL_SetPortStruct(LHLD ***lhld***, LPPORT ***lpportStruct***);

Visual Basic for Windows

Function AL_SetPortStruct(ByVal ***lhld*** As Long, ***lpportStruct*** As PORT) As Long

LHLD *lhld* handle of the LDSD
LPPORT *lpportStruct* address of the user port structure

The **AL_SetPortStruct** function sets the LDSD digitl I/O port settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpportStruct	Specifies a 32 bit far pointer to a port information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagPORT
{
 char    achPortResName[MAX_OPTIONS_NAME_STR];   //Device string name, 4, 8 or 16
 long    lPortResId;                             //Device port resolution 4, 8 or 16 bit
 long    lPortMask;                              //Device port masking bits

 // Special digital input and output control features.
 LPVOID  lpvPortControl;                         //Set to NULL if not used by a device
                                                 //Currently the 5600 Digital cards
                                                 //are the only devices that support
                                                 //lpvPortControl.

}PORT;
typedef PORT FAR * LPPORT;
```

NOTE: achPortResName and lPortResId both represent the ports resolution. If lportResId is set to 0 the achPortResName string is used to set the port's resolution, otherwise lPortResId is used.

```
typedef struct tag5600PORT
{// Special digital input control for the 5600 series

 long    lDebounce;                  //DISABLED, ENABLED
 long    lPortModeSpecification;     //DISABLED, AND, OR
 long    lPatternLatch;              //DISABLED, ENABLED
 long    lDataPathPolarity;          //DataPathPolarity allows each bit for PortA
                                     //to be individualy inverted.
                                     //If the associated bit is set "1", data read
                                     //from that bit on PortA will be inverted. For
                                     //detailed definitions see the 5600 manuals
                                     //ADVANCED PROGRAMMING
                                     //TECHNIQUES. When the
                                     //PortResolution=16,  PatternPolarity,
                                     //PatternTransition and PatternMask are set
                                     //in WORDs (0xFFFF).
```

```
long        lSpecialIoControl;          //The SpecialIoControl is used for BIT
                                        // CATCHER operation mode. If the port is
                                        //configured for InputConfig =
                                        //BIT_CATCHER and the associated
                                        //bit is set "1", the port will return "0" for
                                        //that bit until a "1" occurs at the input. The
                                        //port will then read back a "1" even if the
                                        //"1" goes away. When ADLIB reads the
                                        //port, each bit enabled and in the "1" will
                                        //automatically be reset and ready for the
                                        //next input transition. For detailed
                                        //definitions see the 5600 manuals
                                        //ADVANCED PROGRAMMING
                                        //TECHNIQUES. When the
                                        //PortResolution=16, PatternPolarity,
                                        //PatternTransition and PatternMask
                                        //are set in WORDs (0xFFFF).
 long       lPatternPolarity;           //The PatternPolarity, PatternTransition and
                                        //PatternMask define the match condition.
 Long       lPatternTransition;         //for the PatternModeSpec AND / OR
                                        //modes.
 long       lPatternMask;               //For detailed definitions see the 5600
                                        //manuals ADVANCED PROGRAMMING
                                        //TECHNIQUES. When the
                                        //PortResolution=16, PatternPolarity,
                                        //PatternTransition and PatternMask
                                        //are set in WORDs (0xFFFF).

 long       lIgnoreIpError;             //Ignore subsequent match conditions
 long       lTimeConstant;              //Debounce TimeConstant =
                                        //period(usec) / 2(usec) 1000us / 2us = 500
                                        //Valid 2 to 65535, a TimeConstant of
                                        //0=65536, whole numbers only
}A5600PORT;
typedef A5600PORT FAR * LP5600PORT;
```

Returns:
On success ERRNUM is set to 1 and the lpportSruct structure is filled with the device's digital I/O port settings, otherwise ERRNUM contains the last error code that occurred during the call and the lpportStruct structure may contain invalid information.

Supported Logical Device Subsystems:
5532TTL:
5600 Series ACI, DCI, CCI, TTL: DIN0

19.7 AL_GetPortStruct

Prototype **C\C++**

ERRNUM AL_GetPortStruct(LHLD ***lhld***, LPPORT ***lpportStruct***);

Visual Basic for Windows

Function AL_GetPortStruct(ByVal ***lhld*** As Long, ***lpportStruct*** As PORT) As Long

LHLD *lhld* handle of the LDSD
LPPORT *lpportStruct* address of the user port structure

The **AL_GetPortStruct** function provides access to LDSD digitl I/O port settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpportStruct	Specifies a 32 bit far pointer to a port information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagPORT
{
 char    achPortResName[MAX_OPTIONS_NAME_STR];
 long            lPortResId;

 long            lPortMask;

 // Special digital input and output control features.
 LPVOID          lpvPortControl;

}PORT;
typedef PORT FAR * LPPORT;


typedef struct tag5600PORT
{// Special digital input control for the 5600 series

 long            lDebounce;                  //DISABLED, ENABLED
 long            lPortModeSpecification;     //DISABLED, AND, OR
 long            lPatternLatch;              //DISABLED, ENABLED
 long            lDataPathPolarity;          //DataPathPolarity allows each bit for PortA
                                             //to be individualy inverted.
                                             //If the associated bit is set "1", data read
                                             //from that bit on PortA will be inverted. For
                                             //detailed definitions see the 5600 manuals
                                             //ADVANCED PROGRAMMING
                                             //TECHNIQUES. When the
                                             //PortResolution=16, PatternPolarity,
                                             //PatternTransition and PatternMask are set
                                             //in WORDs (0xFFFF).
```

```
long        lSpecialIoControl;      //The SpecialIoControl is used for BIT
                                    // CATCHER operation mode. If the port is
                                    //configured for InputConfig =
                                    //BIT_CATCHER and the associated
                                    //bit is set "1", the port will return "0" for
                                    //that bit until a "1" occurs at the input. The
                                    //port will then read back a "1" even if the
                                    //"1" goes away. When ADLIB reads the
                                    //port, each bit enabled and in the "1" will
                                    //automatically be reset and ready for the
                                    //next input transition. For detailed
                                    //definitions see the 5600 manuals
                                    //ADVANCED PROGRAMMING
                                    //TECHNIQUES. When the
                                    //PortResolution=16, PatternPolarity,
                                    //PatternTransition and PatternMask
                                    //are set in WORDs (0xFFFF).
long        lPatternPolarity;       //The PatternPolarity, PatternTransition and
                                    //PatternMask define the match condition.
Long        lPatternTransition;     //for the PatternModeSpec AND / OR
                                    //modes.
long        lPatternMask;           //For detailed definitions see the 5600
                                    //manuals ADVANCED PROGRAMMING
                                    //TECHNIQUES. When the
                                    //PortResolution=16, PatternPolarity,
                                    //PatternTransition and PatternMask
                                    //are set in WORDs (0xFFFF).
long        lIgnoreIpError;         //Ignore subsequent match conditions
long        lTimeConstant;          //Debounce TimeConstant =
                                    //period(usec) / 2(usec) 1000us / 2us = 500
                                    //Valid 2 to 65535, a TimeConstant of
                                    //0=65536, whole numbers only
}A5600PORT;
typedef A5600PORT FAR * LP5600PORT;
```

Returns:
On success ERRNUM is set to 1 and the lpportSruct structure is filled with the device's digital I/O port settings, otherwise ERRNUM contains the last error code that occurred during the call and the lpportStruct structure may contain invalid information.

Supported Logical Device Subsystems:
5532TTL:
5600 Series ACI, DCI, CCI and TTL: DIN0

20. TRIGGERING MODES

20.1 AL_SetTriggerMode

Prototype **C\C++**

ERRNUM AL_SetTriggerMode(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetTriggerMode(ByVal ***lhld*** As Long,
ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the trigger mode string

The **AL_SetTriggerMode** function sets the hardware trigger operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired trigger mode string. The available string settings are device independent and are verified by ADLIB against the available trigger mode options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	DISABLED, POST_TRIG, PRE_TRIG or ABOUT_TRIG
5500HR: ADC0, Options	DISABLED or SCAN_TRIG
5508LC: ADC0, Options	DISABLED or SCAN_TRIG
5508LF: ADC0, Options	DISABLED or SCAN_TRIG
5508SCi: ADC0, Options	DISABLED or SCAN_TRIG
5508SHR: ADC0, Options	DISABLED or SCAN_TRIG
5508TC/BG/RTD: ADC0, Options	DISABLED or SCAN_TRIG
5800 Series: ADC0, Options:	DISABLED, POST_TRIG, PRE_TRIG or ABOUT_TRIG
Pci55xx Series: ADC0, Options:	DISABLED, POST_TRIG, PRE_TRIG or ABOUT_TRIG
Pci55xx Series: DAC0, Options:	DISABLED, POST_TRIG

20.2 AL_SetTriggerSource

Prototype **C\C++**

ERRNUM AL_SetTriggerSource(LHLD ***lhld***, LPSTR ***lpstrSource***);

Visual Basic for Windows
Function AL_SetTriggerSource(ByVal ***lhld*** As Long,
ByVal ***lpstrSource*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrSource* address of the trigger source string

The **AL_SetTriggerSource** function sets the hardware trigger source of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSource	Points to the desired trigger source string. The available string settings are device independent and are verified by ADLIB against the available trigger source options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	DISABLED or EXTERNAL
5500HR: ADC0, Options	DISABLED or CTR2
5508LC: ADC0, Options	DISABLED, EXTERNAL or CTR1
5508LF: ADC0, Options	DISABLED, EXTERNAL or CTR1
5508SCi: ADC0, Options	DISABLED, EXTERNAL or CTR1
5508SHR: ADC0, Options	DISABLED or CTR2
5525/50MF: ADC0, Options:	DISABLED or CTR2
5800 Series: ADC0, Options:	DISABLED or EXTERNAL
Pci55xx Series: ADC0, DAC0, Options:	DISABLED or EXTERNAL

20.3 AL_SetTrigSourceSignal

Prototype **C\C++**

ERRNUM AL_SetTrigSourceSignal(LHLD ***lhld***, LPSTR ***lpstrSourceSignal***);

Visual Basic for Windows
Function AL_SetTrigSourceSignal(ByVal ***lhld*** As Long,
ByVal ***lpstrSourceSignal*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrSourceSignal* address of the trigger source signal string

The **AL_SetTrigSourceSignal** function sets the hardware trigger source signal of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSourceSignal	Points to the desired trigger source signal string. The available string settings are device independent and are verified by ADLIB against the available trigger source signal options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	FALLING_EDGE or RISING_EDGE
5800 Series: ADC0, Options:	FALLING_EDGE or RISING_EDGE
Pci55xx Series: ADC0, DAC0, Options:	FALLING_EDGE or RISING_EDGE

20.4 AL_SetTriggerRate

Prototype **C\C++**

ERRNUM AL_SetTriggerRate(LHLD ***lhld***, double ***dRate***, long ***lUnits***);

Visual Basic for Windows
Function AL_SetTriggerRate(ByVal ***lhld*** As Long, ByVal ***dRate*** As Double,
ByVal ***lUnits*** As Long) As Long

LHLD *lhld*	handle of the LDSD
double *dRate*	specifies the rate of triggers
long *l*Units	specifies the rate units

The **AL_SetTriggerRate** function sets the hardware trigger rate of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dRate	Specifies the rate at which a trigger will occur.
lUnits	Sets the units in which the above rate parameter is specified. The available settings are defined as HERTZ or TICS in the ADLIB include file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5500HR: ADC0, Options:	15.259 to 15000
5508LC: ADC0, Options:	15.259 to 100000
5508LF: ADC0, Options:	15.259 to 100000
5508SCi: ADC0, Options:	15.259 to 100000
5508SHR: ADC0, Options:	61.036 to 47000

20.5 AL_SetPostSampleCount

Prototype **C\C++**

ERRNUM AL_SetPostSampleCount(LHLD ***lhld***, long ***lCount***);

Visual Basic for Windows
Function AL_SetPostSampleCount(ByVal ***lhld*** As Long,
ByVal ***lCount*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lCount* specifies the number of sample counts

The **AL_SetPostSampleCount** function sets the post trigger samples in the LDSD. This function is made available for hardware devices that provide a post trigger counting mechanism. When the hardware device receives a trigger, it provides (n) more samples, then stops. The post trigger samples specifies the number of data samples to be obtained after a valid hardware trigger has been received.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lCount	Sets the number of samples to be obtained after a trigger has been received. The available ranges are device dependent and are verified by ADLIB against the min./max. range specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	0 to 65535	used for PRE_TRIG or ABOUT_TRIG Triggermode
5800 Series: ADC0, Options:	0 to 65535	used for PRE_TRIG or ABOUT_TRIG Triggermode
Pci55xx Series: ADC0, Options:	0 to 65535	used for PRE_TRIG or ABOUT_TRIG Triggermode

20.6 AL_GetTriggerStruct

Prototype **C\C++**

ERRNUM AL_GetTriggerStruct(LHLD ***lhld***, LPTRIG ***lptrigStruct***);

Visual Basic for Windows
Function AL_GetTriggerStruct(ByVal ***lhld*** As Long,
lptrigStruct As TRIGGER) As Long

LHLD *lhld* handle of the LDSD
LPTRIG *lptrigStruct* address of the user trigger structure

The **AL_GetTriggerStruct** function provides access to all LDSD trigger settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lptrigStruct	Specifies a 32 bit far pointer to a trigger information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagTRIGGER
{
        char    achTrigModeName[MAX_OPTIONS_NAME_STRING];
        long    lTrigModeId;

        char    achTrigSrcName[MAX_OPTIONS_NAME_STRING];
        long    lTrigSrcId;

        char    achTrigSrcSignalName[MAX_OPTIONS_NAME_STRING];
        long    lTrigSrcSignalId;

        long    lTrigRateUnits;
        double  dTrigRate;

        long    lPostSampleCount;

}TRIGGER;
typedef TRIGGER FAR * LPTRIG;
```

Returns:
On success ERRNUM is set to 1 and the lptrigStruct structure is filled with the device's triggering settings, otherwise ERRNUM contains the last error code that occurred during the call and the lptrigStruct structure may contain invalid information.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
Pci55xx Series: ADC0, DAC0

20.7 AL_SetTriggerOutput

Prototype **C\C++**

ERRNUM AL_SetTriggerOutput(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetTriggerOutput(ByVal ***lhld*** As Long,
ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the TriggerOutput mode string

The **AL_SetTriggerOutput** function sets the hardware TriggerOutput operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired TriggerOutput mode string. The available string settings are device independent and are verified by ADLIB against the available trigger mode options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx Series: ADC0, Options: ENABLED, DISABLED

20.8 AL_GetTriggerOutput

Prototype **C\C++**

ERRNUM AL_GetTriggerOutput(LHLD ***lhld***, LPSTR ***lpstrTriggerOutput***, long ***Maxlen***);

Visual Basic for Windows

Function AL_GetTriggerOutput(ByVal ***lhld*** As Long, ByVal ***lpstrTriggerOutput*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrTriggerOutput*	destination address of the TriggerOutput string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataCode** function retrieves the current TriggerOutput mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrTriggerOutput	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:

On success ERRNUM is set to 1 and the destination string *lpstrTriggerOutput* is set to the LDSD TriggerOutput mode, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:

Pci55xx Series: ADC0

21. CLOCKING MODES

21.1 AL_SetClockSource

Prototype **C\C++**

ERRNUM AL_SetClockSource(LHLD ***lhld***, LPSTR ***lpstrSource***);

Visual Basic for Windows
Function AL_SetClockSource(ByVal ***lhld*** As Long,
ByVal ***lpstrSource*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrSource* address of the clock source string

The **AL_SetClockSource** function sets the hardware clocking source of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSource	Points to the desired clock source string. The available string settings are device independent and are verified by ADLIB against the available clock source options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, EXT_RISING_EDGE or EXT FALLING_EDGE
5500HR: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5508LC: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5508LF: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5508SCi: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5508SHR: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5508TC/BG/RTD:ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5516DMA: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5525/50MF: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, or EXT FALLING_EDGE
5800 Series: ADC0, Options:	SOFTWARE_CONVERT, INTERNAL, EXT_RISING_EDGE or EXT FALLING_EDGE
4012AD Series: ADC0 Options:	SOFTWARE_CONVERT
4112AD Series: ADC0 Options:	SOFTWARE_CONVERT
5500MF: ADC0, Options:	SOFTWARE_CONVERT
Pci55xx Series: ADC0, Dac0, Dac1 Options:	SOFTWARE_CONVERT, INTERNAL, EXT_RISING_EDGE or EXT FALLING_EDGE

21.2 AL_SetClkSourceSignal

Prototype **C\C++**

ERRNUM AL_SetClkSourceSignal(LHLD ***lhld***, LPSTR ***lpstrSourceSignal***);

Visual Basic for Windows

Function AL_SetClkSourceSignal(ByVal ***lhld*** As Long,
ByVal ***lpstrSourceSignal*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrSourceSignal* address of the clocking source signal string

The **AL_SetClockSourceSignal** function sets the hardware clocking source Signal of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSourceSignal	Points to the desired clocking source signal string. The available string settings are device independent and are verified by ADLIB against the available clock source signal options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
NONE

21.3 AL_SetClockRate

Prototype **C\C++**

ERRNUM AL_SetClockRate(LHLD ***lhld***, double ***dRate***, long ***lUnits***);

Visual Basic for Windows

Function AL_SetClockRate(ByVal ***lhld*** As Long, ByVal ***dRate*** As Double,
ByVal ***lUnits*** As Long) As Long

LHLD *lhld*	handle of the LDSD
double *dRate*	specifies the rate of the clock
long *l*Units	specifies the rate units

The **AL_SetClockRate** function sets the hardware clocking rate of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dRate	Specifies the rate at which a clocking will occur.
lUnits	Sets the units in which the above rate parameter is specified. The available settings are defined as HERTZ TICS in the ADLIB include file. The 4012AD and 4112AD also support MSECONDS (miliseconds) in the Windows 95 or 98 environment.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5401, 5402, 5801 and 5802: ADC0, Options:	1.164188e-3 to 333000
5403, 5404, 5803 and 5804: ADC0, Options:	1.164188e-3 to 100000
5500HR: ADC0, Options:	15.259 to 15000
5508LC: ADC0, Options:	15.259 to 100000
5508LF: ADC0, Options:	15.259 to 100000
5508SCi: ADC0, Options:	15.259 to 100000
5508SHR: ADC0, Options:	61.036 to 47000
5516DMA: ADC0, Options:	15.259 to 60000
5525/50MF: ADC0, Options:	15.259 to 40000
4012AD: ADC0, Options:	1 to 65535 miliseconds
4112AD: ADC0, Options:	1 to 65535 miliseconds
Pci5500, 5501, 5502: ADC0, Options:	0.23283e-3 to 100000
Pci5500, 5501, 5502: DAC0, DAC1, Options:	0.23283e -3 to 200000
Pci5503, 5504: ADC0, Options:	0.23283e -3 to 200000
Pci5503, 5504, DAC0, DAC1, Options:	0.23283e -3 to 200000

21.4 AL_GetActualClkRate

Prototype **C\C++**

(32 bit) ERRNUM AL_GetActualClkRate(LHLD ***lhld***, double ****dActualRate***);

Visual Basic for Windows

Function AL_GetActualClkRate(ByVal ***lhld*** As Long, ByVal ***dActaulRate*** As Double) As Long

LHLD *lhld* handle of the LDSD

double *dActualRate* points to a double value to receiving the value.

The **AL_GetActualClkRate** function is used to determine the frequency actually available by a device's clocking hardware. The clock frequency specified in the AL_SetClockRate.vi function may not be achievable due to hardware limitations.

Note: The AL_SetClockRate clocking rate parameters are not programmed to a device until it initialized with AL_InitializeDevice. This function uses the clocking rate parameters from the last call to the AL_SetClockRate function to determine the actual available hardware rate and does not reflect the current devices programmed rate until the device has been initialized. Once a device has been initialized, if the AL_SetClockRate is called again with different clocking rate parameters this function will not return the current device's programmed clocking frequency until the device is re-initialized. Therefore this function only returns the actual rate that can be set by the AL_SetClockRate function. To get the current Clocking Rate programmed to the device, it must be initialized before calling this function.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dActualRate	Contains the actual hardware clocking rate.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5401, 5402, 5801 and 5802: ADC0
5403, 5404, 5803 and 5804: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5516DMA: ADC0
5525/50MF: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

21.5 AL_MaxClkRate

Prototype **C\C++**

(32 bit) ERRNUM AL_GetMaxClkRate(LHLD ***lhld***, double ****dMaxRate***);

Visual Basic for Windows

Function AL_GetMaxClkRate(ByVal ***lhld*** As Long, ByVal ***dMaxRate*** As Double) As Long

LHLD *lhld* handle of the LDSD

double *dMaxRate* points to a double value to receiving the value.

The **AL_GetMaxClkRate** function is used to determine the Maximum frequency actually available by a device's clocking hardware.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dMaxRate	Contains the Maximum hardware clocking rate.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
PCI-550X Series
5401, 5402, 5801 and 5802: ADC0
5403, 5404, 5803 and 5804: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5516DMA: ADC0
5525/50MF: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

21.6

21.7 AL_GetClockStruct

Prototype **C\C++**

ERRNUM AL_GetClockStruct(LHLD ***lhld***, LPCLK ***lpclkStruct***);

Visual Basic for Windows
Function AL_GetClockStruct(ByVal ***lhld*** As Long, ***lpclkStruct*** As CLOCK) As Long

LHLD *lhld* handle of the LDSD
LPTRIG *lpclkStruct* address of the user clock structure

The **AL_GetClockStruct** function provides access to all LDSD clock settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpclkStruct	Specifies a 32 bit far pointer to a clocking information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagCLOCK
{
        char    achClkSourceName[MAX_OPTIONS_NAME_STRING];
        long    lClkSourceId;

        char    achClkSrcSignalName[MAX_OPTIONS_NAME_STRING];
        long    lClkSrcSignalId;

        double  dClkRate;
        long    lClkRateUnits;

}CLOCK;
typedef  CLOCK FAR * LPCLK;
```

Returns:
On success ERRNUM is set to 1 and the lpclkStruct structure is filled with the device's clocking settings, otherwise ERRNUM contains the last error code that occurred during the call and the lpclkStruct structure may contain invalid information.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD: ADC0
4112AD: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

21.8 AL_SetClockOutput

Prototype **C\C++**

ERRNUM AL_SetClockOutput(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetClockOutput(ByVal ***lhld*** As Long,
ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the ClockOutput mode string

The **AL_SetClockOutput** function sets the hardware ClockOutput operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired ClockOutput string. The available string settings are device independent and are verified by ADLIB against the available clock output mode options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx Series: ADC0, Options: ENABLED, DISABLED

21.9 AL_GetClockOutput

Prototype **C\C++**

ERRNUM AL_GetClockOutput(LHLD ***lhld***, LPSTR ***lpstrClockOutput***, long ***lMaxlen***);

Visual Basic for Windows

Function AL_GetClockOutput(ByVal ***lhld*** As Long, ByVal ***lpstrClockOutput*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrClockOutput*	destination address of the ClockOutput string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetClockOutput** function retrieves the current ClockOutput state of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrClockOutput	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:

On success ERRNUM is set to 1 and the destination string *lpstrClockOutput* is set to the LDSD ClockOutput state, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:

Pci55xx Series: ADC0

22. GATING MODES

22.1 AL_SetGateSource

Prototype **C\C++**

ERRNUM AL_SetGateSource(LHLD ***lhld***, LPSTR ***lpstrSource***);

Visual Basic for Windows
Function AL_SetGateSource(ByVal ***lhld*** As Long,
ByVal ***lpstrSource*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrSource* address of the gate source string

The **AL_SetGateSource** function sets the hardware gate source of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSource	Points to the desired gate source string. The available string settings are device independent and are verified by ADLIB against the available gate source options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	DISABLED, SWGATE or EXTGATE
5800 Series: ADC0, Options:	DISABLED, SWGATE or EXTGATE
Pci55xx Series: ADC0, DAC0, Options:	DISABLED, SWGATE or EXTGATE

22.2 AL_SetGateLevel

Prototype **C\C++**

ERRNUM AL_SetGateLevel(LHLD ***lhld***, LPSTR ***lpstrGateLevel***);

Visual Basic for Windows
Function AL_SetGateLevel(ByVal ***lhld*** As Long,
ByVal ***lpstrGateLevel*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrGateLevel* address of the gate Level string

The **AL_SetGateLevel** function sets the hardware gate level of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrGateLevel	Points to the desired gate level string. The available string settings are device independent and are verified by ADLIB against the available gate source options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	ACTIVE_HIGH or ACTIVE_LOW
5800 Series: ADC0, Options:	ACTIVE_HIGH or ACTIVE_LOW
Pci55xx Series: ADC0, DAC0, Options:	ACTIVE_HIGH or ACTIVE_LOW

22.3 AL_GetGateStruct

Prototype **C\C++**

ERRNUM AL_GetGateStruct(LHLD ***lhld***, LPGATE ***lpgateStruct***);

Visual Basic for Windows
Function AL_GetGateStruct(ByVal ***lhld*** As Long, ***lpgateStruct*** As GATE) As Long

LHLD *lhld* handle of the LDSD
LPGATE *lpgateStruct* address of the user gate structure

The **AL_GetGateStruct** function provides access to all LDSD gate settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpgateStruct	Specifies a 32 bit far pointer to a gate information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagGATE
{
        char    achGateSourceName[MAX_OPTIONS_NAME_STRING];
        long    lGateSourceId;

        long    lGateLevel;

}GATE;
typedef GATE FAR * LPGATE;
```

Returns:
On success ERRNUM is set to 1 and the *lpgateStruct* structure is filled with the device's gate settings, otherwise ERRNUM contains the last error code that occurred during the call and the *lpgateStruct* structure may contain invalid information.

Supported Logical Device Subsystems:
5400 Series: ADC0
5800 Series: ADC0
Pci55xx Series: ADC0, DAC0

22.4 AL_SetSwGate

Prototype **C\C++**

ERRNUM AL_SetSwGate(LHLD ***lhld***, long ***lSwGate***);

Visual Basic for Windows
Function AL_SetSwGate(ByVal ***lhld*** As Long, ByVal ***lSwGate*** As Long) As Long

LHLD *lhld* handle of the LDSD
long lSwGate specifies the desired state of the hardware gate

The **AL_SetSwGate** function enables or disables a hardware control gate.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lSwGate	Specifies the hardware gate state. The available settings are defined in the ADLIB include file as follows: • DISABLED The device is stopped. • ENABLED The device is enabled to run.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	DISABLED or ENABLED
5800 Series: ADC0, Options:	DISABLED or ENABLED
Pci55xx Series: ADC0,DAC0, Options:	DISABLED or ENABLED

22.5 AL_GetSwGate

Prototype **C\C++**

STATUS AL_GetSwGate(LHLD **lhld**);

Visual Basic for Windows
Function AL_GetSwGateStatus(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetSwGate** function retrieves the current hardware control gate status.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success STATUS is set to either DISABLED or ENABLED, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5800 Series: ADC0
Pci55xx Series: ADC0, DAC0

23. BURST MODES

23.1 AL_SetBurstMode

Prototype **C\C++**

ERRNUM AL_SetBurstMode(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetBurstMode(ByVal ***lhld*** As Long, ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the Burst mode string

The **AL_SetBurstMode** function sets the hardware Burst operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired Burst mode string. The available string settings are device independent and are verified by ADLIB against the available Burst mode options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0, Options: DISABLED or ENABLED
5525/50MF: ADC0, Options: DISABLED or ENABLED
5800 Series: ADC0, Options: DISABLED or ENABLED
Pci55xx Series: ADC0, Options: DISABLED or ENABLED

23.2 AL_SetBurstLength

Prototype **C\C++**

ERRNUM AL_SetBurstLength(LHLD ***lhld***, long ***lBurstSamples***);

Visual Basic for Windows
Function AL_SetBurstLength(ByVal ***lhld*** As Long,
ByVal ***lBurstSamples*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lBurstSamples* specifies the number of burst samples

The AL_**SetBurstLength** function sets the number of samples obtained during a burst in the LDSD. This function is made available for hardware devices that provide burst length mechanism. When the hardware device starts a burst, it provides (n) more sample, then stops. The Burst samples specify the number of data samples to be obtained after a valid hardware burst trigger.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBurstSamples	Sets the number of samples to be obtained after a Burst has been received. The available range is device independent and is verified by ADLIB against the min./max. burst length specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0, Options: 1 to 256
5800 Series: ADC0, Options: 1 to 256
Pci55xx Series: ADC0, Options: 1 to 65535

23.3 AL_SetBurstRate

Prototype **C\C++**

ERRNUM AL_SetBurstRate(LHLD ***lhld***, double ***dRate***, long ***lUnits***);

Visual Basic for Windows
Function AL_SetBurstRate(ByVal ***lhld*** As Long, ByVal ***dRate*** As Double,
ByVal ***lUnits*** As Long) As Long

LHLD *lhld* handle of the LDSD
double *dRate* specifies the rate of burst clocks
long *l*Units specifies the rate units

The **AL_SetBurstRate** function sets the hardware Burst rate clock of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dRate	Specifies the rate at which a burst clock will occur.
lUnits	Sets the units in which the above rate parameter is specified. The available settings are defined as HERTZ or TICS in the ADLIB include file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5401, 5402, 5801 and 5802: ADC0, Options:	3 to 64 specified in Tics
5403, 5404, 5803 and 5804: ADC0, Options:	10 to 64 specified in Tics
5525/50MF: ADC0, Options:	25 to 65535 specified in Tics
Pci55xx Series: ADC0, Options:	1 to 65535 specified in Tics

23.4 AL_GetBurstStruct

Prototype **C\C++**

ERRNUM AL_GetBurstStruct(LHLD ***lhld***, LPBURST ***lpburstStruct***);

Visual Basic for Windows
Function AL_GetBurstStruct(ByVal ***lhld*** As Long, ***lpburstStruct*** As BURST) As Long

LHLD *lhld* handle of the LDSD
LPBURST *lpburstStruct* address of the user burst structure

The **AL_GetBurstStruct** function provides access to all LDSD burst settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpburstStruct	Specifies a 32 bit far pointer to a Burst information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagBURST
{
        char    achBurstModeName[MAX_OPTIONS_NAME_STRING];
        long    lBurstModeId;

        long    lBurstRateUnits;
        double  dBurstRate;

        long    lBurstSamples;

}BURST;
typedef BURST FAR * LPBURST;
```

Returns:
On success ERRNUM is set to 1 and the *lpburstStruct* structure is filled with the device's burst settings, otherwise ERRNUM contains the last error code that occurred during the call and the *lpburstStruct* structure may contain invalid information.

Supported Logical Device Subsystems:
5400 Series: ADC0
5525/50MF: ADC0
5800 Series: ADC0
Pci55xx Series: ADC0

24. CHANNEL SELECTIONS

24.1 AL_SetChannelList

Prototype **C\C++**

ERRNUM AL_SetChannelList(LHLD ***lhld***, LPCHANLIST ***lpchanlist***);

Visual Basic for Windows
Function AL_SetChannelList(ByVal ***lhld*** As Long,
lpchanlist As CHANLIST) As Long

LHLD *lhld* handle of the LDSD
LPCHANLIST *lpchanlist* address of the user channel list structure

The **AL_SetChannelList** function sets the active channel and gain settings in the LDSD. If a device supports gain settings, the list is considered a channel and gain list, and must be set appropriately. See examples below.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpchanlist	Specifies a 32 bit far pointer to a channel information structure. This structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagCHANLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of numbers */

        char        achChannelList[MAX_CHANLIST_STRLEN];

        long        lNumElements;
        long        alChannelList[MAX_CHANLIST_ARRAYLEN];

}CHANLIST;
typedef  CHANLIST FAR * LPCHANLIST;
```

Returns:
On success ERRNUM is set to 1 and the information contained in the lpchanlist structure is transferred to the LDSD, otherwise ERRNUM contains the last error code that occurred during the call.

Examples:
- string “0,1,2,3,4,5,6,7”
- string with gains “0(1),1(2),2(4),3(8),4(1),5(2),6(4),7(8)”
- array {0,1,2,3,4,5,6,7};
- array with gains {0,1,1,2,2,4,3,8,4,1,5,2,6,4,7,8};

Supported Logical Device Subsystems:

5401, 5402, 5801 and 5802: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8
5403, 5404, 5803 and 5804: ADC0, Options:	Min/Max Channel, Gains = 1, 10, 100, 500
5500HR: ADC0, Options:	Min/Max Channel
5500MF:ADC0, Options:	Min/Max Channel
5508LC: ADC0, Options:	Min/Max Channel
5508LF: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8 or 1, 2, 5, 10
5508SCi: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8 or 1, 2, 5, 10
5508SHR: ADC0, Options:	Min/Max Channel
5508TC/BG/RTD: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8
5516DMA: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8
5525/50MF: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8
4012AD: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 4, 8
4112AD: ADC0, Options:	Min/Max Channel, Gains = 1, 5, 10, 20, 100, 200, 500, 1000
Pci5500: ADC0, Options:	Min/Max Channel
Pci5501, Pci5503: ADC0, Options:	Min/Max Channel, Gains = 1, 2, 5, 10
Pci5502, Pci5504: ADC0, Options:	Min/Max Channel, Gains = 1, 10, 100, 1000

24.2 AL_GetChannelList

Prototype **C\C++**

ERRNUM AL_GetChannelList(LHLD ***lhld***, LPCHANLIST ***lpchanlist***);

Visual Basic for Windows
Function AL_GetChannelList(ByVal ***lhld*** As Long,
lpchanlist As CHANLIST) As Long

LHLD *lhld* handle of the LDSD
LPCHANLIST *lpchanlist* address of the user channel list structure

The **AL_GetChannelList** function retrieves the current LDSD channel and gain settings. If a device supports gain settings, the list is considered a channel and gain list, and will be set in a channel then gain sequence. See examples below.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpchanlist	Specifies a 32 bit far pointer to a channel information structure. The structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagCHANLIST
{
        long        lType;    /* 0 = Not Defined */
                              /* 1 = Use string list */
                              /* 2 = Use array of numbers */

        char        achChannelList[MAX_CHANLIST_STRLEN];

        long        lNumElements;
        long        alChannelList[MAX_CHANLIST_ARRAYLEN];

}CHANLIST;
typedef  CHANLIST FAR * LPCHANLIST;
```

Returns:
On success ERRNUM is set to 1 and the lpchanlist structure is filled with the device's channel list settings, otherwise ERRNUM contains the last error code that occurred during the call, and the lpchanlist structure may contain invalid information.

Examples:
- string “0,1,2,3,4,5,6,7”
- string with gains “0(1),1(2),2(4),3(8),4(1),5(2),6(4),7(8)”
- array {0,1,2,3,4,5,6,7};
- array with gains {0,1,1,2,2,4,3,8,4,1,5,2,6,4,7,8};

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0

24.3 AL_SetMinStartChan

Prototype **C\C++**

ERRNUM AL_SetMinStartChan(LHLD ***lhld***, long ***lMinChan***);

Visual Basic for Windows
Function AL_SetMinStartChan(ByVal ***lhld*** As Long,
ByVal ***lMinChan*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lMinChan* specifies the minimum channel number

The **AL_SetMinStartChan** function sets the LDSD maximum allowable starting channel number for a device subsystem.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lMinChan	Minimum channel number allowed.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	0 to 63
5500HR: ADC0, Options:	0 to 15
5500MF:ADC0, Options:	0 to 15
5508LC: ADC0, Options:	0 to 7
5508LF: ADC0, Options:	0 to 7
5508SCi: ADC0, Options:	0 to 31
5508SHR: ADC0, Options:	0 to 7
5508TC/BG/RTD: ADC0, Options:	0 to 7
5516DMA: ADC0, Options:	0 to 15
5525/50MF: ADC0, Options:	0 to 31
5800 Series: ADC0, Options:	0 to 63
4012AD Series: ADC0, Options:	0 to 127
4112AD Series: ADC0, Options:	0 to 127
Pci55xx Series: ADC0, Options:	0 to 127

24.4 AL_GetMinStartChan

Prototype **C\C++**

STATUS AL_GetMinStartChan(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetMinStartChan(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetMinStartChan** function retrieves the current LDSD minimum starting channel for a device subsystem.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success STATUS is set to LDSD minimum starting channel for that device subsystem, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0

24.5 AL_SetMaxEndChan

Prototype **C\C++**

ERRNUM AL_API AL_SetMaxEndChan(LHLD ***lhld***, long ***lMaxChan***);

Visual Basic for Windows
Function AL_SetMaxEndChan(ByVal ***lhld*** As Long,
ByVal ***lMaxChan*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lMaxChan* specifies the maximum channel number

The **AL_SetMaxEndChan** function sets the LDSD maximum allowable ending channel number for a device subsystem.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lMinChan	Maximum channel number allowed.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	0 to 63
5500HR: ADC0, Options:	0 to 15
5500MF:ADC0, Options:	0 to 15
5508LC: ADC0, Options:	0 to 7
5508LF: ADC0, Options:	0 to 7
5508SCi: ADC0, Options:	0 to 31
5508SHR: ADC0, Options:	0 to 7
5508TC/BG/RTD: ADC0	0 to 7
5516DMA: ADC0, Options:	0 to 15
5525/50MF: ADC0, Options:	0 to 31
5800 Series: ADC0, Options:	0 to 63
4012AD Series: ADC0, Options:	0 to 127
Pci55xx Series: ADC0, Options:	0 to 127

24.6 AL_GetMaxEndChan

Prototype **C\C++**

STATUS AL_GetMaxEndChan(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetMaxEndChan(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetMaxEndChan** function retrieves the current LDSD maximum ending channel for a device subsystem.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success STATUS is set to LDSD maximum ending channel for that device subsystem, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0

25. EXPANSION PANEL SELECTIONS

25.1 AL_SetExpPanels

Prototype **C\C++**

ERRNUM AL_SetExpPanels(LHLD ***lhld***, LPEXPPANELLIST ***lpexppanellist***);

Visual Basic for Windows
Function AL_SetExpPanels(ByVal ***lhld*** As Long,
lpexppanellist As EXPPANELLIST) As Long

LHLD *lhld* handle of the LDSD
LPEXPPANELLIST *lpexppanellist* address of the expansion panel list structure

The **AL_SetExpPanels** function sets a board's input panel type(s) in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpexppanellist	Specifies a 32 bit far pointer to a expansion panel information structure. This structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagEXPPANELLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of numbers */

        char        achPanelList[MAX_EXPPANELLIST_STRLEN];

        long        lNumElements;
        long        alPanelList[MAX_EXPPANELLIST_ARRAYLEN];

}EXPPANELLIST;
typedef  EXPPANELLIST FAR * LPEXPPANELLIST;
```

Returns:
On success ERRNUM is set to 1 and the information contained in the *lpexppanellist* structure is transferred to the LDSD, otherwise ERRNUM contains the last error code that occurred during the call.

Examples:

- string “PNLNAME-TC, PNLNAME-TC, PNLNAME-16, PNLNAME-32”
- array {1,1,2,,3};

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	TB5800-16, TB5800-TC or TB5800-64
5508LF: ADC0, Options:	NONE, 5508EXi
5508SCi: ADC0, Options:	NONE, 5508EXi
5525/50MF: ADC0, Options:	NONE, A4012HLEX or A4012TCEX
5800 Series: ADC0, Options:	TB5800-16, TB5800-TC or TB5800-64
4012AD Series: ADC0, Options:	A4012HLEX, A4012TCEX, A4012CLEX, A4012AMEX, A4012CSEX and A4012BGEX
4112AD Series: ADC0, Options:	A4112WRSX and A4112HCVX
Pci5501, 5502, 5503, 5504: ADC0, Options:	PCI_TB5500_16, PCI_TB5500_64 or PCI_TB5500_TC

25.2 AL_GetExpPanelStruct

Prototype **C\C++**

ERRNUM AL_GetExpPanelStruct(LHLD ***lhld***, LPEXPPANELLIST ***lpexppanellist*)**;

Visual Basic for Windows
Function AL_GetExpPanelStruct(ByVal ***lhld*** As Long,
lpexppanellist As EXPPANELLIST) As Long

LHLD *lhld* handle of the LDSD
LPEXPPANELLIST *lpexppanellist* destination address of the expansion panel list structure

The **AL_GetExpPanelStruct** function retrieves to the current LDSD expansion panel list settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpexppanellist	Specifies a 32 bit far pointer to an expansion panel list structure. The structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagEXPPANELLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of numbers */

        char        achPanelList[MAX_EXPPANELLIST_STRLEN];

        long        lNumElements;
        long        alPanelList[MAX_EXPPANELLIST_ARRAYLEN];

}EXPPANELLIST;
typedef  EXPPANELLIST FAR * LPEXPPANELLIST ;
```

Returns:
On success ERRNUM is set to 1 and the *lpexppanellist* structure is filled with the device's expansion panel list settings, otherwise ERRNUM contains the last error code that occurred during the call and the *lpexppanellist* structure may contain invalid information.

Examples:
- string “PNLNAME-TC, PNLNAME-TC, PNLNAME-16, PNLNAME-32”
- array {1,1,2,3};

Supported Logical Device Subsystems:
5400 Series: ADC0
5508LF: ADC0
5508SCi: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci5501, 5502, 5503, 5504 Series: ADC0

25.3 AL_SetExpPanelGains

Prototype **C\C++**

ERRNUM AL_SetExpPanelGains(LHLD ***lhld***, LPEXPPANELGAINLIST ***lpexppanelgainlist***);

Visual Basic for Windows

Function AL_SetExpPanelGains(ByVal ***lhld*** As Long, ***lpexppanelgainlist*** As EXPPANELGAINLIST) As Long

LHLD *lhld* handle of the LDSD

LPEXPPANELGAINLIST *lpexppanelgainlist* address of the expansion panel gain list structure

The **AL_SetExpPanelGains** function sets a board's input panels channel gains in the LDSD. The list is considered a channel and gain list and must be set appropriately, see examples below.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpexppanelgainlist	Specifies a 32 bit far pointer to a expansion panel gain information structure. This structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagEXPPANELGAINLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of numbers */

        char        achPanelGainList[MAX_EXPPANELGAINLIST_STRLEN];

        long        lNumElements;
        long        alPanelGainList[MAX_EXPPANELGAINLIST_ARRAYLEN];

}EXPPANELGAINLIST;
typedef  EXPPANELGAINLIST FAR * LPEXPPANELGAINLIST;
```

Returns:

On success ERRNUM is set to 1 and the information contained in the *lpexppanelgainlist* structure is transferred to the LDSD, otherwise ERRNUM contains the last error code that occurred during the call.

Examples:

- string "0(100),1(100),2(100),3(100),4(100),5(100),6(100),7(100)"
- array {0,100, 1,100, 2,100, 3,100, 4,100, 5,100, 6,100, 7,100};

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	Min/Max Channel, NO Limit on panel gain
5508LF: ADC0, Options:	Min/Max Channel, NO Limit on panel gain
5508SCi: ADC0, Options:	Min/Max Channel, NO Limit on panel gain
5800 Series: ADC0, Options:	Min/Max Channel, NO Limit on panel gain
4012AD Series ADC0, Options:	Min/Max Channel, NO Limit on panel gain
4112AD Series, ADC0, Options:	Min/Max Channel, NO Limit on panel gain
Pci5501, 5502, 5503, 5504 Series: ADC0, Options:	Min/Max Channel, NO Limit on panel gain

25.4 AL_GetExpPanelGainStruct

Prototype **C\C++**

ERRNUM AL_GetExpPanelGainStruct(LHLD ***lhld***, LPEXPPANELGAINLIST ***lpexppanelgainlist***);

Visual Basic for Windows

Function AL_GetExpPanelGainStruct(ByVal ***lhld*** As Long, ***lpexppanelgainlist*** As EXPPANELGAINLIST) As Long

LHLD *lhld*	handle of the LDSD
LPEXPPANELGAINLIST *lpexppanelgainlist*	destination address of the expansion panel gain list

The **AL_GetExpPanelGainStruct** function retrieves to the current LDSD expansion panel gain list settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpexppanellist	Specifies a 32 bit far pointer to an expansion panel gain list structure. The structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagEXPPANELGAINLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of numbers */

        char        achPanelGainList[MAX_EXPPANELGAINLIST_STRLEN];

        long        lNumElements;
        long        alPanelGainList[MAX_EXPPANELGAINLIST_ARRAYLEN];

}EXPPANELGAINLIST;
typedef  EXPPANELGAINLIST FAR * LPEXPPANELGAINLIST;
```

Returns:

On success ERRNUM is set to 1 and the *lpexppanelgainlist* structure is filled with the device's expansion panel gain list settings, otherwise ERRNUM contains the last error code that occurred during the call and the *lpexppanelgainlist* structure may contain invalid information.

Examples:

- string "0(1),1(2),2(4),3(8),4(1),5(2),6(4),7(8)"
- array {0,1, 1,2, 2,4, 3,8, 4,1, 5,2, 6,4, 7,8};

Supported Logical Device Subsystems:

5400 Series: ADC0
5508LF: ADC0
5508SCi: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci5501, 5502, 5503, 5504: ADC0

26. GLOBAL GAIN SETTINGS

26.1 AL_SetGainGlobal

Prototype **C\C++**

ERRNUM AL_SetGainGlobal(LHLD ***lhld***, lpstr ***lpstrGain***);

Visual Basic for Windows
Function AL_SetGainGlobal(ByVal ***lhld*** As Long, ByVal ***lpstrGain*** As String) As Long

LHLD *lhld* handle of the LDSD
lpstr *lpstrGain* address of the gain string

The **AL_SetGainGlobal** function globally sets all channel list gains to the specified *lpstrGain* in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be released.
lpstrGain	Points to the desired gain setting string. The available string settings are device independent and are verified by ADLIB against the available gain options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5401, 5403, 5801 and 5803: ADC0, Options:	1, 2, 4, 8
5402, 5404, 5802 and 5804: ADC0, Options:	1, 10, 100, 500
5508LF: ADC0, Options:	1, 2, 4, 8 or 1, 2, 5, 10
5508SCi: ADC0, Options:	1, 2, 4, 8 or 1, 2, 5, 10
5508TC/BG/RTD: ADC0, Options:	1, 2, 4, 8
5516DMA: ADC0, Options:	1, 2, 4, 8
5525/50MF: ADC0, Options:	1, 2, 4, 8
4012AD Series: ADC0, Options:	1, 2, 4, 8
4112AD Series: ADC0, Options:	1, 5, 10, 20, 100, 200, 500, 1000
Pci5501, Pci5503: ADC0, Options:	1, 2, 5, 10
Pci5502, Pci5504: ADC0, Options:	1, 10, 100, 1000

27. THERMOCOUPLE SUPPORT SELECTIONS

27.1 AL_SetCjList

Prototype **C\C++**

ERRNUM AL_SetCjList(LHLD ***lhld***, LPCJLIST ***lpcjlist***);

Visual Basic for Windows
Function AL_SetCjList(ByVal ***lhld*** As Long, ***lpcjlist*** As CJLIST) As Long

LHLD *lhld* handle of the LDSD
LPCHANLIST *lpchanlist* address of the user CJ list structure

The **AL_SetCjList** function sets the CJ list settings in the LDSD. The CJ listing is comprised of a channel number and CJ type. The CJ list may contain channel/CJ setting that are not currently set in the channel listing itself, only those channels selected within the channel list will be set to the appropriate CJ type when running. The CJ ID type setting in the array can be obtained from the capabilities file.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpcjlist	Specifies a 32 bit far pointer to a CJ information structure. The structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagCJLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of ints */

        char        achCjList[MAX_CJ_LIST_STR];

        long        lNumElements;
        long        alCjList[MAX_CJ_LIST];

}CJLIST;
typedef CJLIST FAR * LPCJLIST;
```

Returns:
On success ERRNUM is set to 1 and the information contained in the lpcjlist structure is transferred to the LDSD, otherwise ERRNUM contains the last error code that occurred during the call.

Examples:
- string "0(BNBS),1(JNBS),2(KNBS),3(SNBS),4(ENBS),5(TNBS),6(RNBS),7(BNBS)"
- array {0,1000, 1,1001, 2,1002, 3,1003, 4,1004, 5,1005, 6,1006, 7,1007};

Supported Logical Device Subsystems:

5400 Series: ADC0, Options: Min/Max Channel, BNBS,JNBS,KNBS,SNBS,ENBS,TNBS,RNBS
5508SCi: ADC0, Options: Min/Max Channel, BNBS,JNBS,KNBS,SNBS,ENBS,TNBS,RNBS
5508TC: ADC0, Options: Min/Max Channel, BNBS,JNBS,KNBS,SNBS,ENBS,TNBS,RNBS
5525/50MF: ADC0, Options: Min/Max Channel, BNBS,JNBS,KNBS,SNBS,ENBS,TNBS,RNBS
5800 Series: ADC0, Options: Min/Max Channel, BNBS,JNBS,KNBS,SNBS,ENBS,TNBS,RNBS
Pci5501, 5502, 5503, 5504 Series: ADC0, Options: Min/Max Channel, BNBS, JNBS, KNBS, SNBS, ENBS, TNBS, RNBS

27.2 AL_GetCjList

Prototype **C\C++**

ERRNUM AL_GetCjList(LHLD ***lhld***, LPCJLIST ***lpcjlist***);

Visual Basic for Windows
Function AL_GetCjList(ByVal ***lhld*** As Long, ***lpcjlist*** As CJLIST) As Long

LHLD *lhld* handle of the LDSD
LPCHANLIST *lpchanlist* destination address of the user channel list structure

The **AL_GetCjList** function retrieves to the current LDSD CJ list settings. The CJ listing is comprised of a channel number and CJ type. The CJ list may contain channel/CJ setting that are not currently set in the channel listing itself. The CJ ID type setting in the array can be obtained from the capabilities file.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpchanlist	Specifies a 32 bit far pointer to a CJ information structure. This structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagCJLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of ints */

        char        achCjList[MAX_CJ_LIST_STR];

        long        lNumElements;
        long        alCjList[MAX_CJ_LIST];

}CJLIST;
typedef CJLIST FAR * LPCJLIST;
```

Returns:
On success ERRNUM is set to 1 and the lpcjlist structure is filled with the device's CJ list settings, otherwise ERRNUM contains the last error code that occurred during the call and the lpcjlist structure may contain invalid information.

Examples:
- string "0(BNBS),1(JNBS),2(KNBS),3(SNBS),4(ENBS),5(TNBS),6(RNBS),7(BNBS)"
- array {0,1000, 1,1001, 2,1002, 3,1003, 4,1004, 5,1005, 6,1006, 7,1007};

Supported Logical Device Subsystems:
5400 Series: ADC0
5508SCi: ADC0
5508TC: ADC0
5525/50MF: ADC0
5800 Series: ADC0
Pci5501, 5502, 5503, 5504 Series: ADC0

27.3 AL_SetCjGlobal

Prototype **C\C++**

ERRNUM AL_SetCjGlobal(LHLD ***lhld***, lpstr ***lpstrCj***);

Visual Basic for Windows
Function AL_SetCjGlobal(ByVal ***lhld*** As Long, ByVal ***lpstrCj*** As String) As Long

LHLD *lhld* handle of the LDSD
lpstr *lpstrCjt* address of the CJ type string

The **AL_SetCjGlobal** function globally sets all Cj list thermocouple types to the specified *lpstrCj* in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be released.
lpstrCj	Points to the desired Cj type setting string. The available string settings are device independent and are verified by ADLIB against the available Cj options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	BNBS,JNBS,KNBS,ENBS,TNBS,RNBS,SNBS
5508SCi: ADC0, Options:	BNBS,JNBS,KNBS,ENBS,TNBS,RNBS,SNBS
5508TC: ADC0, Options:	BNBS,JNBS,KNBS,ENBS,TNBS,RNBS,SNBS
5525/50MF: ADC0, Options:	BNBS,JNBS,KNBS,ENBS,TNBS,RNBS,SNBS
5800 Series: ADC0, Options:	BNBS,JNBS,KNBS,ENBS,TNBS,RNBS,SNBS
Pci5501, 5502, 5503, 5504 Series: ADC0, Options:	BNBS,JNBS,KNBS,ENBS,TNBS,RNBS,SNBS

28. DMA CONFIGURATIONS

28.1 AL_SetDmaMode

Prototype **C\C++**

ERRNUM AL_SetDmaMode(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetDmaMode(ByVal ***lhld*** As Long, ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the DMA mode string

The **AL_SetDmaMode** function sets the hardware DMA operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired DMA mode string. The available string settings are device independent and are verified by ADLIB against the available DMA mode options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0, Options: SINGLE
5500HR: ADC0, Options: SINGLE
5508LC: ADC0, Options: SINGLE
5508LF: ADC0, Options: SINGLE
5508SCi: ADC0, Options: SINGLE
5508SHR: ADC0, Options: SINGLE
5516DMA: ADC0, Options: SINGLE
5800 Series: ADC0, Options: SINGLE

28.2 AL_GetDmaMode

Prototype **C\C++**

ERRNUM AL_GetDmaMode(LHLD ***lhld***, LPSTR ***lpstrMode***, long ***lMaxLength***);

Visual Basic for Windows
Function AL_GetDmaMode(ByVal ***lhld*** As Long, ByVal ***lpstrMode*** As String,
ByVal ***lMaxLength*** As Long) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* destination address of the DMA mode string
long *lMaxLength* the count in bytes to be copied

The **AL_GetDmaMode** function retrieves the current DMA operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrMode* is set to the LDSD DMA mode, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5516DMA: ADC0
5800 Series: ADC0

28.3 AL_SetDmaChan

Prototype **C\C++**

ERRNUM AL_SetDmaChan(LHLD ***lhld***, LPSTR ***lpstrChan***);

Visual Basic for Windows
Function AL_SetDmaChan(ByVal ***lhld*** As Long, ByVal ***lpstrChan*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the DMA channel string

The **AL_SetDmaChan** function sets the hardware DMA channel of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode	Points to the desired DMA channel string. The available string settings are device independent and are verified by ADLIB against the available DMA channel options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	DISABLED, DMA5, DMA6, DMA7
5500HR: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5508LC: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5508LF: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5508SCi: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5508SHR: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5516DMA: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5525/50MF: ADC0, Options:	DISABLED, DMA1, DMA2, DMA3
5800 Series: ADC0, Options:	DISABLED, DMA5, DMA6, DMA7

28.4 AL_GetDmaChan

Prototype **C\C++**

ERRNUM AL_GetDmaChan(LHLD ***lhld***, LPSTR ***lpstrChan***, long ***lMaxLength***);

Visual Basic for Windows
Function AL_GetDmaChan(ByVal ***lhld*** As Long, ByVal ***lpstrChan*** As String,
ByVal ***lMaxLength*** As Long) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrChan* destination address of the DMA channel string
long *lMaxLength* the count in bytes to be copied

The **AL_GetDmaChan** function retrieves the current DMA channel of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrChan	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrChan* is set to the LDSD DMA channel, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0

29. INTERRUPT CONFIGURATIONS

29.1 AL_SetIrqLevel

Prototype **C\C++**

ERRNUM AL_SetIrqLevel(LHLD ***lhld***, LPSTR ***lpstrIrqLevel***);

Visual Basic for Windows
Function AL_SetIrqLevel(ByVal ***lhld*** As Long, ByVal ***lpstrIrqLevel*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrIrqLevel* address of the IRQ level string

The **AL_SetIrqLevel** function sets the hardware IRQ level of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrIrqLevel	Points to the desired IRQ level string. The available string settings are device independent and are verified by ADLIB against the available IRQ level options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	DISABLED, IRQ3, IRQ5, IRQ7, IRQ10, IRQ11, IRQ15
5500HR: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5508LC: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5508LF: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5508SCi: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5508SHR: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5508TC/BG/RTD: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5516DMA: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5525/50MF: ADC0, Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5600 Series ACI, DCI, CCI, TTL: DIN0 Options:	DISABLED, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7
5800 Series: ADC0, Options:	DISABLED, IRQ3, IRQ5, IRQ7, IRQ10, IRQ11, IRQ15

29.2 AL_GetIrqLevel

Prototype **C\C++**

ERRNUM AL_GetIrqLevel(LHLD ***lhld***, LPSTR ***lpstrIrqLevel***, long ***lMaxLength*****)**;

Visual Basic for Windows
Function AL_GetIrqLevel(ByVal ***lhld*** As Long, ByVal ***lpstrIrqLevel*** As String,
ByVal ***lMaxLength*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrIrqLevel*	destination address of the IRQ level string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetIrqLevel** function retrieves the current IRQ level of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrIrqLevel	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrIrqLevel* is set to the LDSD IRQ level, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0

30. SIGNAL PATHS

30.1 AL_SetSignalPath

Prototype **C\C++**

ERRNUM AL_SetSignalPath(LHLD ***lhld***, LPSTR ***lpstrSignalPath***);

Visual Basic for Windows
Function AL_SetSignalPath(ByVal ***lhld*** As Long,
ByVal ***lpstrSignalPath*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrSignalPath* address of the signal path string

The **AL_SetSignalPath** function sets the hardware data code of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSignalPath	Points to the desired signal path string. The available string settings are device independent and are verified by ADLIB against the available signal path options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
NONE

30.2 AL_GetSignalPath

Prototype **C\C++**

ERRNUM AL_GetSignalPath(LHLD ***lhld***, LPSTR ***lpstrSignalPath***, long ***lMaxlen***);

Visual Basic for Windows
Function AL_GetSignalPath(ByVal ***lhld*** As Long, ByVal ***lpstrSignalPath*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrSignalPath*	destination address of the signal path string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataCode** function retrieves the current signal path of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrSignalPath	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrSignalPath* is set to the LDSD signal path, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
NONE

31. INPUT CONFIGURATIONS

31.1 AL_SetInputConfig

Prototype **C\C++**

ERRNUM AL_SetInputConfig(LHLD ***lhld***, LPSTR ***lpstrInputConfig***);

Visual Basic for Windows
Function AL_SetInputConfig(ByVal ***lhld*** As Long,
ByVal ***lpstrInputConfig*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrInputConfig* address of the input configuration string

The **AL_SetInputConfig** function sets the hardware input configuration of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrInputConfig	Points to the desired input configuration string. The available string settings are device independent and are verified by ADLIB against the available signal path options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0, Options: DIFFERENTIAL, SINGLE-ENDED or PSEUDO-DIFFERENTIAL
5800 Series: ADC0, Options: DIFFERENTIAL, SINGLE-ENDED or PSEUDO-DIFFERENTIAL

31.2 AL_GetInputConfig

Prototype **C\C++**

ERRNUM AL_GetInputConfig(LHLD ***lhld***, LPSTR ***lpstrInputConfig***, long ***lMaxlen***);

Visual Basic for Windows

Function AL_GetInputConfig(ByVal ***lhld*** As Long, ByVal ***lpstrInputConfig*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrInputConfig*	destination address of the input configuration string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetInputConfig** function retrieves the current signal path of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrInputConfig	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:

On success ERRNUM is set to 1 and the destination string *lpstrInputConfig is* set to the LDSD input configuration, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:

5400 Series: ADC0
5800 Series: ADC0

31.3 AL_SetInputConfigList

Prototype **C\C++**

ERRNUM AL_SetInputConfigList(LHLD ***lhld***, LPINPUTCONFIGLIST ***lpInputConfiglist***);

Visual Basic for Windows
Function AL_SetInputConfigList(ByVal ***lhld*** As Long, ***lpInputConfiglist*** As INPUTCONFIGLIST) As Long

LHLD *lhld*	handle of the LDSD
LPINPUTCONFIGLIST *lpchanlist*	address of the user INPUTCONFIGLIST structure

The **AL_SetInputConfigList** function sets the InputConfig list settings in the LDSD. The INPUTCONFIG listing is comprised of a channel number and INPUTCONFIG type. The INPUTCONFIG list may contain Channel/InputConfig settings that are not currently set in the channel listing itself, only those channels selected within the channel list will be set to the appropriate InputConfig type when running. The InputConfig ID setting in the array can be obtained from the capability file.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpInputConfiglist	Specifies a 32 bit far pointer to a INPUTCONFIG information structure. The structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagINPUTCONFIGLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of ints */

        char        achInputConfigList[MAX_INPUTCONFIG_STRLEN];

        long        lNumElements;
        long        alInputConfigList[MAX_INPUTCONFIG_LIST_ARRAYLEN];

}INPUTCONFIG;
typedef INPUTCONFIG FAR * LPINPUTCONFIGLIST;
```

Returns:
On success ERRNUM is set to 1 and the information contained in the lpInputConfiglist structure is transferred to the LDSD, otherwise ERRNUM contains the last error code that occurred during the call.

Examples:
- string "0(DI),1(PD),2(SE),3(DI),4(DI),5(SE),6(SE),7(PD)"
- array {0,200, 1,220, 2,210, 3,200, 4,200, 5,210, 6,210, 7,220};

Supported Logical Device Subsystems:
Pci55xx Series: ADC0, Options: Min/Max Channel, DI, SE, PD

31.4 AL_GetInputConfigList

Prototype **C\C++**

ERRNUM AL_GetInputConfigList(LHLD ***lhld***, LPINPUTCONFIGLIST ***lpInputConfiglist***);

Visual Basic for Windows
Function AL_GetInputConfigList(ByVal ***lhld*** As Long, ***lpInputConfiglist*** As INPUTCONFIGLIST) As Long

LHLD *lhld* handle of the LDSD
LPINPUTCONFIGLIST *lpInputConfiglist* destination address of the user channel list structure

The **AL_GetInputConfigList** function retrieves to the current LDSD InputConfig list. The INPUTCONFIG listing is comprised of a channel number and InputConfig type. The InputConfig list may contain Channel/InputConfig settings that are not currently set in the channel listing itself. The InputConfigID setting in the array can be obtained from the capability file.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpInputConfiglist	Specifies a 32 bit far pointer to a INPUTCONFIG information structure. This structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagINPUTCONFIGLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of ints */

        char        achInputConfigList[MAX_INPUTCONFIG_STRLEN];

        long        lNumElements;
        long        alInputConfigList[MAX_INPUTCONFIG_LIST_ARRAYLEN];

}INPUTCONFIG;
typedef INPUTCONFIG FAR * LPINPUTCONFIGLIST;
```

Returns:
On success ERRNUM is set to 1 and the lpInputConfiglist structure is filled with the device's InputConfig list settings, otherwise ERRNUM contains the last error code that occurred during the call and the lpInputConfiglist structure may contain invalid information.

Examples:
- string "0(DI),1(PD),2(SE),3(DI),4(DI),5(SE),6(SE),7(PD)"
- array {0,200, 1,220, 2,210, 3,200, 4,200, 5,210, 6,210, 7,220};

Supported Logical Device Subsystems:
Pci55xx Series: ADC0

31.5 AL_SetInputConfigGlobal

Prototype **C\C++**

ERRNUM AL_SetInputConfigGlobal(LHLD ***lhld***, lpstr ***lpstrInputConfig***);

Visual Basic for Windows
Function AL_SetInputConfigGlobal(ByVal ***lhld*** As Long, ByVal ***lpstrInputConfig*** As String) As Long

LHLD *lhld* handle of the LDSD
lpstr *lpstrInputConfig* address of the InputConfig type string

The **AL_SetInputConfigGlobal** function globally sets all InputConfig list input types to the specified *lpstrInputConfig* in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be released.
LpstrInputConfig	Points to the desired InputConfig type setting string. The available string settings are device independent and are verified by ADLIB against the available InputConfig options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx Series: ADC0, Options: Min/Max Channel, DI, SE, PD

32. OUTPUT CONFIGURATIONS

32.1 AL_SetOutputConfig

Prototype **C\C++**

ERRNUM AL_SetOutputConfig(LHLD ***lhld***, LPSTR ***lpstrOutputConfig***);

Visual Basic for Windows
Function AL_SetOutputConfig(ByVal ***lhld*** As Long,
ByVal ***lpstrOutputConfig*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrOutputConfig* address of the output configuration string

The **AL_SetOutputConfig** function sets the hardware output configuration of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrOutputConfig	Points to the desired output configuration string. The available string settings are device independent and are verified by ADLIB against the available signal path options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
NONE

32.2 AL_GetOutputConfig

Prototype **C\C++**

ERRNUM AL_GetOutputConfig(LHLD ***lhld***, LPSTR ***lpstrOutputConfig***, long ***lMaxlen***);

Visual Basic for Windows

Function AL_GetOutputConfig(ByVal *lhld* As Long,
ByVal ***lpstrOutputConfig*** As String,
ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrOutputConfig*	destination address of the output configuration string
long *lMaxLength*	the count in bytes to be copied

The AL_GetOutputConfig function retrieves the current signal path of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrOutputConfig	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrOutputConfig is* set to the LDSD output configuration, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
NONE

33. DATA CODES

33.1 AL_SetDataCode

Prototype **C\C++**

ERRNUM AL_SetDataCode(LHLD ***lhld***, LPSTR ***lpstrDataCode***);

Visual Basic for Windows
Function AL_SetDataCode(ByVal ***lhld*** As Long,
ByVal ***lpstrDataCode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrDataCode* address of the data code string

The **AL_SetDataCode** function sets the hardware data code of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDatacode	Points to the desired data code string. The available string settings are device independent and are verified by ADLIB against the available data code options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: DAC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5500HR: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5500MF:ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5504DA:DAC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5508LC: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5508LF: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5508SCi: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5508TC/BG/RTD: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5516DMA: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5525/50MF: DAC0 and ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
5800 Series: DAC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY

4012AD Series: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY
4112AD Series: ADC0, Options:	TWOS COMPLEMENT, STRAIGHT_BINARY or OFFSET_BINARY

33.2 AL_GetDataCode

Prototype **C\C++**

ERRNUM AL_GetDataCode(LHLD ***lhld***, LPSTR ***lpstrDataCode***, long ***lMaxlen***);

Basic Value for Windows

Function AL_GetDataCode(ByVal ***lhld*** As Long, ByVal ***lpstrDataCode*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrDataCode*	destination address of the data code string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataCode** function retrieves the current data code of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataCode	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrDataCode* is set to the LDSD data code, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: DAC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0 and DAC0
5800 Series: DAC0
4012AD Series: ADC0
4112AD Series: ADC0

34. DATA OFFSETS

34.1 AL_SetDataOffset

Prototype **C\C++**

ERRNUM AL_SetDataOffset(LHLD ***lhld***, LPSTR ***lpstrDataOffset***);

Visual Basic for Windows
Function AL_SetDataOffset(ByVal ***lhld*** As Long,
ByVal ***lpstrDataOffset*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrDataOffset* address of the data offset level string

The **AL_SetDataOffset** function sets the hardware data offset of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataOffset	Points to the desired data offset string. The available string settings are device independent and are verified by ADLIB against the available data offset options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0 and DAC0, Options:	BIPOLAR or UNIPOLAR
5500HR: ADC0, Options:	BIPOLAR or UNIPOLAR
5500MF:ADC0, Options:	BIPOLAR or UNIPOLAR
5504DA:DAC0, Options:	BIPOLAR or UNIPOLAR
5508LC: ADC0, Options:	BIPOLAR or UNIPOLAR
5508LF: ADC0, Options:	BIPOLAR or UNIPOLAR
5508SCi: ADC0, Options:	BIPOLAR or UNIPOLAR
5508SHR: ADC0, Options:	BIPOLAR or UNIPOLAR
5508TC/BG/RTD: ADC0, Options:	BIPOLAR or UNIPOLAR
5516DMA: ADC0, Options:	BIPOLAR or UNIPOLAR
5525/50MF: ADC0 and DAC0, Options:	BIPOLAR or UNIPOLAR
5800 Series: ADC0 and DAC0, Options:	BIPOLAR or UNIPOLAR
4012AD Series: ADC0, Options:	BIPOLAR or UNIPOLAR
4112AD Series: ADC0, Options:	BIPOLAR or UNIPOLAR

34.2 AL_GetDataOffset

Prototype **C\C++**

ERRNUM AL_GetDataOffset(LHLD ***lhld***, LPSTR ***lpstrDataOffset***, long ***lMaxlen***);

Visual Basic for Windows
Function AL_GetDataOffset(ByVal ***lhld*** As Long, ByVal ***lpstrDataOffset*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrDataOffset*	destination address of the data offset string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataCode** function retrieves the current data offset of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataOffset	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrDataOffset* is set to the LDSD data offset, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: ADC0 and DAC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0 and DAC0
5800 Series: ADC0 and DAC0
4012AD Series: ADC0
4112AD Series: ADC0

34.3 AL_SetDataOffsetList

Prototype **C\C++**

ERRNUM AL_SetDataOffsetList(LHLD ***lhld***, LPDATAOFFSETLIST ***lpDataOffsetlist***);

Visual Basic for Windows
Function AL_SetDataOffsetList(ByVal ***lhld*** As Long, ***lpDataOffsetlist*** As DATAOFFSETLIST) As Long

LHLD *lhld* handle of the LDSD
LPCHANLIST *lpchanlist* address of the user DATAOFFSET list structure

The **AL_SetDataOffsetList** function sets the DataOffset list settings in the LDSD. The DataOffset listing is comprised of a channel number and DataOffset type. The DataOffset list may contain Channel/DataOffset settings that are not currently set in the channel listing itself, only those channels selected within the channel list will be set to the appropriate DataOffset type when running. The DataOffset ID setting in the array can be obtained from the capabilities file.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpDataOffsetlist	Specifies a 32 bit far pointer to a DATAOFFSET information structure. The structure has been pre-defined in the ADLIB include file and is defined as follows:

```
typedef struct tagDATAOFFSETLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of ints */

        char        achDataOffsetList[MAX_DATAOFFSET_STRLEN];

        long        lNumElements;
        long        alDataOffsetList[MAX_DATAOFFSET_LIST_ARRAYLEN];

}DATAOFFSET;
typedef DATAOFFSET FAR * LPDATAOFFSETLIST;
```

Returns:
On success ERRNUM is set to 1 and the information contained in the lpDataOffsetlist structure is transferred to the LDSD, otherwise ERRNUM contains the last error code that occurred during the call.

Examples:

- string "0(BIP),1(UNI),2(BIP),3(BIP),4(BIP),5(BIP),6(UNI),7(UNI)"
- array {0,1800, 1,1801, 2, 1800, 3, 1800, 4, 1800, 5, 1800, 6, 1801, 7, 1801};

Supported Logical Device Subsystems:
Pci55xx Series: ADC0, Options: Min/Max Channel, BIP, UNI

34.4 AL_GetDataOffsetList

Prototype **C\C++**

ERRNUM AL_GetDataOffsetList(LHLD ***lhld***, LPDATAOFFSETLIST ***lpDataOffsetlist***);

Visual Basic for Windows
Function AL_GetDataOffsetList(ByVal ***lhld*** As Long, ***lpDataOffsetlist*** As DATAOFFSETLIST) As Long

LHLD *lhld* handle of the LDSD
LPCHANLIST *lpchanlist* destination address of the user channel list structure

The **AL_GetDataOffsetList** function retrieves to the current LDSD DataOffsetlist settings. The DataOffsetlisting is comprised of a channel number and DATAOFFSET type. The DATAOFFSET list may contain Channel/DataOffset settings that are not currently set in the channel listing itself. The DataOffset ID setting in the array can be obtained from the capability file.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpchanlist	Specifies a 32 bit far pointer to a DATAOFFSET information structure. This structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagDATAOFFSETLIST
{
        long        lType;   /* 0 = Not Defined */
                             /* 1 = Use string list */
                             /* 2 = Use array of ints */

        char        achDataOffsetList[MAX_DATAOFFSET_STRLEN];

        long        lNumElements;
        long        alDataOffsetList[MAX_DATAOFFSET_LIST_ARRAYLEN];

}DATAOFFSET
typedef DATAOFFSET FAR * LPDATAOFFSETLIST;
```

Returns:
On success ERRNUM is set to 1 and the lpDataOffsetlist structure is filled with the device's DataOffsetlist settings, otherwise ERRNUM contains the last error code that occurred during the call and the lpDataOffsetlist structure may contain invalid information.

Examples:

- string “0(BIP),1(UNI),2(BIP),3(BIP),4(BIP),5(BIP),6(UNI),7(UNI)”
- array {0,1800, 1,1801, 2, 1800, 3, 1800, 4, 1800, 5, 1800, 6, 1801, 7, 1801};

Supported Logical Device Subsystems:
Pci55xx Series: ADC0

34.5 AL_SetDataOffsetGlobal

Prototype **C\C++**

ERRNUM AL_SetDataOffsetGlobal(LHLD ***lhld***, lpstr ***lpstrDataOffset***);

Visual Basic for Windows
Function AL_SetDataOffsetGlobal(ByVal ***lhld*** As Long, ByVal ***lpstrDataOffset*** As String) As Long

LHLD *lhld* handle of the LDSD
lpstr *lpstrDataOffsett* address of the DATAOFFSET type string

The **AL_SetDataOffsetGlobal** function globally sets all DataOffset list inputs to the specified *lpstrDataOffset* in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem to be released.
lpstrDataOffset	Points to the desired DataOffset type setting string. The available string settings are device independent and are verified by ADLIB against the available DataOffset options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx Series: ADC0, Options: Min/Max Channel, BIP, UNI

35. DATA SPANS

35.1 AL_SetDataSpan

Prototype **C\C++**

ERRNUM AL_SetDataSpan(LHLD ***lhld***, LPSTR ***lpstrDataSpan***);

Visual Basic for Windows
Function AL_SetDataSpan(ByVal ***lhld*** As Long,
ByVal ***lpstrDataSpan*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrDataSpan* address of the data span string

The **AL_SetDataSpan** function sets the hardware data span of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataSpan	Points to the desired data span string. The available string settings are device independent and are verified by ADLIB against the available data span options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
NONE

35.2 AL_GetDataSpan

Prototype **C\C++**

ERRNUM AL_GetDataSpan(LHLD ***lhld***, LPSTR ***lpstrDataSpan***, long ***lMaxlen***);

Visual Basic for Windows
Function AL_GetDataSpan(ByVal ***lhld*** As Long, ByVal ***lpstrDataSpan*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrDataSpan*	destination address of the data span string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataCode** function retrieves the current data span of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataSpan	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrDataSpan* is set to the LDSD data span, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
NONE

36. DATA RANGES

36.1 AL_SetDataRange

Prototype **C\C++**

ERRNUM AL_SetDataRange(LHLD ***lhld***, LPSTR ***lpstrDataRange***);

Visual Basic for Windows
Function AL_SetDataRange(ByVal ***lhld*** As Long,
ByVal ***lpstrDataRange*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrDataRange* address of the data range string

The **AL_SetDataRange** function sets the hardware data range of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataRange	Points to the desired data range string. The available string settings are device independent and are verified by ADLIB against the available data range options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: DAC0, Options:	-10_10 or 0_10
5500HR: ADC0, Options:	-10_10, -5_5 or 0_10
5500MF:ADC0, Options:	-10_10, -5_5 or 0_10
5504DA:DAC0, Options:	-10_10, -5_5 or 0_10
5508LC: ADC0, Options:	-10_10, -5_5 or 0_10
5508LF: ADC0, Options:	-10_10, -5_5 or 0_10
5508SCi: ADC0, Options:	-10_10, -5_5 or 0_10
5508TC/BG/RTD: ADC0, Options:	-10_10, -5_5 or 0_10
5516DMA: ADC0, Options:	-10_10, -5_5 or 0_10
5525/50MF: ADC0 and DAC0, Options:	-10_10, -5_5 or 0_10
5800 Series: DAC0, Options:	-10_10 or 0_10
4012AD Series: ADC0 Options:	-10_10, -5_5 or 0_10
4112AD Series: ADC0 Options:	-10_10, -5_5 or 0_10

36.2 AL_GetDataRange

Prototype **C\C++**

ERRNUM AL_GetDataRange(LHLD ***lhld***, LPSTR ***lpstrDataRange***, long ***lMaxlen***);

Visual Basic for Windows
Function AL_GetDataRange(ByVal ***lhld*** As Long, ByVal ***lpstrDataRange*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrDataRange*	destination address of the data range string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataRange** function retrieves the current data range of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrDataRange	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1 and the destination string *lpstrDataRange* is set to the LDSD data range, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:
5400 Series: DAC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0 and DAC0
5800 Series: DAC0
4012AD Series: ADC0
4112AD Series: ADC0

37. FILTERING

37.1 AL_SetFilterType

Prototype **C\C++**

ERRNUM AL_SetFilterType(LHLD ***lhld***, LPSTR ***lpstrType***);

Visual Basic for Windows
Function AL_ SetFilterType(ByVal ***lhld*** As Long,
ByVal ***lpstrType*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrType* address of the filter type string

The **AL_ SetFilterType** function sets the hardware input filter type of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrType	Points to the desired filter type source string. The available string settings are device independent and are verified by ADLIB against the available filter type options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5508SCi: ADC0, Options: DISABLED, BUTTERWORTH, BESSEL

37.2 AL_SetFilterFreq

Prototype **C\C++**

ERRNUM AL_SetFilterFreq(LHLD ***lhld***, double ***dFreq***);

Visual Basic for Windows
Function AL_ SetFilterFreq(ByVal ***lhld*** As Long, ByVal ***dFreq*** As Double) As Long

LHLD *lhld* handle of the LDSD
double *dFreq* specifies the frequency of the filter

The **AL_ SetFilterFreq** function sets the hardware filter frequency of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dFreq	Specifies the hardware input filter frequency.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5508SCi: ADC0 0 to 100000

37.3 AL_GetFilterStruct

Prototype **C\C++**

ERRNUM AL_GetFilterStruct(LHLD ***lhld***, LPFILTER ***lpfilterStruct***);

Visual Basic for Windows
Function AL_ GetFilterStruct(ByVal ***lhld*** As Long, ***lpfilterStruct*** As FILTER) As Long

LHLD *lhld* handle of the LDSD
LPFILTER *lpfilterStruct* address of the user filter structure

The **AL_ GetFilterStruct** function provides access to all LDSD filter settings.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpfilterStruct	Specifies a 32 bit far pointer to a filter information structure. This structure has been predefined in the ADLIB include file and is defined as follows:

```
typedef struct tagFILTER
{
        char    achFilterTypeName[MAX_OPTIONS_NAME_STRING];
        long    lFilterTypeId;

        double  dFilterFreq;

}FILTER;
typedef  FILTER FAR * LPFILTER;
```

Returns:
On success ERRNUM is set to 1 and the *lpfilterStruct* structure is filled with the device's filter settings, otherwise ERRNUM contains the last error code that occurred during the call and the *lpfilterStruct* structure may contain invalid information.

Supported Logical Device Subsystems:
5508SCi: ADC0

38. COUNTER MODES

38.1 AL_SetCtrMode

Prototype **C\C++**

ERRNUM AL_SetCtrMode(LHLD ***lhld***, LPSTR ***lpstrMode***);

Visual Basic for Windows
Function AL_SetCtrMode(ByVal ***lhld*** As Long,
ByVal ***lpstrMode*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMode* address of the Counter/Timer mode string

The **AL_SetCtrMode** function sets the hardware Counter/Timer operation mode of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMode are device	Points to the desired Counter/Timer mode string. The available string settings
	independent and are verified by ADLIB against the available trigger mode options specified in the device's capabilities file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

Pci55xx Series: CTR0, CTR1 Options:	PULSE_COUNTER_INCREMENTAL, PULSE_COUNTER_CUMMULATIVE
Pci55xx Series: CTR2, CTR3 Options:	SQUARE_WAVE_OUTPUT

38.2 AL_GetCtrMode

Prototype **C\C++**

ERRNUM AL_GetCtrMode(LHLD ***lhld***, LPSTR ***lpstrCtrMode***, long ***lMaxlen***);

Visual Basic for Windows

Function AL_GetCtrMode(ByVal ***lhld*** As Long, ByVal ***lpstrCtrMode*** As String, ByVal ***lMaxlen*** As Long) As Long

LHLD *lhld*	handle of the LDSD
LPSTR *lpstrCtrMode*	destination address of the Counter/Timer mode string
long *lMaxLength*	the count in bytes to be copied

The **AL_GetDataCode** function retrieves the current Counter/Timer modeof the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrCtrMode	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user string , including the NULL termination character.

Returns:

On success ERRNUM is set to 1 and the destination string *lpstrCtrMode* is set to the LDSD Counter/Timer mode, otherwise ERRNUM contains the last error code that occurred during the call and the destination string is considered invalid.

Supported Logical Device Subsystems:

Pci55xx Series: CTR0, CTR1, CTR2, CTR3

39. COUNTER / TIMER

39.1 AL_CounterIn

Prototype **C\C++**

STATUS AL_API AL_CounterIn(LHLD ***lhld***);

Visual Basic for Windows
Function AL_CounterIn(ByVal ***lhld*** As Long,) As Long

LHLD *lhld* handle of the LDSD

The **AL_CounterIn** function immediately retrieves the current count of a Counter Input port.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.

Returns:
On success **STATUS** contains the current count, otherwise ERRNUM contains the negative error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx Series: CTR0, CTR1

39.2 AL_CounterOut

Prototype **C\C++**

ERRNUM AL_CounterOut(LHLD ***lhld***, double ***dRate***, long ***lUnits***);

Visual Basic for Windows

Function AL_CounterOut(ByVal ***lhld*** As Long, ByVal ***dRate*** As Double,
ByVal ***lUnits*** As Long) As Long

LHLD *lhld*	handle of the LDSD
double *dRate*	specifies the rate of the Counter Output port
long *l*Units	specifies the rate units

The **AL_CounterOut** function sets the hardware Counter Output port rate of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
dRate	Specifies the rate at which a Counter Output will occur.
lUnits	Sets the units in which the above rate parameter is specified. The available settings are defined as HERTZ or TICS in the ADLIB include file.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx Series: CTR2, CTR3, Options: 7.7 to 500000

40. BUFFER HANDLER FUNCTIONS

40.1 AL_ClearBufferDoneFlag

Prototype **C\C++**

ERRNUM AL_ClearBufferDoneFlag(LHLD ***lhld***, long ***lBuffNum***);

Visual Basic for Windows
Function AL_ClearBufferDoneFlag(ByVal ***lhld*** As Long,
ByVal ***lBuffNum*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lBuffNum* specifies the buffer

The **AL_ClearBufferDoneFlag** function informs the ADLIB LDSD that the buffer is available for use.

All buffers are flagged as done when filled and it's the user applications' responsibility to inform the LDSD that the buffer is once again available. This function is therefor required when the LDSD cycle mode is set to AL_CONTINUOUS_CYCLE, otherwise LDSD will generate an error when it attempts to reuse a buffer that has not been released back to the LDSD. Setting ***lBuffNum*** to **INIT_ALL_BUFFERS** will reinitialize all buffers for the specified LDSD, this is most usefull in the AL_SINGLE_CYCLE mode when all buffers have been completed.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBuffNum	Specifies the buffer ID number to be released back to the LDSD.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.2 AL_GetDoneBuffPtr

Prototype **C\C++**

LPBUFFSTRUCT AL_GetDoneBuffPtr(LHLD lhld);

Visual Basic for Windows
Not supported

LHLD *lhld* handle of the LDSD

The **AL_GetDoneBuffPtr** function retrieves a buffer pointer from the LDSD FIFO. Buffers are placed into the FIFO and are returned by **AL_GetDoneBuffPtr** as a first done, first returned ordering.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.

Returns:
On success LPBUFFSTRUCT points to the current buffer, otherwise LPBUFFSTRUCT is set to NULL indicating no buffers are available. The LPBUFFSTRUCT is a structure defined in adlib.h which contains a pointer to the buffer and current status information of the buffer. See the Memory Buffer Allocation chapter for a complete description of buffer handling concepts and the LPBUFFSTRUCT structure details.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.3 AL_GetBuffPtr

Prototype **C\C++**

LPBUFFSTRUCT AL_GetBuffPtr(LHLD ***lhld***, long ***lBuffNum***);

Visual Basic for Windows
Not supported

LHLD *lhld* handle of the LDSD
long *lBuffNum* specifies the buffer

The **AL_GetBuffPtr** function retrieves a buffer pointer from the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBuffNum	Specifies the buffer ID number from which a pointer is returned.

Returns:
On success LPBUFFSTRUCT points to the desired buffer, otherwise LPBUFFSTRUCT is set to NULL indicating the buffer is not available. The LPBUFFSTRUCT is a structure defined in adlib.h which contains a pointer to the buffer and current status information of the buffer. See the Memory Buffer Allocation chapter for a complete description of buffer handling concepts and the LPBUFFSTRUCT structure details.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.4 AL_CopyBuffer

Prototype **C\C++**

(16 Bit) ERRNUM AL_CopyBuffer(LHLD ***lhld***, long ***lBuffNum***,
int _far *****fpDataBuff***, long ***lStartPoint***, long ***lCount***);

(32 Bit) ERRNUM AL_CopyBuffer(LHLD ***lhld***, long ***lBuffNum***,
int.****fpDataBuff***, long ***lStartPoint***, long ***lCount***);

Visual Basic for Windows
Function AL_ CopyBuffer(ByVal ***lhld*** As Long, ByVal ***lBuffNum*** As Long,
fpDataBuff as Integer, ByVal ***lStartPoint*** As Long,
ByVal ***lCount*** As Long) As Long

LHLD *lhld* handle of the LDSD

long *lBuffNum* specifies the buffer number

int _far * *fpDataBuff* destination address of the data buffer

long *lStartPoint* starting point in the source buffer to be copied

long *lCount* number of samples

The **AL_ GetCopyBuffer** function copies an LDSD source buffer specified by *lBuffNum* to the user *fpDataBuff* destination buffer.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBuffNum	Specifies the ID number of the LDSD buffer to be copied.
fpDataBuff	Specifies the user buffer destination address.
lStartPoint	Starting sample point in the LDSD source buffer to be copied.
lCount	Number of LDSD data buffer samples to copy.

Returns:
On success **fpDataBuff** is set with the specified LDSD buffer's data, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.5 AL_SetBuffer

Prototype **C\C++**

(16 Bit) ERRNUM AL_SetBuffer(LHLD ***lhld***, long ***lBuffNum***, int _far ****fpDataBuff***, long ***lStartPoint***, long ***lCount***);

(32 Bit) ERRNUM AL_SetBuffer(LHLD ***lhld***, long ***lBuffNum***, int ****fpDataBuff***, long ***lStartPoint***, long ***lCount***);

Visual Basic for Windows

Function AL_ SetBuffer(ByVal ***lhld*** As Long, ByVal ***lBuffNum*** As Long, ***fpDataBuff*** as Integer, ByVal ***lStartPoint*** As Long, ByVal ***lCount*** As Long) As Long

LHLD *lhld*	handle of the LDSD
long *lBuffNum*	specifies the buffer number
int _far * *fpDataBuff*	source address of the data buffer
long *lStartPoint*	starting point in the destination buffer to be set
long *lCount*	number of samples

The **AL_SetBuffer** function copies the user *fpDataBuff* source buffer to the LDSD destination buffer specified by *lBuffNum*.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBuffNum	Specifies the ID number of the LDSD buffer to be copied.
fpDataBuff	Specifies the user buffer source address.
lStartPoint	Starting sample point in the LDSD source buffer to be set.
lCount	Number of *fpDataBuff* samples to be copied.

Returns:
On success the specified LDSD data buffer is updated with the specified *fpDataBuff* data samples, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5532TTL: DOT0, DOT1, DOT2, DOT3

40.6 AL_GetBufferStatus

Prototype **C\C++**

ERRNUM AL_GetBufferStatus(LHLD ***lhld***,
LPDATABUFFSTATUS lpDataBuffStat,
long ***lBuffNum***);

Visual Basic for Windows
Function AL_ GetBufferStatus(ByVal ***lhld*** As Long,
lpDataBuffStat As LPDATABUFFSTATUS,
ByVal ***lBuffNum*** As Long) As Long

LHLD *lhld* handle of the LDSD

LPDATABUFFSTATUS *lpDataBuffStat* address of the **LPDATABUFFSTATUS** structure

long *lBuffNum* specifies the buffer number

The **AL_ GetBufferStatus** function retrieves a buffer's current status.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpDataBuffStat	Points to the data buffer status structure to be filled.
lBuffNum	Specifies the ID number of the buffer from which the status information is obtained. • DONE_BUFFER Gets the next available buffer

Returns:
On success *lpDataBuffStat* is set with the specified *lBuffNum* or the next available buffer status, otherwise ERRNUM contains the last error code that occurred during the call. If *lBuffNum* is set to DONE_BUFFER a return value of 0 indicates the buffer is incomplete, a return value of 1 indicates the buffer is complete.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.7 AL_SetBufferDoneHandler

Prototype **C\C++**

ERRNUM AL_SetBufferDoneHandler(LHLD ***lhld***, LPSTR ***lpstrMethod***)

Visual Basic for Windows

Function AL_SetBufferDoneHandler(ByVal ***lhld*** As Long,
ByVal ***lpstrMethod*** As String) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMethod* address of the notification method string

The **AL_SetBufferDoneHandler** function sets the buffer done handler method of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMethod	Points to the desired notification method. The available settings are defined in the ADLIB include file as follows: • AL_CHECK_BUFFER — Buffers are polled for completion • AL_POSTMESSAGE — ADLIB calls a user function when a buffer is completed. • AL_POSTMESSAGE_PARAMS — ADLIB calls a user function with the user specified lparam and wparam set by the AL_SetBufferDoneHandlerParams function when a buffer is completed.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2 and DIN3
5532TTL: DOT0, DOT1, DOT2 and DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0

40.8 AL_GetBufferDoneHandler

Prototype **C\C++**

ERRNUM AL_GetBufferDoneHandler(LHLD ***lhld***, LPSTR ***lpstrMethod***, long ***lMaxLength);***

Visual Basic for Windows

Function AL_GetBufferDoneHandler(ByVal ***lhld*** As Long,
ByVal ***lpstrMethod*** As String,
ByVal ***lMaxLength*** As Long) As Long

LHLD *lhld* handle of the LDSD
LPSTR *lpstrMethod* destination address of the notification method string
long *lMaxLength* the count in bytes to be copied

The **AL_GetBufferDoneHandler** function returns the buffer done handler method.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpstrMethod	Points to the destination string for the copy.
lMaxLength	Maximum length in bytes to be copied to the user method string, including the NULL termination character.

Returns:
On success ERRNUM is set to 1, *lpstrMethod* is set to the LDSD done buffer handler method. Otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.9 AL_SetBufferDoneHandlerMsg

Prototype **C\C++**

ERRNUM AL_SetBufferDoneHandlerMsg(LHLD ***lhld***, HWND ***hwnd***, UINT ***uiMsg***);

Visual Basic for Windows
Function AL_SetBufferDoneHandlerMsg(ByVal ***lhld*** As Long,
ByVal ***hwnd*** As Integer,
ByVal ***uiMsg*** As Integer) As Long

LHLD *lhld* handle of the LDSD
HWND hwnd handle of the destination window
UINT uiMsg message to post

The **AL_SetBufferDoneHandlerMsg** function sets the buffer done handler message window and message identifier in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
hwnd	Identifies the window to which the message will be posted.
uiMsg	Specifies the message to be posted.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.10 AL_SetBufferDoneHandlerParams

Prototype **C\C++**

ERRNUM AL_SetBufferDoneHandlerParams(LHLD ***lhld***, WPARAM ***wparam***, LPARAM ***wparam***)

Visual Basic for Windows

Function AL_SetBufferDoneHandlerParams (ByVal ***lhld*** As Long, ByVal ***wparam*** As Long, ByVal ***lparam*** As Long) As Long

LHLD *lhld*	handle of the LDSD
WPARAM *wparam*	value of the windows wparam parameter to be posted.
LPARAM *lparam*	value of the windows lparam parameter to be posted.

The **AL_SetBufferDoneHandlerParams** function sets the wparam and lparam parameters to be posted to a user application when the ADLIB function AL_SetBufferDoneHandler is set to AL_POSTMESSAGE_PARAMS.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
wparam	Actual WPARAM value to be posted:
lparam	Actual LPARAM value to be posted:

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2 and DIN3
5532TTL: DOT0, DOT1, DOT2 and DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.11 AL_GetBufferDoneHandlerMsg

Prototype **C\C++**

ERRNUM AL_GetBufferDoneHandlerMsg(LHLD ***lhld***, LPWORD ***lpwordHwnd***, LPWORD ***lpwordMsg***);

Visual Basic for Windows

Function AL_GetBufferDoneHandlerMsg(ByVal ***lhld*** As Long,
lpwordHwnd As Integer,
lpwordMsg As Integer) As Long

LHLD *lhld*	handle of the LDSD
LPWORD	lpwordHwnddestination address for the window handle
LPWORD	lpwordMsgdestination address for the window message ID

The **AL_GetBufferDoneHandlerMsg** function returns the current buffer done message handler window handle and message ID of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpwordHwnd	Points to the destination address for window handle.
lpwordMsg	Points to the destination address for posting message ID.

Returns:
On success ERRNUM is set to 1, lpwordHwnd is set to the LDSD done buffer window identifier and lpwordMsg points to the message ID number. Otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.12 AL_SetNumOfBuffers

Prototype **C\C++**

ERRNUM AL_SetNumOfBuffers(LHLD ***lhld***, long ***lNumBuffers***);

Visual Basic for Windows
Function AL_SetNumOfBuffers(ByVal ***lhld*** As Long,
ByVal ***lNumBuffers*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lNumBuffers* specifies the quantity of buffers

The **AL_SetNumOfBuffers** function sets the number of buffers in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lNumBuffers	Specifies the quantify of buffers required for a hardware subsystem.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.13 AL_GetNumOfBuffers

Prototype **C\C++**

STATUS AL_GetNumOfBuffers(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetNumOfBuffers(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetNumOfBuffers** function retrieves the number of buffers in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.

Returns:
On success STATUS contains the number of buffers currently set in the LDSD for the specified device, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.14 AL_SetBufferSize

Prototype **C\C++**

ERRNUM AL_SetBufferSize(LHLD ***lhld***, long ***lBuffSize***);

Visual Basic for Windows
Function AL_SetBufferSize(ByVal ***lhld*** As Long, ByVal ***lBuffSize*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lBuffSize* specifies the size of buffer(s)

The **AL_SetBufferSize** function sets the size of buffer(s) in samples in the LDSD.

The actual size in bytes allocated is a factor of the devices sample size. If a given LDSD requires 2 bytes per sample and *lBuffSize* is set to 1000, 2000 bytes would be allocated for each buffer on behalf of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBuffSize	Specifies the size of buffer(s) required for a hardware subsystem.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.15 AL_GetBufferSize

Prototype **C\C++**

STATUS AL_GetBufferSize(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetBufferSize(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetBufferSize** function retrieves the current of buffer size in the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.

Returns:
On success STATUS contains the size of buffer(s) in samples currently set in the LDSD for the specified device, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5600 Series ACI, DCI, CCI, TTL: DIN0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0, DAC0, DAC1

40.16 AL_SetAutoInitBuffers

Prototype **C\C++**

ERRNUM AL_SetAutoInitBuffers(LHLD ***lhld***, long ***lState***)

Visual Basic for Windows
Function AL_ SetAutoInitBuffers(ByVal ***lhld*** As Long, ByVal ***lState*** As long) As Long

LHLD *lhld* handle of the LDSD

long *lState* desired state of the LDSD AutoInitBuffer flag

The **AL_SetAutoInitBuffers** function sets the LDSD buffer's auto initialization authorization. Essentially this function affects whether the *AL_StartDevice* function is allowed to auto initialize all LDSD buffers. This is most useful when a AL_SINGLE_CYCLE operation has completed operation and another acquisition is desired using the same LDSD buffers.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lState	Specifies the desired AutoInitBuffers authorization state of the LDSD. The available settings are defined in the ADLIB include file as follows: • YES The buffers will be auto initialized on *AL_InitDevice calls.* • NO The buffers will **NOT** be auto initialized on *AL_InitDevice* calls.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
All LDSD devices that require buffers.

40.17 AL_GetAutoInitBuffers

Prototype **C\C++**

STATUS AL_GetAutoInitBuffers(LHLD ***lhld***)

Visual Basic for Windows
Function AL_ GetAutoInitBuffers(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetAutoInitBuffers** function retrives the LDSD buffer's auto initialization authorization flag.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.

Returns:
On success STATUS is set to state of the LDSD AutoInitBuffers flag, otherwise STATUS contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
All LDSD devices that require buffers.

40.18 AL_SetPackedData

Prototype **C\C++**

ERRNUM AL_SetPackedData(LHLD ***lhld***, long ***lState***);

Visual Basic for Windows
Function AL_SetPackedData(ByVal ***lhld*** As Long, ByVal ***lState*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lState* specifies the FIFO state

The **AL_SetPackedData** function enables the hardware to pack two data points into each transfer. Essentially this function affects whether a device subsystem transfers one or two data points at a time. Some devices may place their associated channel information in place of the second data point when PackedData mode is ENABLED.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lState	Specifies the Packed FIFO state.

- DISABLED — The device subsystem transfers two data points.transfers.
- ENABLED — The device subsystem transfers one data point, for the PCI-55XX Series cards, the hardware contains the channel/gain/CJ information for each channel

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx: ADC0, DAC0, DAC1, Options: ENABLED, DISABLED

40.19 AL_GetPackedData

Prototype **C\C++**

STATUS AL_GetPackedData(LHLD ***lhld***);

Visual Basic for Windows

Function AL_GetPackedData(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetPackedData** function retrieves the current hardware packed data setting of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success STATUS is set to either ENABLED or DISABLED, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
Pci55xx: ADC0, DAC0, DAC1

41. DATA FORMATTING FUNCTIONS:

41.1 AL_DemuxData

Prototype **C\C++**

(16 Bit) ERRNUM AL_DemuxData(LHLD ***lhld***, LPBUFFSTRUCT ***lpbuffstruct***, float _far * ***DestBuffer***, long ***lLength***);

(32 Bit) ERRNUM AL_DemuxData(LHLD ***lhld***, LPBUFFSTRUCT ***lpbuffstruct***, float * ***DestBuffer***, long ***lLength***);

Visual Basic for Windows
Not supported

LHLD *lhld*	handle of the LDSD
LPBUFFSTRUCT *lpbuffstruct*	input address to the ADLIB buffer
float _far * *DestBuffer*	destination address to the user buffer
long *lLength*	length of the destination buffer in samples

The **AL_DemuxData** function converts the selected ADLIB buffer pointed to by *lpbuffstruct* to Engineering voltage units, placing each samples voltage into the destination buffer pointed to by *fpfDestBuffer*.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lpbuffstruct	points to the ADLIB buffer to demux.
fpfDestBuffer	points to the destination buffer.
lLength	specifies the length in samples of the destination buffer.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0

41.2 AL_DemuxDataSet

Prototype **C\C++**

(16 Bit) ERRNUM AL_DemuxDataSet(LHLD ***lhld***, long ***lBuffNum***,
float _far * ***DestBuffer***, long ***lStartPoint,*** long ***lCount***);

(32 Bit) ERRNUM AL_DemuxDataSet(LHLD ***lhld***, long ***lBuffNum***,
float * ***DestBuffer***, long ***lStartPoint,*** long ***lCount***);

Visual Basic for Windows
Function AL_DemuxDataSet(ByVal ***lhld*** As Long, ByVal ***lBuffNum*** As Long,
DestBuffer As Single, ByVal ***lStartPoint*** As Long,
ByVal ***lCount*** As Long) As Long

LHLD *lhld*	handle of the LDSD
long *lBuffNum*	specifies the buffer number
float _far * *DestBuffer*	destination address to the user buffer
long *lStartPoint*	ADLIB buffer starting point
long *lCount*	conversion count

The **AL_DemuxDataSet** function converts the selected ADLIB buffer selected by *lBuffNum* to Engineering voltage units, placing each samples voltage into the user destination buffer pointed to by *fpfDestBuffer*.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lBuffNum	Specifies the buffer ID number.
fpfDestBuffer	Points to the destination buffer.
lStartPoint	Specifies the starting point in the ADLIB buffer for conversion.
lCount	Specifies the number of samples to be converted.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
Pci55xx Series: ADC0

41.3 AL_TcTemp

Prototype **C\C++**

(16 Bit) ERRNUM AL_TcTemp(float _far * ***Buffer***, long ***lLength***,
WORD ***Type***, WORD ***Units***);

(32 Bit) ERRNUM AL_TcTemp(float * ***Buffer***, long ***lLength***,
WORD ***Type***, WORD ***Units***);

Visual Basic for Windows
Function AL_TcTemp(***Buffer*** As Single, ByVal ***lLength*** As Long,
ByVal ***Type*** As Integer, ByVal ***Units*** As Integer) As Long

float _far * Buffer input address of the buffer to be coverted
long lLength length of the input buffer in samples
WORD Type specifies the thermocouple type

WORD Units specifies the desired engineering units

The **AL_TcTemp** function converts the selected buffer pointed to by fpfBuffer to the specified temperature units.

Parameter	Description
Buffer	Points to the destination buffer.
lLength	Specifies the length in samples of the input buffer.
Type	Specifies the thermocouple type to be converted, the avaliable settings are defined in the adlib include file as follows: BNBS RNBS SNBS KNBS ENBS TNBS JNBS NNBS TDIN JDIN CHOS
Units	Specifies the desired temperature units type to be converted, the available settings are defined in the adlib include file as follows: • Deg_C Celsius degree • Deg_F Fahrenheit degree • Deg_K Kelvin degree • Deg_R Rankine degree

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:

5400 Series: ADC0, Options:	As specified above
5508SCi: ADC0, Options:	As specified above
5508TC: ADC0, Options:	As specified above
5525/50MF: ADC0, Options:	As specified above
5800 Series: ADC0, Options:	As specified above
4012AD Series: ADC0, Options:	As specified above
4112AD Series: ADC0, Options:	As specified above
Pci55xx Series: ADC0, Options:	As specified above

41.4 AL_RtdTemp

Prototype **C\C++**

(16 Bit) ERRNUM AL_RtdTemp(float _far * ***Buffer***, long ***lLength***, WORD ***Units***);

(32 Bit) ERRNUM AL_RtdTemp(float * ***Buffer***, long ***lLength***, WORD ***Units***);

Visual Basic for Windows

Function AL_RtdTemp(***Buffer*** As Single, ByVal ***lLength*** As Long,
ByVal ***Units*** As Integer) As Long

float _far * Buffer	input address of the buffer to be coverted
long lLength	length of the input buffer in samples
WORD Units	specifies the desired engineering units

The **AL_RtdTemp** function converts the selected buffer pointed to by fpfBuffer to the specified temperature units.

Parameter	Description
Buffer	Points to the destination buffer.
lLength	Specifies the length in samples of the input buffer.
Units	Specifies the desired temperature units type to be converted, the available settings are defined in the adlib include file as follows: • Deg_C Celsius degree • Deg_F Fahrenheit degree • Deg_K Kelvin degree • Deg_R Rankine degree

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
5508RTD: ADC0, Options: As specified above
4012AD with CS Options: As specified above

42. ERROR HANDLING FUNCTIONS

42.1 AL_SetErrOnReInitRunning

Prototype **C\C++**

ERRNUM AL_SetErrOnReInitRunning(LHLD ***lhld***, long ***lState***);

Visual Basic for Windows
Function AL_SetErrOnReInitRunning(ByVal ***lhld*** As Long,
ByVal ***lState*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lState* specifies the re-initialization state

The **AL_SetErrOnReInitRunning** function sets the hardware re-initialization authorization of the LDSD. Essentially this function affects whether the *AL_InitDevice* function is allowed to initialize a hardware device while the device itself is in a running state.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lState	Specifies the re-initialization authorization state of the LDSD. The available settings are defined in the ADLIB include file as follows:

- ALLOW_REINIT — The device will first be stopped and then reinitialized to the current settings.
- DISALLOW_REINIT — Reinitialization of the device will not be allowed when the device is running and will return an error.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
All Boards

42.2 AL_GetErrOnReInitRunning

Prototype **C\C++**

STATUS AL_GetErrOnReInitRunning(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetErrOnReInitRunning(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetErrOnReInitRunning** function retrieves the current hardware re-initialization authorization status of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success STATUS is set to either ALLOW_REINIT or DISALLOW_REINIT, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
All Boards

42.3 AL_SetErrOnReleaseRunning

Prototype **C\C++**

ERRNUM AL_SetErrOnReleaseRunning (LHLD ***lhld***, long ***lState***);

Visual Basic for Windows
Function AL_SetErrOnReleaseRunning(ByVal ***lhld*** As Long,
ByVal ***lState*** As Long) As Long

LHLD *lhld* handle of the LDSD
long *lState* specifies the release on running state

The **AL_SetErrOnReleaseRunning** function sets the hardware release on running authorization of the LDSD. Essentially this function affects whether the *AL_ReleaseDevice* function is allowed to release a hardware device while the device itself is in a running state.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem.
lState	Specifies the release on running authorization state of the LDSD. The available settings are defined in the ADLIB include file as follows: • DISALLOW_RELEASE — The device will first be stopped and then released. • ALLOW_RELEASE — Release of the device will not be allowed when the device is running and will return an error.

Returns:
On success ERRNUM is set to 1, otherwise ERRNUM contains the last error code that occurred during the call.

Supported Logical Device Subsystems:
All Boards

42.4 AL_GetErrOnReleaseRunning

Prototype **C\C++**

STATUS AL_GetErrOnReleaseRunning(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetErrOnReleaseRunning(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetErrOnReleaseRunning** function retrieves the current hardware reinitialization authorization status of the LDSD.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the status is to be retrieved.

Returns:
On success STATUS is set to either ALLOW_RELEASE or DISALLOW_RELEASE, otherwise STATUS contains a negative error code that occurred during the call.

Supported Logical Device Subsystems:
All Boards

42.5 AL_GetBoardError

Prototype **C\C++**

ERRMUM AL_GetBoardError(LHLD ***lhld***);

Visual Basic for Windows
Function AL_GetBoardError(ByVal ***lhld*** As Long) As Long

LHLD *lhld* handle of the LDSD

The **AL_GetBoardError** function retrieves the last board driver hardware error code for the specified lhld.

Parameter	Description
lhld	Identifies the instance of the logical device subsystem from which the error is to be retrieved.

Returns:
On success ERRNUM contains the last error code that occurred, otherwise ERRNUM is set to zero (0) if no errors have occurred.

Supported Logical Device Subsystems:
5400 Series: ADC0
5500HR: ADC0
5500MF:ADC0
5504DA:DAC0
5508LC: ADC0
5508LF: ADC0
5508SCi: ADC0
5508SHR: ADC0
5508TC/BG/RTD: ADC0
5516DMA: ADC0
5525/50MF: ADC0
5532TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
5800 Series: ADC0
4012AD Series: ADC0
4112AD Series: ADC0
4608ACO: DOT0
4608DCO: DOT0
4616ACI: DIN0, DIN1
4616CCI: DIN0, DIN1
4616DCI: DIN0, DIN1
4616HCO: DIN0, DIN1
4616OII: DIN0, DIN1
4616RLY: DOT0
4632TTL: DIN0, DIN1, DIN2, DIN3, DOT0, DOT1, DOT2, DOT3
Pci55xx Series: ADC0, DAC0, DAC1

43. ADLIB ERROR CODES

ALERR_NOERRORS	1	No errors occurred during the call.
ALERR_NOT_SUPPORTED	-1	The specified function or argument is not supported.

MEMORY

ALERR_MEMORY_LOW	-100	memory allocation failed.
ALERR_DMA_MEMORY_LOW	-101	DMA memory allocation failed increase. ADACDMABUFFERSIZE in system.ini file.

GENERIC

ALERR_ARRAY_PTR	-200	Invalid array pointer passed to function.
ALERR_STRING_PTR	-201	Invalid string pointer passed to function.
ALERR_MAXSTRING	-202	Input string size exceeds MAX length.
ALERR_MAXARRAY	-203	Input array size exceeds MAX length.
ALERR_INVALID_STRINGLIST	-204	Input string format invalid.
ALERR_DESTINATION_STRLEN	-205	Input parameter destination string length is less than the source string length.
ALERR_MINARRAY	-206	Input array size exceeds MIN length.

BOARD STRUCTURE

ALERR_BOARD_STRUCT_PTR	-300	Invalid BOARD struct pointer.
ALERR_BOARD_ID	-301	No BOARD struct for the specified Board ID exist.
ALERR_BOARD_CAPSFILE_STRPTR	-302	The BOARD struct CAP's file pointer is invalid.
ALERR_BOARD_NOBOARDS	-303	No board configurations found in .con file.
ALERR_BOARD_MAX_ADCCAPS	-304	Max. ADC caps structures have been allocated.
ALERR_BOARD_MAX_DACCAPS	-305	Max. DAC caps structures have been allocated.
ALERR_BOARD_MAX_DINCAPS	-306	Max. DIN caps structures have been allocated.
ALERR_BOARD_MAX_DOTCAPS	-307	Max. DOT caps structures have been allocated.
ALERR_BOARD_MAX_CTRCAPS	-308	Max. CTR caps structures have been allocated.
ALERR_BOARD_MAXSTRING	-309	The specified BOARD string exceeds max. length.

LOGICAL DEVICE HANDLES

ALERR_LHLD	-400	The LHLD specified does not exist.
ALERR_LHLD_MAX	-401	The maximum LHLD have already been allocated.

LDS STRUCTURE

ALERR_LDS_STRUCT_PTR	-500	Invalid LDS struct pointer.
ALERR_INTERNAL_LDS_TYPE	-501	Unknown Logical Device Subsystem type.
ALERR_LDS_MAXSTRING	-502	The LDS type string specified in a call to adlib exceeds the maximum allowed length.
ALERR_LDS_MAXALLOCATED	-503	The maximum LDS have been allocated.
ALERR_LDS_NOCAPS	-504	No CAPS found.
ALERR_LDS_NOCAPSADC	-505	No CAPSADC found.
ALERR_LDS_NOCAPSDAC	-506	No CAPSDAC found.
ALERR_LDS_NOCAPSDIN	-507	No CAPSDIN found.
ALERR_LDS_NOCAPSDOT	-508	No CAPSDOT found.
ALERR_LDS_NOCAPSCTR	-509	No CAPSCTR found.
ALERR_LDS_NOTSUPPORTED	-510	The specified LDS is invalid.

CAPS STRUCTURE

ALERR_CAPS_TYPE	-600	Unknown Capabilities Subsystem type.
ALERR_CAPS_STRUCT_PTR	-601	Invalid CAPS struct pointer.
ALERR_CAPSADC_STRUCT_PTR	-602	Invalid CAPSADC struct pointer.
ALERR_CAPSDAC_STRUCT_PTR	-603	Invalid CAPSCTRIO struct pointer.
ALERR_CAPSDIGIO_STRUCT_PTR	-604	Invalid CAPSDIGIO struct pointer.
ALERR_CAPSCTRIO_STRUCT_PTR	-605	Invalid CAPSCTRIO struct pointer.

ENVIRONMENT STRUCTURE

ALERR_ENV_STRUCT_PTR	-700	Invalid ENV struct pointer.
ALERR_ENV_LOADED	-701	The Environment is already loaded.
ALERR_ENV_NOTLOADED	-702	The Environment is NOT loaded.
ALERR_ENV_MAXSTRING	-703	The specified Environment is too long.

OPTIONS STRUCTURE

ALERR_OPTION_STRUCT_PTR	-800	Invalid OPTIONS struct pointer.
ALERR_OPTION_STRPTR	-801	Invalid option string pointer.
ALERR_OPTION_STRING	-802	Invalid option string format.
ALERR_OPTION_NAME_STRPTR	-803	Invalid option string name pointer.
ALERR_OPTION_NAME_STRING	-804	Invalid option string name.
ALERR_OPTION_ID_STRING	-805	Invalid option string ID.
ALERR_OPTION_CONFIG_TYPE	-806	Invalid option string configuration type.
ALERR_OPTION_MAXSTRING	-807	The option string exceeds the maximum allowed length.

FILE I/O

ALERR_FILE_FOUND	-900	The specified file can not be found.

INI FILE

ALERR_INI_FILEEXIST	-1000	The INFO file (.INI) can not be found.
ALERR_INI_PATH_STRPTR	-1001	Invalid .INI string pointer.
ALERR_INI_SECTION_STRPTR	-1002	Invalid .INI [SECTION] string pointer.
ALERR_INI_ENTRY_STRPTR	-1003	Invalid .INI ENTRY string pointer.
ALERR_INI_SECTION_UNKNOWN	-1004	Unknown .INI [SECTION].
ALERR_INI_BOARDID_RANGE	-1005	Invalid .INI BoardId setting.
ALERR_INI_DTM_STRING	-1006	Invalid .INI DataTransMethod setting.
ALERR_INI_CM_STRING	-1007	Invalid .INI CycleMode setting.
ALERR_INI_DMAMODE_STRING	-1008	Invalid .INI DmaMode setting.
ALERR_INI_DMACHAN_STRING	-1009	Invalid .INI DmaChan setting.
ALERR_INI_IRQMODE_STRING	-1010	Invalid .INI IrqMode setting.
ALERR_INI_IRQLEVEL_STRING	-1011	Invalid .INI IrqLevel setting.
ALERR_INI_ARBREINIT_STRING	-1012	Invalid .INI ArbReinit setting.
ALERR_INI_LOGERRORS_STRING	-1013	Invalid .INI LogErrors setting.
ALERR_INI_ERRONBUFFOVERRUN_STRING	-1014	Invalid .INI ErrorOnBufferOverrun setting.
ALERR_INI_ERRONRELRUNNING_STRING	-1015	Invalid .INI ErrorOnReleaseRunning setting.
ALERR_INI_BUFFERSIZE_LOW	-1016	Invalid .INI BufferSize setting.
ALERR_INI_NUMBUFFER_LOW	-1017	Invalid .INI NumBuffers setting.
ALERR_INI_BUFFERNOTIFYMETHOD_STRING	-1018	Invalid .INI BufferNotificationMethod setting.
ALERR_INI_AUTOINITBUFFERS_STRING	-1019	Invalid .INI AutoInitBuffers setting.
ALERR_INI_ERRONTRIGGEROVERRUN_STRING	-1020	Invalid .INI ErrOnTriggerOverrun setting.

INI FILE - continued

ALERR_INI_MINSTARTCHAN_STRING	-1100	Invalid .INI MinStartChan setting.
ALERR_INI_MAXENDCHAN_STRING	-1101	Invalid .INI MaxEndChan setting.
ALERR_INI_SIGNALPATH_STRING	-1102	Invalid .INI SignalPath setting.
ALERR_INI_INPUTCONFIG_STRING	-1103	Invalid .INI InputConfig setting.
ALERR_INI_CLKMODE_STRING	-1104	Invalid .INI ClkMode setting.
ALERR_INI_CLKSOURCE_STRING	-1105	Invalid .INI ClkSource setting.
ALERR_INI_CLKSOURCESIGNAL_STRING	-1106	Invalid .INI ClkSourceSignal setting.
ALERR_INI_CLKRATE_STRING	-1107	Invalid .INI ClkRate setting.
ALERR_INI_CLKRATE_RANGE	-1108	Invalid .INI ClkRate range.
ALERR_INI_CLKRATEUNITS_STRING	-1109	Invalid .INI ClkRateUnits setting.
ALERR_INI_BURSTLENGTH_STRING	-1110	Invalid .INI BurstLength setting.
ALERR_INI_BURSTLENGTH_RANGE	-1111	Invalid .INI BurstLength range.
ALERR_INI_BURSTRATE_STRING	-1112	Invalid .INI BurstRate setting.
ALERR_INI_BURSTRATE_RANGE	-1113	Invalid .INI BurstRate range.
ALERR_INI_BURSTRATEUNITS_STRING	-1114	Invalid .INI BurstRateUnits setting.
ALERR_INI_TRIGMODE_STRING	-1115	Invalid .INI TrigMode setting.
ALERR_INI_TRIGSRC_STRING	-1116	Invalid .INI TrigSource setting.
ALERR_INI_TRIGSRCSIGNAL_STRING	-1117	Invalid .INI TrigSourceSignal setting.
ALERR_INI_TRIGRATE_STRING	-1118	Invalid .INI TrigRate setting.
ALERR_INI_TRIGRATE_RANGE	-1119	Invalid .INI MinTrigRate or MaxTrigRate setting.
ALERR_INI_TRIGRATEUNITS_STRING	-1120	Invalid .INI TrigRateUnits setting.
ALERR_INI_DATACODE_STRING	-1121	Invalid .INI DataCode setting.
ALERR_INI_DATAOFFSET_STRING	-1122	Invalid .INI DataOffset setting.
ALERR_INI_DATASPAN_STRING	-1123	Invalid .INI DataSpan setting.
ALERR_INI_DATARANGE_STRING	-1124	Invalid .INI DataRange setting.
ALERR_INI_OUTPUTCONFIG_STRING	-1125	Invalid .INI OutputConfig setting.

INI FILE - continued

ALERR_INI_EXPPANEL_STRING	-1126	Invalid .INI ExpansionPanel n setting.
ALERR_INI_POSTSAMPLECOUNT_STRING	-1127	Invalid .INI PostSampleCounts setting.
ALERR_INI_POSTSAMPLECOUNT_RANGE	-1128	Invalid .INI PostSampleCounts range.
ALERR_INI_GATESRC_STRING	-1129	Invalid .INI GatrSource setting.
ALERR_INI_GATESRCLEVEL_STRING	-1130	Invalid .INI GateSourceLevel setting.
ALERR_INI_BURSTMODE_ STRING	-1131	Invalid .INI BurstMode setting.
ALERR_INI_FILTERTYPE_ STRING	-1132	Invalid .INI Filtertype setting.
ALERR_INI_FILTERFREQ_ STRING	-1133	Invalid .INI FilterFreq setting.
ALERR_INI_ FILTERFREQ _ RANGE	-1134	Invalid .INI FilterFreq range.
ALERR_INI_HANDSHAKE_ STRING	-1135	Invalid .INI HandShake setting.
ALERR_INI_PORTRESOLUTION_ STRING	-1136	Invalid .INI PortResolution setting.
ALERR_INI_ PORTMASK _ STRING	-1137	Invalid .INI PortMask setting.
ALERR_INI_ PORTRMASK_ RANGE	-1138	Invalid .INI PortMask range.
ALERR_INI_ CTRMODE_ STRING	-1139	Invalid .INI CtrMode setting.
ALERR_INI_ CTRRATEUNITS_ STRING	-1140	Invalid .INI CtrRateUnits setting.
ALERR_INI_ CTRRATE_ STRING	-1141	Invalid .INI CtrRate setting.
ALERR_INI_ PACKEDDATA_ STRING	-1142	Invalid .INI PackedData setting.
ALERR_INI_ CLKOUTPUT_ STRING	-1143	Invalid .INI ClockOutput setting.
ALERR_INI_ TRIGOUTPUT _ STRING	-1144	Invalid .INI TrigOutput setting.

CAPABILITIES FILE

ALERR_CAPS_FILEEXIST	-2000	The Capabilities (.CAP) can not be found.
ALERR_CAPS_MFG_STRING	-2001	Invalid .CAP manufacture setting.
ALERR_CAPS_MODEL_STRING	-2002	Invalid .CAP model setting.

BOARD

ALERR_CAPS_IOBASEADDRSEL_STRING	-2100	Invalid .CAP IoBaseAddrSelect setting.
ALERR_CAPS_MINIOBASEARRR_STRING	-2101	Invalid .CAP MinIoBaseAddress setting.
ALERR_CAPS_MAXIOBASEARRR_STRING	-2102	Invalid .CAP MaxIoBaseAddress setting.

DRIVER

ALERR_CAPS_DRIVERVERSION_STRING	-2200	Invalid .CAP DriverVersion setting.
ALERR_CAPS_DRIVERNAME_STRING	-2201	Invalid .CAP DriverName setting.
ALERR_CAPS_BOARDVERSION_STRING	-2202	Invalid .CAP BoardVersionSupport setting.
ALERR_CAPS_BOARDIDSUPPORT_STRING	-2203	Invalid .CAP BoardVersionId setting.
ALERR_CAPS_VERSION_STRING	-2204	Invalid .CAP file Version setting.

COMMON DEVICE SETTINGS

ALERR_CAPS_IRQSHAREABLE_STRING	-2300	Invalid .CAP IrqShareable setting.
ALERR_CAPS_DTM_STRING	-2301	Invalid .CAP DataTransMethod setting.
ALERR_CAPS_DTM_ENTRY	-2302	CAP DataTransMethod entry does not exist.
ALERR_CAPS_CM_STRING	-2303	Invalid .CAP CycleMode setting.
ALERR_CAPS_CM_ENTRY	-2304	CAP CycleMode entry does not exist.
ALERR_CAPS_BUFFERNOTIFYMETHOD_STRING	-2305	Invalid .CAP buffer notification setting.
ALERR_CAPS_DMAMODE_STRING	-2307	Invalid .CAP DmaMode setting.
ALERR_CAPS_DMACHAN_STRING	-2308	Invalid .CAP DmaChan setting.
ALERR_CAPS_IRQMODE_STRING	-2309	Invalid .CAP IrqMode setting.
ALERR_CAPS_IRQLEVEL_STRING	-2310	Invalid .CAP IrqLevel setting.
ALERR_CAPS_BUFFERSUPPORT_STRING	-2311	Invalid .CAP BoardVersionId setting
ALERR_CAPS_MINBUFFERS_STRING	-2312	Invalid .CAP MinStartChan setting.
ALERR_CAPS_MAXBUFFERS_STRING	-2313	Invalid .CAP MaxEndChan setting.
ALERR_CAPS_MINBUFFERSIZE_STRING	-2314	Invalid .CAP Min buffer size setting.
ALERR_CAPS_MAXBUFFERSIZE_STRING	-2315	Invalid .CAP Max buffer size setting.
ALERR_CAPS_PACKEDFIFO_STRING	2316	Invalid .CAP PackedFifoSupport setting.

ADC

ALERR_CAPS_GAIN_STRING	-2351	Invalid .CAP GAIN string format.
ALERR_CAPS_MINSTARTCHAN_STRING	-2352	Invalid .CAP MinStartChan setting.
ALERR_CAPS_MAXENDCHAN_STRING	-2353	Invalid .CAP MaxEndChan setting.
ALERR_CAPS_ARBCHAN_STRING	-2354	Invalid .CAP ArbChan setting.
ALERR_CAPS_ARBGAIN_STRING	-2355	Invalid .CAP ArbGain setting.
ALERR_CAPS_CLKMODE_STRING	-2356	Invalid .CAP ClkMode setting.
ALERR_CAPS_CLKSOURCE_STRING	-2357	Invalid .CAP ClkSource setting.
ALERR_CAPS_CLKSOURCESIGNAL_STRING	-2358	Invalid .CAP ClkSourceSignal setting.
ALERR_CAPS_MINCLKRATE_STRING	-2359	Invalid .CAP MinClkRate setting.
ALERR_CAPS_MAXCLKRATE_STRING	-2360	Invalid .CAP MaxClkRate setting.
ALERR_CAPS_MINBURSTLENGTH_STRING	-2361	Invalid .CAP MinBurstLength setting.
ALERR_CAPS_MAXBURSTLENGTH_STRING	-2362	Invalid .CAP MaxBurstLength setting.
ALERR_CAPS_MINBURSTRATE_STRING	-2363	Invalid .CAP MinBurstRate setting.
ALERR_CAPS_MAXBURSTRATE_STRING	-2364	Invalid .CAP MaxBurstRate setting.
ALERR_CAPS_TRIGMODE_STRING	-2365	Invalid .CAP TrigMode setting.
ALERR_CAPS_TRIGSRC_STRING	-2366	Invalid .CAP TrigSource setting.
ALERR_CAPS_TRIGSRCSIGNAL_STRING	-2367	Invalid .CAP TrigSourceSignal setting.
ALERR_CAPS_MINTRIGRATE_STRING	-2368	Invalid .CAP MinTrigRate setting.
ALERR_CAPS_MAXTRIGRATE_STRING	-2369	Invalid .CAP MaxTrigRate setting.
ALERR_CAPS_CJ_STRING	-2370	Invalid .CAP CJ setting.
ALERR_CAPS_ARBCJ_STRING	-2371	Invalid .CAP ArbCj setting.
ALERR_CAPS_DATACODE_STRING	-2372	Invalid .CAP DataCode setting.
ALERR_CAPS_DATAOFFSET_STRING	-2373	Invalid .CAP DataOffset setting.
ALERR_CAPS_DATASPAN_STRING	-2374	Invalid .CAP DataSpan setting.
ALERR_CAPS_FIFOSIZE_STRING	-2375	Invalid .CAP FifoSize setting.
ALERR_CAPS_DATARANGE_STRING	-2376	Invalid .CAP DataRange setting.

ADC - continued

ALERR_CAPS_OUTPUTCONFIG_STRING	-2377	Invalid .CAP OutputConfig setting.
ALERR_CAPS_MAXEXPPANELS_STRING	-2378	Invalid .CAP MaxNumExpPanels setting.
ALERR_CAPS_EXPPANEL_STRING	-2379	Invalid .CAP ExpPanel setting.
ALERR_CAPS_MAXEXPPANELS_MAX	-2380	Invalid .CAP MaxNumExpPanels too High.
ALERR_CAPS_BYTEPERSMPL_STRING	-2381	Invalid .CAP BytesPerSample setting.
ALERR_CAPS_MINPOSTSAMPLE_STRING	-2382	Invalid .CAP MinPostSamples setting.
ALERR_CAPS_MAXPOSTSAMPLE_STRING	-2383	Invalid .CAP MaxPostSamples setting.
ALERR_CAPS_GATESRC_STRING	-2384	Invalid .CAP GateSource setting.
ALERR_CAPS_GATESRCLEVEL_STRING	-2385	Invalid .CAP GateSourceLevel setting.
ALERR_CAPS_BURSTMODE_STRING	-2386	Invalid .CAP BurstMode setting.
ALERR_CAPS_SIGNALPATH_STRING	-2387	Invalid .CAP SignalPath setting.
ALERR_CAPS_INPUTCONFIG_STRING	-2388	Invalid .CAP InputConfig setting.
ALERR_CAPS_DATAMASK_STRING	-2389	Invalid .CAP DataMask setting.
ALERR_CAPS_FILTERTYPE_STRING	-2390	Invalid .CAP FilterType setting.
ALERR_CAPS_MINFILTERFREQ_STRING	-2391	Invalid .CAP MinFilterFreq setting.
ALERR_CAPS_MAXFILTERFREQ_STRING	-2392	Invalid .CAP MaxFilterFreq setting.
ALERR_CAPS_HANDSHAKE_STRING	-2393	Invalid .CAP HandShake setting.
ALERR_CAPS_PORTRESOLUTION_STRING	-2394	Invalid .CAP PortResolution setting.
ALERR_CAPS_MINPORTMASK_STRING	-2395	Invalid .CAP MinPortMask setting.
ALERR_CAPS_MAXPORTMASK_STRING	-2396	Invalid .CAP MaxPortMask setting.
ALERR_CAPS_CLKOUTPUT_STRING	- 2397	Invalid .CAP ClockOutput setting.
ALERR_CAPS_TRIGOUTPUT_STRING	- 2398	Invalid .CAP TrigOutput setting.
ALERR_CAPS_INPUTCONFIG_STRING	- 2399	Invalid .CAP InputConfig setting.
ALERR_CAPS_DATAOFFSET_STRING	- 2400	Invalid .CAP DataOffset setting.
ALERR_CAPS_CTRMODE_STRING	- 2401	Invalid .CAP CtrMode setting.
ALERR_CAPS_MINRATE_STRING	- 2402	Invalid .CAP MinRate setting.

ADC - continued

ALERR_CAPS_MAXRATE_STRING	-	2403	Invalid .CAP MaxRate setting.

CONFIGURATION FILE

ALERR_CON_FILEEXIST	-3000	The Configuration (.CON) can not be found.
ALERR_CON_FILE_MAXSTRING	-3001	The (.CON) file string exceeds the maximum allowed length.
ALERR_CON_FILE_STRPTR	-3002	Invalid .CON file string pointer.
ALERR_CON_BOARDSECTION_STRPTR	-3003	Invalid .CON board section pointer.
ALERR_CON_BOARDSECTION_STRING	-3004	Invalid .CON board section string
ALERR_CON_BOARDID_SAME	-3005	Invalid .CON BoardId setting.
ALERR_CON_BOARDID_RANGE	-3006	Invalid .CON BoardId setting.
ALERR_CON_IOBASEADDR_STRING	-3007	Invalid .CON IoBaseAddr setting.
ALERR_CON_BOARDCAPSFILE_STRING	-3008	Invalid .CON capabilities file setting.
ALERR_CON_DRVPATH_STRING	-3009	Invalid .CON driver Path setting.
ALERR_SYSTEMDRV_STRING	-3010	Invalid .CON System driver setting.

DEVICE SUBSYSTEM CHANNELS

ALERR_CHANLIST_LISTTYPE	-4000	Invalid Channel list struct list lType, is it an Array or String list.
ALERR_CHANLIST_STRPTR	-4001	Invalid Channel list string pointer.
ALERR_CHANLIST_STRING	-4002	Invalid Channel list string.
ALERR_CHANLIST_MAXSTRLEN	-4003	Invalid Channel list string length.
ALERR_CHANLIST_ARRAYPTR	-4004	Invalid Channel list array pointer.
ALERR_CHANLIST_ARRAY_LENTHPTR	-4005	Invalid Channel list array length pointer.
ALERR_CHANLIST_MAXARRAYLEN	-4006	Invalid Channel list array length.
ALERR_CHANLIST_MINCHAN	-4007	Invalid Channel in list.
ALERR_CHANLIST_MAXCHAN	-4008	Invalid Channel in list.
ALERR_CHANLIST_NONSEQ	-4009	Non-sequential Channels are not supported.
ALERR_GLOBALGAIN_STRPTR	-4010	Invalid Global Gain string pointer.
ALERR_GLOBALGAIN_GAIN	-4011	Invalid Gain type.
ALERR_CHANGAINLIST_LISTTYPE	-4050	Invalid Channel list struct list lType, is it an Array or String list.
ALERR_CHANGAINLIST_STRPTR	-4051	Invalid Channel list string pointer.
ALERR_CHANGAINLIST_STRING	-4052	Invalid Channel list string.
ALERR_CHANGAINLIST_MAXSTRLEN	-4053	Invalid Channel list string length.
ALERR_CHANGAINLIST_ARRAYPTR	-4054	Invalid Channel list array pointer.
ALERR_CHANGAINLIST_ARRAY_LENTHPTR	-4055	Invalid Channel list array length pointer.
ALERR_CHANGAINLIST_MAXARRAYLEN	-4056	Invalid Channel list array length.
ALERR_CHANGAINLIST_MINCHAN	-4057	Invalid Channel in list.
ALERR_CHANGAINLIST_MAXCHAN	-4058	Invalid Channel in list.
ALERR_CHANGAINLIST_NONSEQ	-4059	Non-sequential Channels are not supported.
ALERR_CHANGAINLIST_SYNTAX	-4060	Invalid ChanGainList syntax.
ALERR_CHANGAINLIST_ARBGAINS	-4061	Aribitrary Gains are not supported.
ALERR_CHANGAINLIST_GAINTYPE	-4062	Invalid Gain type.

DEVICE SUBSYSTEM EXPANSION PANEL GAINS

ALERR_EXPPANELGAINLIST_LISTTYPE	-4100	Invalid ExpPanelGainList structure list lType.
ALERR_EXPPANELGAINLIST_STRPTR	-4101	Invalid ExpPanelGainList string pointer.
ALERR_EXPPANELGAINLIST_STRING	-4102	Invalid ExpPanelGainList string.
ALERR_EXPPANELGAINLIST_MAXSTRLEN	-4103	Invalid ExpPanelGainList string length.
ALERR_EXPPANELGAINLIST_GAINSETTING	-4104	Invalid ExpPanelGainList gain setting.
ALERR_EXPPANELGAINLIST_ARRAYPTR	-4105	Invalid ExpPanelGainList array pointer.
ALERR_EXPPANELGAINLIST_ARRAY_LENGTHPTR	-4106	Invalid ExpPanelGainList array length pointer.
ALERR_EXPPANELGAINLIST_MAXARRAYLEN	-4107	Invalid ExpPanelGainList array length.
ALERR_EXPPANELGAINLIST_MINCHAN	-4108	Invalid ExpPanelGainList channel.
ALERR_EXPPANELGAINLIST_MAXCHAN	-4109	Invalid ExpPanelGainList channel.
ALERR_EXPPANELGAINLIST_SYNTAX	-4110	Invalid ExpPanelGainList syntax.

DEVICE SUBSYSTEM INPUT CONFIGURATION LIST

ALERR_INPUTCONFIG_LISTTYPE	-4120	Invalid InputConfig structure list lType.
ALERR_ INPUTCONFIG _STRPTR	-4121	Invalid InputConfig string pointer.
ALERR_ INPUTCONFIG _STRING	-4122	Invalid InputConfig string.
ALERR_ INPUTCONFIG _MAXSTRLEN	-4123	Invalid InputConfig string length.
ALERR_ INPUTCONFIG _GAINSETTING	-4124	Invalid InputConfig gain setting.
ALERR_ INPUTCONFIG _ARRAYPTR	-4125	Invalid InputConfig array pointer.
ALERR_ INPUTCONFIG _ARRAY_LENGTHPTR	-4126	Invalid InputConfig array length pointer.
ALERR_ INPUTCONFIG _MAXARRAYLEN	-4127	Invalid InputConfig array length.
ALERR_ INPUTCONFIG _MINCHAN	-4128	Invalid InputConfig min channel specified.
ALERR_ INPUTCONFIG _MAXCHAN	-4129	Invalid InputConfig max channel specified.
ALERR_ INPUTCONFIG _SYNTAX	-4130	Invalid InputConfig list syntax
ALERR_GLOBAL INPUTCONFIG _STRPTR	-4131	Invalid global InputConfig string pointer.
ALERR_GLOBAL INPUTCONFIG_TYPE	-4132	Invalid global InputConfig type

DEVICE SUBSYSTEM DATA OFFSET LIST

ALERR_DATAOFFSET_LISTTYPE	-4140	Invalid DataOffset structure list lType.
ALERR_ DATAOFFSET _STRPTR	-4141	Invalid DataOffset string pointer.
ALERR_ DATAOFFSET _STRING	-4142	Invalid DataOffset string.
ALERR_ DATAOFFSET _MAXSTRLEN	-4143	Invalid DataOffset string length.
ALERR_ DATAOFFSET _ GAINSETTING	-4144	Invalid DataOffset gain setting.
ALERR_ DATAOFFSET _ARRAYPTR	-4145	Invalid DataOffset array pointer.
ALERR_ DATAOFFSET _ARRAY_LENGTHPTR	-4146	Invalid DataOffset array length pointer.
ALERR_ DATAOFFSET _MAXARRAYLEN	-4147	Invalid DataOffset array length.
ALERR_ DATAOFFSET _MINCHAN	-4148	Invalid DataOffset min channel specified.
ALERR_ DATAOFFSET _MAXCHAN	-4149	Invalid DataOffset max channel specified.
ALERR_ DATAOFFSET _SYNTAX	-4150	Invalid DataOffset list syntax

DEVICE SUBSYSTEM DATA OFFSET LIST (con't)

ALERR_GLOBAL DATAOFFSET _STRPTR	-4151	Invalid global DataOffset string pointer.
ALERR_GLOBAL DATAOFFSET _TYPE	-4152	Invalid global DataOffset type

DEVICE SUBSYSTEM EXPANSION PANEL

ALERR_EXPPANELLIST_LISTTYPE	-4200	Invalid ExpPanelList structure list lType.
ALERR_EXPPANELLIST_STRPTR	-4201	Invalid ExpPanelList string pointer.
ALERR_EXPPANELLIST_STRING	-4202	Invalid ExpPanelList string.
ALERR_EXPPANELLIST_MAXSTRLEN	-4203	Invalid ExpPanelList string length.
ALERR_EXPPANELLIST_SETTING	-4204	Invalid ExpPanelList gain setting.
ALERR_EXPPANELLIST_ARRAYPTR	-4205	Invalid ExpPanelList array pointer.
ALERR_EXPPANELLIST_ARRAY_LENGTHPTR	-4206	Invalid ExpPanelList array length pointer.
ALERR_EXPPANELLIST_MAXARRAYLEN	-4207	Invalid ExpPanelList array length.
ALERR_EXPPANELLIST_MAXPANELS	-4208	Invalid Number of ExpPanels specified.
ALERR_EXPPANELLIST_PANELTYPE	-4209	Invalid ExpPanelList panel type.

DEVICE SUBSYSTEM THERMOUCOUPLES

ALERR_CJLIST_LISTTYPE	-4401	Invalid CJ structure list lType.
ALERR_CJLIST_STRPTR	-4402	Invalid CJ string pointer.
ALERR_CJLIST_STRING	-4403	Invalid CJ string.
ALERR_CJLIST_MAXSTRLEN	-4404	Invalid CJ string length.
ALERR_CJLIST_ARRAYPTR	-4405	Invalid CJ array pointer.
ALERR_CJLIST_ARRAY_LENGTHPTR	-4406	Invalid CJ array length pointer.
ALERR_CJLIST_MAXARRAYLEN	-4407	Invalid CJ array length.
ALERR_CJLIST_ARBCJ	-4408	Arbitrary CJs are not supported.
ALERR_CJLIST_MINCHAN	-4409	Invalid CJ channel specified.
ALERR_CJLIST_MAXCHAN	-4410	Invalid CJ channel specified.
ALERR_CJLIST_CJTYPE	-4411	Invalid CJ type
ALERR_CJLIST_SYNTAX	-4412	Invalid CJ list syntax
ALERR_GLOBALCJ_STRPTR	-4413	Invalid global CJ string pointer.
ALERR_GLOBALCJ_CJTYPE	-4414	Invalid global CJ type

TRIGGERS

ALERR_TRIGGERMODE_UNSUPPORTED	-4500	Trigger Mode is not supported.
ALERR_TRIGGERMODE_OPTION	-4501	Specified trigger mode source is not supported.
ALERR_TRIGGER_RATELOW	-4502	Invalid trigger rate specified.
ALERR_TRIGGER_RATEHIGH	-4503	Invalid trigger rate specified.
ALERR_TRIGGER_MINPOSTSAMPLECOUNT	-4504	Invalid post sample count specified.
ALERR_TRIGGER_MAXPOSTSAMPLECOUNT	-4505	Invalid post sample count specified.
ALERR_TRIGGER_POSTSMPLCNT_BUFFSIZE	-4506	Specified Post sample counts > bufferSize.

TRIGGER SOURCES

ALERR_TRIGGERSOURCE_OPTION	-4507	Specified Trigger source is not supported.
ALERR_TRIGGERSOURCE_UNSUPPORTED	-4510	Trigger sources is not supported.

TRIGGER SIGNALS

ALERR_TRIGGERSOURCESIGNAL_UNSUPPORTED	-4508	Triggering is not supported.
ALERR_TRIGGERSOURCESIGNAL_OPTION	-4509	Specified trigger signal source is not supported.

TRIGGER OUTPUTS

ALERR_TRIGGERSOUTPUT_UNSUPPORTED	- 4511	Triggering Output is not supported.
ALERR_TRIGGERSOUTPUT_OPTION	-4512	Specified trigger Output is not supported.

CLOCKING

ALERR_CLOCKING_UNSUPPORTED	-4600	Clocking is not supported.
ALERR_CLOCKING_OPTION	-4601	Specified Clock option is not supported.
ALERR_CLOCKING_RATELOW	-4602	Invalid clock rate specified.
ALERR_CLOCKING_RATEHIGH	-4603	Invalid clock rate specified.

CLOCK SIGNALS

ALERR_CLOCKSIGNAL_UNSUPPORTED	-4604	Clocking is not supported.
ALERR_CLOCKSIGNAL_OPTION	-4605	Specified Clock option is not supported.

CLOCK OUTPUTS

ALERR_CLOCKSOUTPUT_UNSUPPORTED	-4606	ClockOutput is not supported.
ALERR_CLOCKOUTPUT_OPTION	-4607	Specified ClockOutput is not supported.

DATA CODE

ALERR_DATACODE_UNSUPPORTED	-4700	Data Coding is not supported.
ALERR_DATACODE	-4701	Specified Data Code option is not valid.

DATA OFFSET

ALERR_DATAOFFSET_UNSUPPORTED	-4800	Data Offset is not supported.
ALERR_DATAOFFSET	-4801	Specified Data Offset option is not valid.

DMA

ALERR_DMA_MODES_UNSUPPORTED	-4900	DMA Modes are not supported.
ALERR_DMA_MODE	-4901	Specified DMA Modes option is not valid.
ALERR_DMA_CHANS_UNSUPPORTED	-4902	DMA Channels are not supported.
ALERR_DMA_CHAN	-4903	Specified DMA Channel option is not valid.
ALERR_DMA_INUSE	-4904	Specified DMA Channel is already in use.
ALERR_SETDMASTATUS_FLAG	-4905	The status flag specified is unknown.

INTERRUPTS

ALERR_IRQ_LEVELS_UNSUPPORTED	-5000	IRQ Levels are not supported.
ALERR_IRQ_LEVEL	-5001	Specified IRQ Level option is not valid.
ALERR_IRQ_INUSE	-5002	Specified IRQ Level is already in use.
ALERR_SETIRQSTATUS_FLAG	-5003	The status flag specified is unknown.
ALERR_SETIRQSTATUS_OWNER	-5004	Invalid IRQ owner setting status flags.

SIGNAL PATHS

ALERR_SIGNALPATH_UNSUPPORTED	-5100	Signal Paths are not supported.
ALERR_SIGNALPATH	-5101	Specified Signal Path option is not valid.

DATA TRANSFER METHOD

ALERR_DTM_UNSUPPORTED	-5200	DTM Methods are not supported.
ALERR_DTM	-5201	Specified DTM Method option is not valid.

CYCLE MODE

ALERR_CM_UNSUPPORTED	-5300	CM Methods are not supported.
ALERR_CM	-5301	Specified CM option is not valid.
ALERR_CM_NUMBUFFERS	-5302	Selected CM option is not valid for the number of buffers specified.

DATA SPAN

ALERR_DATASPAN_UNSUPPORTED	-5400	Data Span is not supported.
ALERR_DATASPAN	-5401	Specified Data Span option is not valid.

DATA RANGE

ALERR_DATARANGE_UNSUPPORTED	-5500	Data Range is not supported.
ALERR_DATARANGE	-5501	Specified Data Range option is not valid.

INPUT CONFIG

ALERR_INPUTCONFIG_UNSUPPORTED	-5600	Input Config is not supported.
ALERR_INPUTCONFIG	-5601	Specified Input Config option is not valid.

OUTPUT CONFIG

ALERR_OUTPUTCONFIG_UNSUPPORTED	-5700	Signal Paths are not supported.
ALERR_OUTPUTCONFIG	-5701	Specified Signal Path option is not valid.

BUFFER NOTIFICATION METHOD

ALERR_SETBUFFDONEHANDLER_UNSUPPORTED	-5800	Buffer Notification Methods are not supported.
ALERR_SETBUFFDONEHANDLER_METHOD	-5801	Specified buffer Notification Method option is not valid.
ALERR_SETBUFFDONEHANDLERMSG_UNSUPPORTED	-5802	Buffer PostMessage Notification is not supported.
ALERR_SETBUFFDONEHANDLERFUNC_UNSUPPORTED	-5803	Buffer CallBack Notifications are not supported.
ALERR_ SETBUFFDONEHANDLERMSGPARAMS_UNSUPPORTED	-5804	Buffer PostMessageParams are not supported.

START DEVICE

ALERR_DEVICE_BUSY	-5900	The device is already running.
ALERR_DEVICE_UNINITIALIZED	-5901	The device is not initialized.

DATA BUFFER CONTROL ERRORS

ALERR_BUFFER_HANDLER	-6000	Invalid handler specified.
ALERR_BUFFER_SIZE	-6001	Invalid buffer size specified.
ALERR_BUFFER_NUMBEROF	-6002	Invalid number of buffers specified.
ALERR_BUFFER_TYPE	-6003	Invalid buffer type specified.
ALERR_BUFFER_NOTIFY_METHOD	-6004	Invalid buffer notification method specified.
ALERR_BUFFER_NOTSUPPORTED	-6005	Buffer(s) are not supported on this LHLD.
ALERR_BUFFER_POINTER	-6006	Buffer pointer equal to NULL.
ALERR_BUFFER_ERROR	-6007	An error condition occured in the previous buffer making the next DONE_BUFFER unavailable.
ALERR_DONEBUFFER_NOTREADY	-6008	The next buffer has not completed.
ALERR_BUFFER_STARTPOINT	-6009	The specified buffer StartPoint position number is out of range for the available buffer size.
ALERR_BUFFER_MAXCOUNT	-6010	The specified number of buffer counts exceeds the available buffer size.
ALERR_BUFFER_STATUS_PTR	-6011	Invalid LPDATABUFFSTATUS specified.
ALERR_BUFFER_NUMBER	-6012	Invalid buffer number specified.
ALERR_BUFFER_INTERNAL	-6013	Unable to locate specified buffer.

BURST MODE CONTROL ERRORS

ALERR_BURSTMODE_UNSUPPORTED	-7000	Burst mode is not supported on this LHLD.
ALERR_BURSTMODE_OPTION	-7001	Invalid burst mode specified.
ALERR_BURST_RATELOW	-7002	Burst mode rate too low for this LHLD.
ALERR_BURST_RATEHIGH	-7003	Burst mode rate too high for this LHLD.
ALERR_BURST_MINLENGTH	-7004	Burst mode length too low for this LHLD.
ALERR_BURST_MAXLENGTH	-7005	Burst mode length too high for this LHLD

GATING

ALERR_GATING_UNSUPPORTED	-8000	Gating is not supported.
ALERR_GATING_OPTION	-8001	Specified Gate option is not supported.
ALERR_SETSWGATE_UNSUPPORTED	-8002	SetSwGate function is not supported.
ALERR_GETSWGATE_UNSUPPORTED	-8003	GetSwGate function is not supported.

GATE LEVELS

ALERR_GATELEVELS_UNSUPPORTED	-9000	Gate levels are not supported.
ALERR_GATELEVEL_OPTION	-9001	Specified Gate level option is not supported.

DRIVERS

ALERR_DRV_NOT_LOADED	-10002	The board's device driver is not loaded.
ALERR_DRV_ADDR_STRUCT_PTR	-10004	Invalid Driver address struct pointer.
ALERR_DRV_STOPDEV_ADDR_PTR	-10005	Driver StopDevice function not found.
ALERR_DRV_DTM_ADDR_PTR	-10006	Driver DataTransMethod function not found.
ALERR_DRV_CM_ADDR_PTR	-10007	Driver CycleMode function not found.
ALERR_DRV_SETDMA_ADDR_PTR	-10008	Driver SetDma function not found.
ALERR_DRV_SETIRQ_ADDR_PTR	-10009	Driver SetIrq function not found.
ALERR_DRV_INIT_ADDR_PTR	-10010	Driver InitAddress function not found.
ALERR_DRV_SETCHANGAIN_ADDR_PTR	-10011	Driver SetChanGain function not found.
ALERR_DRV_SETCJ_ADDR_PTR	-10012	Driver SetCj function not found.
ALERR_DRV_SETBURSTMODE_ADDR_PTR	-10013	Driver SetBurstMode function not found.
ALERR_DRV_SETCLKMODE_ADDR_PTR	-10014	Driver SetClkMode function not found.
ALERR_DRV_SETCLKSRC_ADDR_PTR	-10015	Driver SetClkSrc function not found.
ALERR_DRV_SETCLKSRCSIG_ADDR_PTR	-10016	Driver SetClkSrcSig function not found.
ALERR_DRV_SETTRIGMODE_ADDR_PTR	-10017	Driver SetTrigMode function not found.
ALERR_DRV_SETTRIGSRC_ADDR_PTR	-10018	Driver SetTrigSrc function not found.
ALERR_DRV_SETTRIGSRCSIG_ADDR_PTR	-10019	Driver SetTrigSrcSig function not found.
ALERR_DRV_SETINPUT_ADDR_PTR	-10020	Driver SetInputConfig function not found.
ALERR_DRV_SETDATACODE_ADDR_PTR	-10021	Driver SetDataCode function not found.
ALERR_DRV_SETDATAOFFSET_ADDR_PTR	-10022	Driver SetDataOffset function not found.
ALERR_DRV_SETDATASPAN_ADDR_PTR	-10023	Driver SetDataSpan function not found.
ALERR_DRV_SETSIGNALPATH_ADDR_PTR	-10024	Driver SetSignalPath function not found.
ALERR_DRV_SETDATARANGE_ADDR_PTR	-10025	Driver SetDataRange function not found.
ALERR_DRV_SETOUTPUTCONFIG_ADDR_PTR	-10026	Driver SetOutputConfig function not found.
ALERR_DRV_SETBUFFERS_ADDR_PTR	-10027	Driver SetBuffers function not found.
ALERR_DRV_SETDAOUTPUT_ADDR_PTR	-10028	Driver SetDaOutput function not found.

DRIVERS - continued

ALERR_DRV_BUFFERNOTIFYMETHOD_ADDR_PTR	-10029	Driver SetBuffNotifyMethod function not found.
ALERR_DRV_SETPOSTSMPLCOUNTS_ADDR_PTR	-10030	Driver SetOutputConfig function not found.
ALERR_DRV_STARTDEV_ADDR_PTR	-10031	Driver StartDevice function not found.
ALERR_DRV_GETDEVSTATUS_ADDR_PTR	-10032	Driver GetDeviceStatus function not found.
ALERR_DRV_DIGINPUT_ADDR_PTR	-10033	Driver Digital Input function not found.
ALERR_DRV_DIGBITSTEST_ADDR_PTR	-10034	Driver Digital Bit Test function not found.
ALERR_DRV_DIGOUTPUT_ADDR_PTR	-10035	Driver Digital Output function not found.
ALERR_DRV_SETGATESRC_ADDR_PTR	-10036	Driver SetGateSrc function not found.
ALERR_DRV_SETGATESRCLEVEL_ADDR_PTR	-10037	Driver SetGateSrcLevel function not found.
ALERR_DRV_SETSWGATE_ADDR_PTR	-10038	Driver SetSWGate function not found.
ALERR_DRV_GETSWGATE_ADDR_PTR	-10039	Driver GetSWGate function not found.
ALERR_DRV_BOARDHWID_ADDR_PTR	-10040	Driver GetBoardID function not found.
ALERR_DRV_BOARDHWVER_ADDR_PTR	-10041	Driver GetBoardVer function not found.
ALERR_DRV_DRIVERVER_ADDR_PTR	-10042	Driver GetDrvVer function not found.
ALERR_DRV_BOARDERROR_ADDR_PTR	-10043	Driver GetBrdError function not found.
ALERR_DRV_BUFFERMESSAGEHANDLER_ADDR_PTR	-10044	Driver SetBuffmessagehandler function not found.
ALERR_DRV_BUFFERCALLBACKFUNC_ADDR_PTR	-10045	Driver SetBuffcallbackfunc function not found.
ALERR_DRV_DEMUXDATA_ADDR_PTR	-10046	Driver DemuxData function not found.
ALERR_DRV_DEMUXDATASET_ADDR_PTR	-10047	Driver DemuxDataSet function not found.
ALERR_DRV_SETFILTERTYPE_ADDR_PTR	-10048	Driver SetfilterType function not found.
ALERR_DRV_SETHANDSHAKE_ADDR_PTR	-10049	Driver SetHandShake function not found.

DRIVERS - continued

ALERR_DRV_SETPORTRESOLUTION_ADDR_PTR	-10050	Driver SetPortResolution function not found.
ALERR_DRV_SETERRONTRIGOVERRUN_ADDR_PTR	-10051	Driver SetErrOnTrigOverrun function not found.
ALERR_DRV_SETPORTCONTROL_ADDR_PTR	-10052	Driver SetPortControl function not found.
ALERR_DRV_BUFFERMESSAGEHANDLERPARAMS_ADDR_PTR	-10053	Driver SetBuffMessageHandlerParams function not found.
ALERR_DRV_GETACTUALCLOCKRATE_ADDR_PTR	-10054	Driver GetActualClkRate function not found.
ALERR_DRV_SETPACKEDDATA_ADDR_PTR	-10055	Driver SetPackedData function not found.
ALERR_DRV_SETCLKOUTPUT_ADDR_PTR	-10056	Driver SetClkOutput function not found.
ALERR_DRV_SETTRIGOUTPUT_ADDR_PTR	-10057	Driver SetTriggerOutput function not found.
ALERR_DRV_SETINPUTCONFIGLIST_ADDR_PTR	-10058	Driver InputConfigList function not found.
ALERR_DRV_SETDATAOFFSETLIST_ADDR_PTR	-10059	Driver DataOffsetList function not found.
ALERR_DRV_SETCTRMODE_ADDR_PTR	-10060	Driver CtrMode function not found.
ALERR_DRV_COUNTEROUT_ADDR_PTR	-10061	Driver CounterOut function not found.
ALERR_DRV_COUNTERIN_ADDR_PTR	-10062	Driver CounterIn function not found.

DEVICE DRIVER HANDLES

ALERR_LHDRVSUBSYS	-10100	The LHDRVSUBSYS specified does not exist.
ALERR_LHDRVSUBSYS_MAX	-10101	The maximum LHDRVSUBSYS have already been allocated.

RUNTIME ERRORS

ALERR_RELEASE_RUNNING	-10200	An attempt to release the LHLD was made while the LHLD was running.

FILTERS

ALERR_FILTERS_UNSUPPORTED	-10300	Filtering is not supported on this LHLD.
ALERR_FILTERS_OPTION	-10301	Invalid Filter Type specified.
ALERR_FILTERS_FREQLOW	-10302	Filter frequency too low for this LHLD.
ALERR_FILTERS_FREQHIGH	-10303	Filter frequency too high for this LHLD.

HANDSHAKE

ALERR_HANDSHAKING_UNSUPPORTED	-10400	Handshaking is not supported on this LHLD
ALERR_HANDSHAKE_OPTION	-10401	Invalid Handshake specified

PORT RESOLUTION

ALERR_PORTRESOLUTION_UNSUPPORTED	-10500	PortResolution is not supported on this LHLD.
ALERR_PORTRESOLUTION_OPTION	-10501	Invalid PortResolution specified.
ALERR_PORTMASK_MINMASK	-10502	Invalid PortMask minimum.
ALERR_PORTMASK_MAXMASK	-10503	Invalid PortMask maximum.
ALERR_PORTCONTROL_STRUCT_PTR	-10504	Invalid PortControl structure pointer.
ALERR_PORTCONTROL_STRUCT_UNSUPPORTED	-10505	PortControl structure is not supported.

CTR MODE

ALERR_CTRMODE_UNSUPPORTED	10600	Ctr Mode is not supported on this LHLD.
ALERR_CTRMODE_OPTION	10601	Invalid Ctr Mode specified.

CTR MODE (continued)

ALERR_COUNTER_RATELOW	-10700	Invalid Counter rate specified.
ALERR_COUNTER_RATEHIGH	-10701	Invalid Counter rate specified.

ENVIRONMENT SPECIFIC ERROR CODES

ALERR_LOADLIBRARY_ERROFFSET	-100000	ADLIB ERROR CODE OFFSET.
ALERR_LOADLIBRARY	-100001	System was out of memory, executable file was corrupt, or relocations were invalid.
ALERR_LOADLIBRARY_FILE	-100002	File was not found.
ALERR_LOADLIBRARY_PATH	-100003	Path was not found.
ALERR_LOADLIBRARY_LINK	-100005	Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
ALERR_LOADLIBRARY_DATASEG	-100006	Library required separate data segments for each task.
ALERR_LOADLIBRARY_MEM	-100008	There was insufficient memory to start the application.
ALERR_LOADLIBRARY_VER	-100010	Windows version was incorrect.
ALERR_LOADLIBRARY_INV	-100011	Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
ALERR_LOADLIBRARY_OPSYS	-100012	Application was designed for a different operating system.
ALERR_LOADLIBRARY_DOS40	-100013	Application was designed for MS-DOS 4.0.
ALERR_LOADLIBRARY_UNKNOWN	-100014	Type of executable file was unknown.
ALERR_LOADLIBRARY_REALMODE	-100015	Attempt was made to load a real-mode application (developed for an earlier version of Windows).

ENVIRONMENT SPECIFIC ERROR CODES - continued

ALERR_LOADLIBRARY_SECONDINST	-100016	Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read only.
ALERR_LOADLIBRARY_COMPRESSED	-100019	Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
ALERR_LOADLIBRARY_DLLMISSING	-100020	Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
ALERR_LOADLIBRARY_32BITREQ	-100021	Application requires Microsoft Windows 32-bit extensions.

-10XXXX
Note that 32 bit load library errors are a combination of GetlastError() and AL_LOADLIBRARY_ERROROFFSET. To determine the exact error that occurred subtract AL_LOADLIBRARY_ERROROFFSET from the Error Code returned and see the SDK Error Codes.

ADLIB INTERNAL ERROR CODES

ALERR_INTERNAL_ISARBITRARY_PARAMETER	-200000	Parameter validation failed.
ALERR_INTERNAL_ISNUMSEQLIST_PARAMETER	-200001	Parameter validation failed.

44. BOARD DRIVER ERROR CODES

DRVERR_DRVMAIN_PROCSTATE =	-401000	Invalid DriverMain procedure specified.
DRVERR_BOARD_NOT_PRESENT =	-401001	Unable to locate specified board.
DRVERR_INV_HBRDINST =	-401002	Invalid Board instance specified.
DRVERR_DEVICE_UNINITIALIZED =	-401003	The device has not been initialized.
DRVERR_MEMORY_LOW =	-401004	Unable to allocate memory.
DRVERR_MAXBOARDS_ALLOCATED =	-401005	The maximum boards have already been allocated.
DRVERR_BOARD_RESET_TIMEOUT =	-401006	Unable to reset the specified board.
DRVERR_PRODUCTID_UNMATCHED =	-401007	The product ID is incorrect for the specified board.
DRVERR_INV_BUFFER =	-401008	Invalid buffer in call to driver.
DRVERR_DEVICE_INUSE =	-401009	The specified device is busy.
DRVERR_SYSTEM_DRIVER_NOTFOUND =	-401010	System Driver Not Found.
DRVERR_SYSTEM_DRIVER_IOCTL =	-401011	An error occurred during an IOCTL Driver call.
DRVERR_BOARD_RESET_FAILED =	-401012	The driver was unable to properly reset the specified device.
DRVERR_THREAD_BUSY_TIMEOUT =	-401013	The internal driver is not responding to STOPAD command.
DRVERR_SYSTEM_DMATRANS_FAILED =	-401014	The System driver DMA Transfer initiation failed.
DRVERR_THREAD_READY_TIMEOUT =	-401015	An Internal Driver Thread has stopped responding to the system.
DRVERR_INVALID_DRVSUBSYS =	-401016	The specified driver subsystem does not exist.
DRVERR_MMTIMER_ALLOCATION=	-401017	The specified driver subsystem does not exist.
DRVERR_SYSTEM_DRIVER_CREATE_EVENT=	-401018	The system driver can not insert an event object.
DRVERR_SYSTEM_DRIVER_IRP_INSERT=	-401019	The system driver can not insert an IRP buffer

DRVERR_CYCLEMODE_TRANSMETHOD_CONFLICT =	-401100	A conflict between the CycleMode and transfermethod has been specified, creating an invalid setup.
DRVERR_BUFFNOTIFY_TRANS_METHOD_CONFLICT =	-401101	A conflict between the buffer notification method and Transfermethod has been specified, creating an invalid setup.
DRVERR_DMA_ARBCJ_CHANNELS_NOTSUPPORTED =	-401102	Aribitrary thermocouple channels are not supported with DMA.
DRVERR_CYCLEMODE_CLKSOURCE_CONFLICT =	-401103	A conflict between the CycleMode and ClockSource has been specified, creating an invalid setup.
DRVERR_CLKSOURCE_TRANSMETHOD_CONFLICT =	-401104	A conflict between the ClockSource and TransferMethod has been specified, creating an invalid setup.
DRVERR_ARBCJ_CHANNELS_NOTSUPPORTED =	-401105	Aribitrary thermocouple channels are not supported.
DRVERR_DMA_ARBCHANS_NOTSUPPORTED =	-401106	Aribitrary channels are not supported.
DRVERR_DMA_ARBGAINS_NOTSUPPORTED =	-401107	Aribitrary gains are not supported.
DRVERR_TRIGMODE_CLKSOURCE_CONFLICT =	-401108	A conflict between the TriggerMode and ClockSource has been specified, creating an invalid setup.
DRVERR_TRIGMODE_ARBCJ_CONFLICT =	-401109	A conflict between the TriggerMode and arbitrary CJ channels has been specified, creating an invalid setup.

DRVERR_TRIGMODE_ARBGAIN_CONFLICT =	-401110	A conflict between the TriggerMode and arbitrary Gains has been specified, creating an invalid setup.
DRVERR_TRIGMODE_NONSEQCHAN_CONFLICT =	-401111	A conflict between the TriggerMode and non-sequential channels has been specified, creating an invalid setup.
DRVERR_BURSTMODE_CLKSOURCE_CONFLICT=	-401112	A conflict between the BurstMode and ClockSource has been specified, creating an invalid setup.
DRVERR_TRIGSOURCE_ BURSTMODE_ CONFLICT=	-401113	A conflict between the TriggerSource and BurstMode has been specified, creating an invalid setup.
DRVERR_TRIGSOURCE_CLKSOURCE_ CONFLICT=	-401114	A conflict between the TriggerSource and ClkSource has been specified, creating an invalid setup.
DRVERR_TRIGSOURCE_TRIGMODE_ CONFLICT=	-401115	A conflict between the TriggerSource and TriggerMode has been specified, creating an invalid setup.
DRVERR_GATESOURCE_TRANSMETHOD_ CONFLICT=	-401116	A conflict between the GateSource and Transfer Method has been specified creating an invalid setup.
DRVERR_TRIGSOURCE_TRANSMETHOD_ CONFLICT=	-401117	A conflict between the TriggerSource and Transfer Method has been specified, creating an invalid setup.
DRVERR_C40PORT_ARBCJ_CHANNES_NOT SUPPORTED =	-401118	Arbitrary thermocouple channels are not supported over the C40 Port.

DRVERR_DAOUTPUT_TRANFERMETHOD_CONFLICT =	-401119	A conflict between the DAC Output and Transfer Method has been specified, creating an invalid setup.
DRVERR_CLKSOURCE_COUNTER0_CONFLICT =	-401120	A conflict between the Clock Source and Counter 0 has been specified, creating an invalid setup.
DRVERR_CLKSOURCE_TIMER1_CONFLICT =	-401121	A conflict between the Clock Source and Timer 1 has been specified, creating an invalid setup.
DRVERR_CLKSOURCE_TIMER0_CONFLICT =	-401122	A conflict between the Clock Source and Timer 0 has been specified, creating an invalid setup.
DRVERR_CLKSOURCE_COUNTER1_CONFLICT =	-401123	A conflict between the Clock Source and Counter 1 has been specified, creating an invalid setup.
DRVERR_ DRVERR_CLKRATEUNITS_INVALID =	-401124	An invalid ClockRateUnits setting was specified.

DRVERR_GAINCHANLIST_MINLEN =	-401200	The specified channel gain list is invalid.
DRVERR_GAINCHANLIST_MAXLEN =	-401201	The specified channel gain list is invalid.
DRVERR_GAINCHANLIST_ARRAY_PTR=	-401202	Invalid channel gain list.

DRVERR_PANELGAINLIST_MINLEN =	-401300	The specified panel gain list is invalid.
DRVERR_PANELGAINLIST_MAXLEN =	-401301	The specified panel gain list is invalid.
DRVERR_PANELGAINLIST_ARRAY_PTR =	-401302	Invalid panel gain list.

DRVERR_EXPPANEL_MINLEN =	-401400	The specified expansion panel gain list is invalid.

DRVERR_EXPPANEL_MAXLEN =	-401401	The specified expansion panel gain list is invalid.
DRVERR_EXPPANEL_ARRAY_PTR =	-401402	Invalid expansion channel gain list.
DRVERR_CJLIST_MINLEN =	-401500	The specified CJ list is invalid.
DRVERR_CJLIST_MAXLEN =	-401501	The specified CJ list is invalid.
DRVERR_CJLIST_ARRAY_PTR =	-401502	Invalid CJ list.
DRVERR_ADLOWRATE =	-401600	The specified A/D Clocking rate is low.
DRVERR_ADWITHTC_RATEHIGH =	-401601	The desired A/D clocking rate with thermocouple expanders is too high.
DRVERR_DAOUTRANGE =	-401700	Invalid D/A range specified.
DRVERR_DA_NOTAVAILABLE =	-401701	D/A support is not available.
DRVERR_DAUNITS_INVALID =	-401702	The specified D/A unit is not available.
DRVERR_IRQ_INUSE =	-401800	The specified Interrupt level is already in use.
DRVERR_INTERRUPTS_NOTSUPPORTED =	-401801	Interrupts are not supported.
DRVERR_DMA_INUSE =	-401900	The specified DMA level is already in use.
DRVERR_DMA_NOTSUPPORTED =	-401901	DMA is not supported.
DRVERR_DMA_TRANSFER=	-401902	DMA Transfer Error.
DRVERR_ADFIFO_OVERRUN =	-402000	The board's A/D FIFO overrun bit has set.
DRVERR_BURST =	-402001	The board's A/D Burst error bit has set.
DRVERR_CLOCK =	-402002	The board's A/D Clock error bit has set.
DRVERR_ADDATA_OVERRUN =	-402003	The board's A/D DATA overrun bit has set.
DRVERR_TRIGGER_OVERRUN =	-402004	The board's trigger overrun bit has set.

DRVERR_INTERRUPT_OVERRUN =	-402005	The board's interrupt rate is too fast for the driver to respond.
DRVERR_DADATA_UNDERRUN =	-402006	The board’s D/A data underrun bit has set.
DRVERR_DAFIFO_UNDERRUN =	-402007	The board’s D/A data underrun bit has set.
DRVERR_BUFFER_OVERRUN =	-403000	The Current buffer has overrun.
DRVERR_BUFFER_NEXTBUSY =	-403001	The next buffer in sequence is not available for use by the driver.
DRVERR_GETDEVSTATUS_PARAM =	-404000	Invalid status function argument parameter specified.
DRVERR_STOPDEV_NOTSUPPORTED =	-405000	The requested operation is not supported.
DRVERR_INITDEV_NOTSUPPORTED =	-405001	The requested operation is not supported.
DRVERR_STARTDEV_NOTSUPPORTED =	-405002	The requested operation is not supported.
DRVERR_SETCYCLEMODE_NOTSUPPORTED =	-405003	The requested operation is not supported.
DRVERR_SETDATATRANSMETHOD_NOTSUPPORTED =	-405004	The requested operation is not supported.
DRVERR_SETGCLIST_NOTSUPPORTED =	-405005	The requested operation is not supported.
DRVERR_SETPANELLIST_NOTSUPPORTED =	-405006	The requested operation is not supported.
DRVERR_SETCJLIST_NOTSUPPORTED =	-405007	The requested operation is not supported.
DRVERR_SETCLOCKSOURCE_NOTSUPPORTED =	-405008	The requested operation is not supported.
DRVERR_SETTRIGGERMODE_NOTSUPPORTED =	-405009	The requested operation is not supported.
DRVERR_SETTRIGGERSOURCE_NOTSUPPORTED =	-405010	The requested operation is not supported.
DRVERR_SETTRIGGERSOURCESIG_NOTSUPPORTED =	-405011	The requested operation is not supported.

DRVERR_SETGATESOURCE_NOTSUPPORTED =	-405012	The requested operation is not supported.
DRVERR_SETSWGATE_NOTSUPPORTED =	-405013	The requested operation is not supported.
DRVERR_GETSWGATE_NOTSUPPORTED =	-405014	The requested operation is not supported.
DRVERR_GATEANDTRIG_NOTSUPPORTED =	-405015	The requested operation is not supported.
DRVERR_SETGATESOURCELEVEL_NOTSUPPORTED =	-405016	The requested operation is not supported.
DRVERR_SETDATACODE_NOTSUPPORTED =	-405017	The requested operation is not supported.
DRVERR_SETDATAOFFSET_NOTSUPPORTED =	-405018	The requested operation is not supported.
DRVERR_SETBURSTMODE_NOTSUPPORTED =	-405019	The requested operation is not supported.
DRVERR_SETINPUTCONFIG_NOTSUPPORTED =	-405020	The requested operation is not supported.
DRVERR_SETDATARANGE_NOTSUPPORTED =	-405021	The requested operation is not supported.
DRVERR_SETDAOUTPUT_NOTSUPPORTED =	-405022	The requested operation is not supported.
DRVERR_SETBUFFERS_NOTSUPPORTED =	-405023	The requested operation is not supported.
DRVERR_SETBUFFNOTIFYMETHOD_NOTSUPPORTED =	-405024	The requested operation is not supported.
DRVERR_CALLBACKHANDLER_INVALID =	-405025	The Specified call-back handler is invalid.
DRVERR_SETPOSTSAMPLECOUNTS_NOTSUPPORTED =	-405026	The requested operation is not supported.
DRVERR_SETBUFFMSGHANDLER_NOTSUPPORTED =	-405027	The requested operation is not supported.
DRVERR_GETDEVSTATUS_NOTSUPPORTED =	-405028	The requested operation is not supported.
DRVERR_DIGITALINPUT_NOTSUPPORTED =	-405029	The requested operation is not supported.

DRVERR_DIGITALOUTPUT_NOTSUPPORTED =	-405030	The requested operation is not supported.
DRVERR_BOARD_READYBIT_TIMEOUT =	-405031	The boards ready bit flag indicator is not setting.
DRVERR_SETFILTERTYPE_NOTSUPPORTED =	-405032	Filter types are not supported.
DRVERR_SETERRONTRIGOVERRUN_NOTSUPPORTED =	-405033	ErrOnTrigOverrun is not supported.
DRVERR_SETBUFFMSGHANDLERPARAMS_NOTSUPPORTED -	-405034	User PostMessage parameters are not supported
DRVERR_GETACTUALCLOCKRATE_NOTSUPPORTED =	-405035	The requested operation is not supported or the clock source is unknown such as ext.clocking.
DRVERR_SETDATAOFFSETLIST_NOTSUPPORTED =	-405036	The requested operation is not supported.
DRVERR_SETTRIGGEROUTPUT_NOTSUPPORTED =	-405037	The requested operation is not supported.
DRVERR_SETCLOCKOUTPUT_NOTSUPPORTED =	-405038	The requested operation is not supported.
DRVERR_SETCTRMODE_NOTSUPPORTED =	-405039	The requested operation is not supported.
DRVERR_COUNTERIN_NOTSUPPORTED =	-405040	The requested operation is not supported.
DRVERR_COUNTEROUT_NOTSUPPORTED =	-405041	The requested operation is not supported.
DRVERR_SETINPUTCONFIGLIST_NOTSUPPORTED =	-405042	The requested operation is not supported.
DRVERR_GETBOARDERROR_NOTSUPPORTED =	-405043	The requested operation is not supported.
DRVERR_SETPACKEDDATA_NOTSUPPORTED =	-405044	The requested operation is not supported.
DRVERR_BOARD_FNEBIT_TIMEOUT =	-405045	The boards FNE bit is not setting after a conversion has been initiated.

DRVERR_BURSTRATE_LOW=	-406000	The specified Burst rate is low.
DRVERR_BURSTRATE_HIGH=	-406001	The specified Burst rate is high.
DRVERR_SCANRATE_LOW=	-406100	The specified Scan rate is low.
DRVERR_SCANRATE_HIGH=	-406101	The specified Scan rate is high.