User manual for

UEIDAQ

Software for WIN-10x/30x Data Acquisition Board

1197

High Performance Data Acquisition Software for IBM PC, PC/XT, PC/AT and compatible Computer Systems.

Tenth Edition.

November 1997 Printing

Table of Contents

INDEX

Preface

This manual is written for users of UEIDAQ software package. It provides all information necessary to successfully use the supplied driver software in conjunction with several languages.

This manual assumes:

- That you have a basic knowledge of electronic circuitry and measurement techniques.
- That you are familiar with the host PC which you are using.
- That you are capable of writing your own programs.
- That you have read chapters 1 through 4 of the user manual accompanying your WIN-10/30

The manual contains the following sections.

Chapter 1 - Introduction.

- Chapter 1 contains an overview of UEIDAQ software package, and its capabilities.

Chapter 2 - Quick Reference.

- Chapter 2 provides a quick reference to the various driver functions.

Chapter 3 - Data structures.

- Chapter 3 discusses the various data structures, global variable and constants used by the driver package.

Chapter 4 - DOS Programmers Guide

- Chapter 4 discusses various aspects of the driver software, including memory models, segment naming, files and modules for DOS applications.

Chapter 5 - Windows Programmers Guide

- Chapter 5 provides a discussion of various issues involved with programming under the Windows environment. Prior to reading this section, you should have read the chapter on DOS operations.

Chapter 6 - Function Reference.

- Chapter 6 provides a detailed description of each driver function.

Appendix

- A: Reading Wdemo1 Packed Data Files
- B: Reading Status for Windows Data Files

Chapter 1

1. Introduction

This chapter introduces UEIDAQ system

1.1. Introduction

UEIDAQ software has been written to allow the easy use of all WIN-30 functions from various high level languages. Also included with the driver are :

- Source code (in C) for the entire driver system.
- Demonstration software in a variety of languages.

UEIDAQ comes in the following flavors:

i. UEIDAQ for DOS. This provides DOS support via linkable object modules

ii. UEIDAQ for Windows 3.1. This provides 16-bit support via a DLL and virtual device driver(Vxd)

iii. UEIDAQ for Windows 95: This provides 32-bit support via a DLL and virtual device driver(Vxd)

iv. UEIDAQ for Windows NT: This provides 32-bit support via a DLL and kernel mode driver. (KMD)

This manual covers the operation of all the above.

Note: The latest versions of software, including source code is available via the internet at www.ueidaq.com – select Software Updates and download the appropriate disks.

1.2. Boards supported

UEIDAQ package supports all WIN-10/30 and UEI-30 series boards in the range.

1.2.3. 32-bit series boards.

The 32-bit series of boards consists of eight boards. All 32-bit series boards are supported in PC/AT systems, and are fully compatible with the 16-bit series of boards. The first part of the "32-bit" series of boards is the WIN-30 series, which features input resolution of 12 bits:

	Analog Inputs	**Analog Outputs**
WIN-10/30D	16 single ended	-
WIN-10/30DA	16 single ended	Two 12-bit, two 16-bit
WIN-10/30DS	16 simultaneously sampled	Two 12-bit, two 16-bit
WIN-10/30DS/4	4 simultaneously sampled	Two 12-bit, two 16-bit
WIN-10/30PGL	8 differential, programmable gains of 1, 10, 100, 1000	Two 12-bit, two 16-bit
WIN-10/30PGH	8 differential, programmable gains of 1, 10, 100, 1000	Two 12-bit, two 16-bit
WIN-10/30PGSL	8 differential simultaneously sampled, programmable gains of 1, 2, 4, 8	Two 12-bit, two 16-bit
WIN-10/30PGSH	8 differential simultaneously sampled, programmable gains of 1, 10,100, 1000	Two 12-bit, two 16-bit

Note: The WIN-10 boards only have two 16-bit D/A’s.

The WIN-30 series of boards is also available in 16 bit analog input resolution, as the WIN-3016 series, detailed below:

	Analog Inputs	**Analog Outputs**
WIN-1016/3016D	16 single ended	-
WIN-1016/3016DA	16 single ended	Two 12-bit, two 16-bit
WIN-1016/3016DS	16 simultaneously sampled	Two 12-bit, two 16-bit
WIN-1016/3016DS/4	4 simultaneously sampled	Two 12-bit, two 16-bit
WIN-3016PGL	8 differential, programmable gains of 1, 10, 100, 1000	Two 12-bit, two 16-bit
WIN-3016PGH	8 differential, programmable gains of 1, 2, 4, 8	Two 12-bit, two 16-bit
WIN-3016PGSL	8 differential simultaneously sampled, programmable gains of 1, 2, 4, 8,	Two 12-bit, two 16-bit
WIN-3016PGSH	8 differential simultaneously sampled, programmable gains of 1, 10,100, 1000,	Two 12-bit, two 16-bit

Note: The WIN-1016 boards only have two 16-bit D/A’s.

1.3. Windows support

Windows support is provided as follows:

- Windows 3.1/95 support is via a DLL (Dynamic Linked Library) and a virtual device driver (Vxd). Both of these must be accessible to Windows for the driver system to operate. Installation of the DLL and Vxd are discussed further in section 5.1 of this manual.

- Windows NT 3.51 and greater is fully supported. Please refer to section 5.1 of this manual.

- A minimum configuration of a 386 processor and 4 Mbytes of memory is required. A 386DX, 486 or Pentium processor and 8 Mbytes of memory are recommended.

1.4. Languages supported

The following languages are supported :

- Visual C++ (Version 1.52 (16-bit) and 4.0 (32-bit) or later)

- Microsoft QuickBasic (V4.5 or later)

- Microsoft QuickC (V2 or later)

- Microsoft FORTRAN (V5 or later)

- Borland C++ (V4.00 or later).

- Borland Pascal for DOS (version 7.0 or later). The Borland Pascal driver does not support interrupts.

- Borland Pascal for Windows(version 7.0 or later).

- Visual Basic (version 3.0 or later)

- QNX operating system (optional)

- Almost all other compiled languages can also be used in conjunction with one or the other of the object modules.

1.5. Functions supported

Functions supported vary, depending on the board in question:

1.5.1. WIN-10/30D and WIN-1016/3016D

: Diagnostics, obtain a single A/D sample, obtain a series of A/D samples on either a single or multiple channels, either single/dual channel DMA or rep string operation, gap free dual DMA channel operation, extended memory access, interrupt operations, digital I/O.

1.5.2. WIN-10/30DA and WIN-1016/3-16DA

: Diagnostics, obtain a single A/D sample, obtain a series of A/D samples on either a single or multiple channels, either single/dual channel DMA or rep string operation, gap free dual DMA channel operation, extended memory access, interrupt operations, analog output on all four D/A outputs, waveform generation, digital I/O.

1.5.3. WIN-10/30DS and WIN-1016/3016DS

The WIN-30DS and WIN-30DS/4 share all of the abilities of the WIN-30DA, and appear identical to software. The only difference in operation comes about in block mode operation, where all sampled channels are sampled simultaneously. All references in this manual to WIN-30D boards also applies to WIN-30DS and WIN-30DS/4 boards, unless specifically stated otherwise.

1.5.4. WIN-10/30PG and WIN-1016/3016PG

The shares all the functions of the WIN-30DA, but in addition allows the gain of each analog input channel to be individually set.

1.5.5. WIN-10/30PGS and WIN-1016/3016PGS

The shares all the functions of the WIN-10/30PG, but in addition allows simultaneous sample of each analog input channel.

Note: The name WIN-30 will be used for referencing the WIN-10 boards.

Chapter 2

2. Quick Reference

This section provides a quick reference to the various driver functions. Detailed descriptions are provided later.

Function	Description
Daq_version	Returns the driver version number.
Vxd_version	Returns the version number of the Windows 3.1 VXD; also serves as a check for the presence of the VXD
Select_board	Selects the logical board number (used for multi-board applications).
Set_base	Sets the board base address.
Get_base	Retrieves the board base address.
Daq_close	Closes the driver.
Load_dsp	Down loads code to the DSP processor.
Init	Initializes the board.
Get_type	Gets the board type.
Set_type	Sets the board type.
Diag	Diagnostics function and board identification.
Config_bd	Configures interrupt and DMA levels, analog input ranges, clock settings etc.
Ad_chan_cfg -	Configures AD channels (more advanced than Config_bd)
Rtc_off	Switches the PC real-time clock off.
Rtc_on	Switches the PC real-time clock on.
Clean	Readies the board for an A/D conversion.

Ch_list_load	Loads the hardware channel list and block counter.
Set_gain	Sets the gain of a single input channel.
Get_gain	Retrieves the gain of a selected input channel.
Load_gain	Loads the UEI-30 gain memory.
D_mode	Configures the digital I/O ports.
D_in	Digital input.
D_out	Digital output.
Set_osc	Sets the master oscillator frequency.
Get_osc	Reads the master oscillator frequency
Ad_clock	Sets the A/D clock divider.
Da_clock	Sets the D/A clock divider.
Ad_prescaler	Sets the A/D clock prescaler.
Ad_in	Obtains a single A/D sample.
Ad_in_1, Ad_in_2	Obtains a single A/D value for systems with UEI-81 channel expansion boards.
Da_out	Outputs a value to a D/A converter.
S_chan	Obtains a series of samples from a single channel.
M_chan	Obtains a series of samples from a sequence of channels.
Mb_chan	Obtains a series of samples from a sequence of channels in block trigger mode.
Mbh_chan	Obtains a series of samples from multiple channels using Smart Cache (rep-string) operations. Greater than 32K samples and 1 MHz throughput supported.
Mibh_chan	Obtains a series of samples from multiple channels using Smart Cache (rep-string) operations, using interrupts to operate in background mode. Greater than 32K samples and 1 MHz throughput supported.
Mdb_chan	Obtains a series of samples from a sequence of channels in block trigger mode, under DMA control.
Mdh_chan	Uses gap-free DMA to obtain more than 32K samples.
Mde_chan	Uses gap-free DMA to obtain A/D samples into extended memory.
Mdih_chan	Uses gap-free DMA to obtain more than 32K samples, channel swapping handled in background via interrupts.
Mdih_cont	Continuos gap-free DMA into a huge array, with background processing of

	DMA channel swaps.
Ad_cont_close	Close continuos acquisition system
Xfer_dma_res	Transfers samples from extended memory to a user buffer.
E_mem_size	Checks how many samples can be obtained into extended memory.
Mi_chan	Obtains a series of samples from a sequence of channels using interrupts. This is a background task.
Int_chk	Checks for completion of any interrupt operation.
Int_close	Shuts down the interrupt system.
Sd_chan	Obtains a block of samples from a single channel under DMA. This is a background task.
Dma_init	Initializes the DMA system.
Dma_chk	Checks for completion of DMA operations, as well as reprogramming the DMA system for gap-free DMA.
Dma_close	Shuts down the DMA system after Sd_chan.
Cntr_cfg	Configures the uncommitted counter/timer.
Cntr_write	Writes a count value to the uncommitted counter/timer.
Cntr_read	Reads a count value from the uncommitted counter/timer.
Wfm_chan	Generates a waveform from board D/A outputs.
Wfmih_chan	Generates a waveform from board D/A outputs, using interrupts.

Chapter 3

3. Data Structures

This chapter provides a description of the various data structures and constants used by the driver.

3.1. Board Type

Board types are identified by a single integer value, which is used by the driver to store the type of board. This is internally stored by the driver. The board's type is normally auto detected by the Diag driver function, but can be directly set by the Set_type function. Board type can be obtained by the Get_type function, which can return the following values:

i. found_w30d (7). This indicates that a WIN-10/30D,DA,DS4 or DS was detected.

ii. found_w30pgl (8). This indicates that a WIN-10/30PGL or WIN-10/30PGSL was detected.

iii. found_w30pgh (9). This indicates that a WIN-10/30PGH or WIN-10/30PGSH was detected.

iv. found_w3016d (10). This indicates that a WIN-1016/3016D,DA,DS4 or DS was detected.

Warning
The function Diag must be called prior to using any other driver function, in order to initialize the board type information, or the Set_type function must be used to do this. Failure to initialize the board type will result in unpredictable results.

3.2. Gain Values

Gain values are used by the driver system to set channel gain for boards which support channel gains. Codes corresponding to various gains are given below:

PGxH Boards	PGxL Boards	Gain code
1	1	0
2	10	1
4	100	2
8	1000	3

3.3. Return codes

Six return codes are used by UEIDAQ. These are all signed integers, and are defined in the include files and TURBO Pascal unit files.

3.3.1. ok_30 (0).

A return code of 0 indicates that the operation was successfully performed.

3.3.2. par_30 (-1).

A return code of -1 indicates that the function was not performed as one or more of the function parameters were out of range, invalid or inconsistent.

3.3.3. err_30 (-2).

A return code of -2 indicates that a error was encountered while performing an operation. This is normally due to a data overrun, typically as a result of too high a clock rate. This return code is never generated by UEI-26, UEI-30 or UEI-39 boards, as they do not have error detection.

3.3.4. mem_30 (-3).

This error code is generated by the extended memory functions, either as there is too little memory to perform the requested operation, or because a memory error was encountered while transferring data.

3.3.5. task_30 (-4).

This error code can be generated by any of the driver functions when running under Windows 3.1. UEIDAQ is limited to a maximum of eight physical boards, which may be accessed by a maximum of sixteen separate tasks. The task_30 error code indicates that more than 16 tasks attempted to access the boards under UEIDAQ's control. Note that each time you run an application, a new task is created. Thus you can run only 16 instances of the same application, if that application uses UEIDAQ.

3.3.6. n_comp_30 (1).

A return code of 1 indicates that an interrupt or DMA operation has not yet completed.

3.4. Data Format

When using the driver, analog data from the A/D and to the D/A converters is always in the form of right justified offset binary code. (The most negative voltage is represented as 0 Hex, the most positive as FFF hex). The driver automatically converts data formats for boards which make use of other formats (such as the UEI-126). The four most significant bits of any A/D data are always 0. The most significant four bits of D/A data are ignored.

Note that data to the 8-bit D/A converters is also treated as 12 bit data. The lower 4 bits are simply ignored.

If a series of samples is obtained, the data is laid out in the order that it was sampled. This is shown in figure 3.1, assuming that channels 3, 7 and 1 were sampled in that order. The driver automatically handles packing and unpacking of data, for boards which support this feature.

	Bit 7 (MSB)				Bit 0 (LSB)
Byte 0	**Sample 0 Bits 7-0**				
Byte 1	**0**	**0**	**0**	**0**	**Sample 0 Bits 11-8**
Byte 2	**Sample 1 Bits 7-0**				
Byte 3	**0**	**0**	**0**	**0**	**Sample 1 Bits 11-8**
Byte 4	**Sample 2 Bits 7-0**				
Byte 5	**0**	**0**	**0**	**0**	**Sample 2 Bits 11-8**
	Rest of samples				

Figure 3.1. Data layout.

3.4.1. Channel List

Several of the A/D input functions use a channel list array. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Once the last channel is sampled, the first channel is sampled again, until a total of n_samp conversion have been performed.

Generally, if the channel list contains less than 32 channels, the hardware channel list (if present on the board in question) is used. If not, and in the case of old series boards, then conventional techniques are used. Throughput for channel list mode operation can be considerably higher.

Example : in order to sample channels 1, 5 and 6 the channel list as follows :

location 0 - 1

location 1 - 5

location 2 - 6

location 3 - 16

Note that for the UEI-127, there are certain restrictions on the channel list:

- No channel may appear more than once.
- Channels must appear in ascending order.

For example, a list containing <1, 3, 4> would be valid, but <1, 2, 1> or <1, 4, 2> would not.

Chapter 4

4. DOS Programmers Guide

This chapter provides a guide for programmers using the driver system to write DOS programs.

4.1. Library Contents

The driver code is supplied in the form of library modules (.LIB files), 2 Quick libraries (.QLB files), 5 object modules (.OBJ files), a Borland Pascal unit (.TPU file) and several include files for the various supported languages. For any given program, you will only require a few of these, depending on the functions you require, and the memory model in use.

All of the driver code is derived from two basic driver modules. These are the following :

i. PC30S. This module is the basic module of the set, and includes all the directly callable driver functions. It may be compiled from PC30S.C using either Borland C version 3.1 or Microsoft C version 7.0 or later. Most other C compilers should also be able to compile this module.

ii. PC30A. This module provides DOS access to the interrupt and DMA modules, and some high speed board access code. In fact, all functions in this module can easily be performed by C library functions, but are implemented in assembler to avoid dependence on the presence of a C library. Functions in this module are intended only for internal use by driver. This module can be assembled from PC30.ASM by the Microsoft assembler, version 5.00 or later. If the symbol "lg" is defined, on object file suitable for use using large, medium or huge memory models is created. If "lg" is not defined, the an object module suitable for use with tiny, compact and small memory models is created.

4.2. Source code

Source code for the entire driver system can be found in the \SOURCE directory of the distribution disk. Also in this directory is a complete make file for all the library files. This make file assumes that you have Microsoft C V7 and MASM, but most other C compilers and assemblers should work. Borland C++ V3.1 is also required for the compilation of the modules used by Borland Pascal. NOTE: Although we encourage you to modify the supplied source code to suit your purposes, we cannot provide technical support for such modifications.

4.3. Library Modules

The most general form of the driver is to be found in the library modules (.LIB files). These are standard libraries, formatted for both Microsoft and Borland products, and are suitable for use with any language that support standard libraries. They are supplied in a wide variety of calling conventions and memory models, as described below.

Various other languages that cannot use these standard libraries are also supported via files suited to each specific language, for example, a TPU (Turbo Pascal Unit) file for Borland Pascal. These are described below, individually for each language.

4.3.1. Library module naming.

There are 10 library modules, all of which are different versions of a single basic module. This basic module is described below. The basic library name is as follows :

- 30M. This is the driver library module, and consists of two object modules. These are described below.

This module comes in 10 different forms, to suit different languages and memory models.

The actual module names are formed by combining the base name given above with a prefix and a postfix. The prefix gives the language for which the module is written, and the postfix the memory model.

4.3.1.1. Module Prefix.

The following prefixes are used.

i. P : This indicates that the module is intended for Microsoft languages which support the Pascal calling convention. Note : These modules cannot be used in conjunction with Borland Pascal. Special modules are provided for this purpose. See below.

ii. C : This indicates that the module is intended for Microsoft languages which support the C the naming convention.

iii. B : This indicates that the module is intended for Borland languages which support the C the naming convention.

4.3.1.2. Module Postfix.

i. _S : Small memory model.

ii. _C : Compact memory model.

iii. _M : Medium memory model.

iv. _L : Large memory model.

v. _H : Huge memory model.

Example : The main module for a Microsoft Pascal program which uses the small model would be P30M_S.LIB.

4.4. Object Modules

Each library file discussed above consists of one or more object files. These can be extracted for use on their own, if required. This is done by the LIB program supplied with your compiler or operating system.

4.4.1. Object module naming.

For the C files, the actual module names are formed by combining the base name (PC30S) with a prefix and a postfix. The prefix gives the language for which the module is written, and the postfix the memory model. These prefixes and postfixes are the same as described for the library modules above. For the assembler module only two modules are used. These are PC30AS and PC30AL. These support memory models which use near and far calls respectively.

Example : The standard module for a Pascal program which uses the small model would be P30S_S.OBJ.

4.4.2. C Calling convention.

The C language modules make use of Pascal (or PLM) calling conventions, and are declared as such in PC30.H. This is done to ease operation under Windows.

4.4.3. Segment names.

As found in the supplied library files, the segment naming is as follows:

4.4.3.1. Code segment

In the small and compact models, the code segment is _TEXT. In the medium, large and huge models, it is PC30_TEXT.

4.4.3.2. Data segment

In the small and medium model, the data segment is _DATA. In the compact, large and huge model, it is PC30_DATA.

4.5. Microsoft C

All supplied modules are directly compatible with Microsoft C version 7.0, and Visual C++ V1.00. You can either link your program to any of the C30m_x.LIB library files supplied (depending on memory model), or simply compile the C source code files. For example, to compile and link DEMO4.C in the large memory model use :

```
cl /AL demo4.c c30m_l.lib
```

4.5.1. Required files

PC30.H, C30M_x.LIB (depending on memory model)

4.5.2. Examples

DEMO4.C, DEMO5.C, DEMO6.C, DEMO7.C, DEMO8.C, DEMO9.C, DEMO10.C

4.6. Microsoft QuickC

Microsoft QuickC version 2 is fully supported for both the stand-alone and integrated environment operation. In order to operate within the integrated environment, you must load the UEI-30 QuickLibrary, C30M_M.QLB, at startup. This is done as follows :

```
qc /lc30m_m.qlb
```

Your C program must also include the PC30.H file.

If you are using QuickC in stand-alone mode (via the QCL command), you must link your program with the supplied C30M_M.LIB library. If you chose to generate a .EXE file from within QuickC, this is done automatically by QuickC. Note this applies to medium model programs only.

If you are using a version of QuickC other than 2, you may have to regenerate the C30M_M.QLB QuickLibrary. The MA30 make file contains the commands required to do this.

4.6.1. Required files

PC30.H, C30M_M.QLB and C30M_M.LIB

4.6.2. Examples

DEMO4.C, DEMO5.C, DEMO6.C, DEMO7.C, DEMO8.C, DEMO9.C

4.7. Borland C++

All supplied source code modules are directly compatible with Turbo C version 1.5 and later, and Borland C++ version 3.1 and later. In addition, library files (.LIB files) are supplied for Borland C++ V3.1 and Turbo C++. To compile correctly under either Borland product, you must define the "tcc" symbol. The MBCC30 file contains the commands used to generate libraries for all supported Borland products.

You can either link your program to any of the B30m_x.LIB library files supplied (depending on memory model), or simply compile the C source code files. For example, to compile and link DEMO4.C in the large memory model use :

```
bcc -ml -Dtcc demo4.c b30m_l.lib
```

Under the integrated environment, all that is required is to define a BC++ project containing all user written C (or C++) source files, the appropriate LIB file, depending on the memory model selected, and to define the "tcc" symbol. The DEMO4.PRJ file gives an example of how to do this for the DEMO4 sample program.

NOTE : Remember that you must define the "tcc" symbol for use with any Borland C compiler.

4.7.1. Required files

PC30.H, B30M_x.LIB (depending on memory model).

4.7.2. Examples

DEMO4.C, DEMO5.C, DEMO6.C, DEMO7.C, DEMO8.C, DEMO9.C

4.8. Borland Pascal

Borland Pascal requires special naming conventions which are not compatible with standard languages. For this reason special versions of object module 30S is generated by compiling with Borland C++. The object modules used is T30S_L.OBJ. The MBCC30 file contains the commands used to generate libraries for all supported Borland products. Note that in order to compile for Borland Pascal, the "tpc" symbol must be defined.

Similarly, a special version of the PC30A assembler module, called TPC30A, must be used. This is generated by assembling PC30.ASM with the "tpascal" symbol defined.

Note : The interrupt functions cannot be used in conjunction with Borland Pascal.

The two special modules are combined into a single Borland Pascal unit, PC30.TPU.

This module, which is compiled from PC30.PAS, contains both driver modules. In order to use the Borland Pascal Module, all that is required is to declare the PC30 unit under the "USES" clause of the Borland Pascal program.

4.8.1. Required files

PC30.PAS, PC30.TPU

4.8.2. Examples

DEMO1.PAS, DEMO2.PAS, DEMO3.PAS

4.9. Microsoft QuickBasic

Microsoft QuickBasic version 4.5 is fully supported for both the stand-alone and integrated environment operation. In order to operate within the integrated environment, you must load the UEI-30 QuickLibrary, Q30M_L.QLB, at startup. This is done as follows :

```
qb /l q30m_l.qlb
```

Your BASIC program must also include the PC30.INC file.

If you are using QuickBasic is stand-alone mode (via the BC command), you must link your program with the supplied Q30M_L.LIB library. If you chose to generate a .EXE file from within QuickBasic, this is done automatically by QuickBasic.

If you are using a version of QuickBasic other than 4.5, you may have to regenerate the Q30M_L.QLB QuickLibrary. The MA30 make file contains the commands required to do this.

4.9.1. Required files

PC30.INC, Q30M_L.QLB and Q30M_L.LIB

4.9.2. Examples

DEMO11.BAS and DEMO12.BAS

4.10. Microsoft FORTRAN

Microsoft FORTRAN version 5.1 is fully supported. In order to use FORTRAN, you must include the two FORTRAN include files, PC30.FI and PC30.FD into your source code. The PC30.FI file contains function and subroutine definitions, and need be included only at the start of the program. PC30.FD contains variable declarations, and must included into each section of the program that uses driver functions.

You must link your programs to the C30M_L.LIB library file. For example, to compile and link DEMO9.FOR :

```
fl demo13.for c30m_l.lib
```

4.10.1. Required files

PC30.FI, PC30.FD, C30M_L.LIB

4.10.2. Examples

DEMO13.FOR, DEMO14.FOR

4.11. C Include files

Two files which are designed to be included with the user's source code are included on the distribution disk.

4.11.1. PC30.H

This file is designed for inclusion with C programs which make use of any of the driver modules. It contains prototype declarations for all C functions used.

4.11.2. REG_30.H

This file is designed for inclusion with C programs in which the user accesses supported boards at the register level. It contains definition for the various supported board's registers.

4.12. Memory models

In order to make use of the linkable object modules supplied on the UEIDAQ disk, it is necessary to understand certain calling conventions used on 8086 series processors. There are two aspects to this:

i. Memory model. Memory model refers to the length of the pointers used to access data and code. These may be either short (16 bit), long (32 bit) or huge (32 bit). Huge pointers differ from long pointers in that huge pointers can be used on data items of greater than 64K. Long pointers can only address up to 64K. Six memory models are in common use:

   a) Small model. Short data and code pointers. This means that data pointers consist of a single 16 bit word, and calls are of NEAR type. Program and data are in different segments, allowing 64K of data and 64K of program.

   b) Medium model. Short data pointers and long code pointers. Hence calls are of FAR type. Up to 64K of data and 1M of program space is allowed.

   c) Compact model. Long data pointers and short code pointers. Hence calls are of near type. Up to 1M of data and 64K of program space is allowed. Individual data items may only be up to 64K in size.

   d) Large model. Long data and code pointers. Data pointers hence consist of 32 bit pointers and calls are of FAR type. Up to 1M of data and 1M of program space is allowed. Individual data items may only be up to 64K in size.

   e) Huge model. Long data and code pointers. Data pointers hence consist of 32 bit pointers and calls are of FAR type. Up to 1M of data and 1M of program space is allowed. The huge model allows individual data item to be greater than 64K in size.

ii. Stack convention. Almost all software makes use of the same convention for placing data on the stack. This is to push the first parameter first, the second etc. There is however a difference as to when these parameters are removed from the stack (i.e. the stack pointer readjusted).

   a) PLM calling convention. In this case it is the responsibility of the called procedure to remove any parameters (normally by a RET N). This calling convention is used by Pascal and FORTRAN.

   b) C calling convention. In this case the calling program must remove any parameters.

4.13. Multi-board operations

Versions of the driver software from V1.05 support multiple boards. This support is accomplished as follows :

i. Up to eight boards are supported, numbered from 0 to 7. These numbers are purely logical numbers.

ii. Boards are selected by the Select_board function.

iii. Initially, before the Select_board function has been called, board 0 is selected.

iv. Once a board has been selected, then all subsequent function calls apply to that and ONLY that board.

v. The driver internally stores the following information :

   a) Each board's base address.

   b) Each board's type.

   c) Each board's A/D gain settings.

d) When DMA or interrupt operations are in progress, complete state information for each of these operations is also stored.

vi. This means that once each board is initially configured, it is not necessary to reconfigure it after each Select_board call.

vii. For an example of multi-board operations, see Demo15.C

Note that it is the user's responsibility to ensure that no two boards ever try to use the same DMA or interrupt channels. Failure to ensure this can result in damage to the boards in question.

4.14. Differences from previous driver versions

Previous versions of the DOS drivers can be split into two groups, those prior to version 1.05, and those after.

4.15. Versions since 1.05

The present version of the DOS drivers differs from releases since 1.05 as follows:

4.15.1. Internal Structure

Major changes have been made in the internal structure of the driver to support Windows operations. If you simply used the drivers as a "black box", this will not effect you. If however, you modified internal code, or depended on undocumented features, you will need to carefully study the new structure

4.15.2. Rep string support

The Mb_chan function has been enhanced to support Smart Cache (rep-string) operations where possible. This increases its maximum possible throughput to the 1 MHz limit of the WIN-30 series.

A new function, Mibh_chan has been added. This uses interrupts to perform rep-string operations in the background. Once again, 1 MHz throughput is supported.

4.15.3. Background DMA support

When using dual channel gap-free DMA, previous version of the driver required that Dma_chk be called periodically. A new function, Mdih_chan, does away with this requirement, allowing the driver to use TC interrupts to perform this function. Thus dual channel DMA can operate entirely in the background.

4.15.4. Windows support

Massive changes, including the support for new languages etc., have been made to support Windows. Windows support is described in the next chapter. However, the basic programming interface remains unchanged!!

4.15.5. Version 1.13

Version 1.13 adds D/A support for WIN-30DA and DS boards. This includes interrupt driven waveform generation.

4.15.6. Version 1.19

Version 1.19 adds continuos data acquisition support for WIN-30DA and DS boards. This includes DMA and Smart-cache operation

4.15.7. Version 1.2x

Version 1.2x adds support for WIN-3016 series.

4.16. Versions prior to 1.05

In addition to the differences noted for versions since 1.06, the following upgrades were made in version 1.05:

4.16.1. New Features

i. Microsoft FORTRAN V5 support.

ii. Microsoft QuickBasic V4.5 support.

iii. Multi-board support.

iv. XMS memory support.

4.16.2. Global variables

In previous versions of the driver software, global variables were declared in the main program, and could be directly accessed. This is no longer the case. UEIDAQ now does not make use of any global variables at all. Programs that made use of these global variables must be modified to perform the same operations via function calls.

Three global variables are involved in this:

4.16.2.1. Base_30.

All references to base_30 must be replaced by function calls to Set_base (to set the board's base address), and Get_base (to read back the base address).

4.16.2.2. Type_30.

All references to type_30 must be replaced by function calls to Set_base (to set the board's type, unless it has been set by the Diag function), and Get_type (to read the board type).

4.16.2.3. Gain_30.

Gain_30 was an array of integer variables in which the gain settings for each input channel were stored. All references to gain_30 must be replaced by function calls to Set_gain (to set the gain of a particular channel), and Get_gain (to read the gain of a selected channel).

4.16.3. XMS support

XMS extended memory managers are now supported by UEIDAQ :

i. This support is done via the old extended memory functions (e_mem_size, mde_chan and xfer_dma_res). These function operate in exactly the same way as in previous versions of the driver, but now will use XMS memory, if available.

ii. A new function, release_e_mem, has been added to provide XMS support. This functions MUST always be called after all data has been transferred to normal memory, in order to indicate that you no longer require the data in question. The mde_chan function will not be able to use the same memory again until this is done. In addition, if a program terminates without calling this function, the memory in question will be "lost" until the PC is rebooted.

   Remember to call the release_e_mem function after using the mde_chan function!! Also remember to call it for EACH board.

iii. Note that XMS memory is block structured; UEIDAQ can use only the largest available block, not all XMS memory. The e_mem_size function returns the size of the largest block.

Chapter 5

5. Windows Programmers Guide

This chapter provides a guide for programmers using the driver system in order to write Windows programs.

5.1. Windows Requirements

5.1.1. Windows 3.1/95 Requirements

Windows support is provided as follows:

i. Windows 3.1 enhanced mode only is supported. Operation under Windows 3 or earlier, or in Windows 3.1 standard mode is not possible.

ii. A minimum configuration of a 386 processor and 4 Mbytes of memory is required. A 386DX or 486 processor and 8 Mbytes of memory are recommended.

iii. Windows 3.1 and Windows 95 support is via a DLL (Dynamic-Link Library) and a virtual device driver (VXD). Both of these must be accessible to Windows for the driver system to operate. The DLL is UEIDAQ.DLL, and the VXD is UEIDAQV.386 for Windows 3.1 and UEIDAQV.VXD for Windows 95.

iv. Windows locates a dynamic-link library by searching the same directories it searches to find an application module. For Windows to the find the library, it must be in one of the following directories, which Windows searches in the order listed:

   a) The current directory.

   b) The Windows directory. (Default: \Windows\

   c) The Windows system directory. (Default: Windows\System\)

   d) Any of the directories listed in the PATH environment variable.

   e) Any directory in the list of directories mapped in a network.

v. Microsoft recommend that DLL's be loaded into the Windows system directory. This is where the default installation program places UEIDAQ.DLL, but any other valid position is acceptable

v. In order for Windows to load UEIDAQV VXD, the following line must appear in the [386ENH] sections of the Windows SYSTEN.INI file : "device=ueidaqv.386" (for Windows 3.1) “device=ueidaqv.vxd” (for Windows 95) . This assumes that the VXD is in the default location, the c:\UEI directory. If it is not, then the line should be modified accordingly. Once again, this is automatically done by the default installation program.

5.1.2. Windows NT Requirements

For Windows NT: UEIDAQ supports NT versions 3.5 and greater. The driver differs from Windows 3.1 and Windows 95 so please read the following very carefully:

The driver is called DAQDRVR.sys and the DLL’s remain UEIDAQ.DLL. The base address and interupt settings are kept in the Windows NT registry. The UEIDAQ for Windows NT installation will will install and modify the registry for you. After your have rebooted the PC, you must start the DAQDRVR driver by entering NET START DAQDRVR at the DOS prompt or you can modify the NT system to start th driver on power-up.

5.1.2.1 MODIFYING DAQDRVR SETTINGS:

The default hardware settings are set at IRQ 5 and Base Address 0x700.

If the following hardware settings were changed, the registry must be updated using regedit.exe.

Follow the steps accordingly:

Double-Click HKEY_LOCAL_MACHINE
Double-Click SYSTEM
Double-Click CurrentControlSet
Double-Click Services
Double-Click Daqdrvr
Double-Click Parameters
Double-Click Daq0
Select InterruptLevel
Right Click you mouse button
Select Modify from the Popup
Goto Value data box
Type in the IRQ(Interrupt Level in HEX) for your board
- if you set your board at IRQ 7
  Type 7
- if you set your board at IRQ12
  TYPE 12

Select OK
Select PortBase
Right Click you mouse button
Select Modify from the Popup
Goto Value data box
Type in the Base Address(Base Address in HEX) for your board
- if you set your board at Base Address 700
  Type 700
- if you set your board at Base Address 300
  TYPE 300

Select OK
Close the Regedit.exe program(alt-r,x)

Your board is now updated in the Registry. Reboot the machine and Start the driver by typing at the Dos Prompt:

```
Net Start Daqdrvr
```

To start the driver automatically without typing Net Stat Daqdrvr every time you boot Windows NT. Follow the steps accordingly:

1. Double-Click HKEY_LOCAL_MACHINE
2. Double-Click SYSTEM
3. Double-Click CurrentControlSet
4. Double-Click Services
5. Double-Click Daqdrvr
6. Select Start
7. Right Click you mouse button
8. Select Modify from the Popup
9. Goto Value data box
10. Type 2(2 for automatic, 3 for manual)
11. Select OK
14. Close the Regedit.exe program(alt-r,x)

The driver is now updated in the Registry for automatic Startup. Reboot the machine and the driver should start automatically.

5.2. Programming for Windows

UEIDAQ for Windows has specifically been written to be as similar as possible to the older DOS drivers. Programmers that have experience with these should experience no difficulty migrating to the Windows versions.

Most of the information as regards function naming, constants etc. given in the previous chapter also apply to Windows. You have a clear understanding of the previous chapter prior to reading this chapter.

Some specific issues that you should be aware of for Windows are the following:

5.2.1. Event driven programming

The major change from DOS programming to Windows programming is adapting to an event driven environment. Under DOS, it is common practice to poll for the end of an event, simply sitting in a program loop until some event occurs. For example, when waiting for an interrupt operation to complete, a typical program might have used:

```
while(int_chk() ==n_comp_30);
```

This simple stalled the system until the interrupt completed. This is no longer acceptable, as it interferes with other processes.

UEIDAQ takes a twofold approach to this problem:

i. UEIDAQ recognizes that for short operations, it isn't worth yielding to other processes. At best, to yield to another process, and wait for a message to restart, imposes at least 100 uS of overhead. Thus, for any function that will take less than 100 uS, UEIDAQ does not yield.

ii. Functions that take longer than this are only those functions that acquire a series of samples. For these functions, UEIDAQ's normal status querying system (Int_chk and Dma_chk) works fine. The simplest way to still use these functions and avoid hanging up Windows is to use a Windows timer to generate a periodic event. On each of these events, the status of the data acquisition function can be checked. All the Windows demo programs supplied with UEIDAQ show how to do this for the various supported languages.

5.2.2. Multi-tasking

A major difference between DOS and Windows is that under Windows multiple programs may all wish to perform data acquisitions tasks. UEIDAQ handles this as follows:

i. Up to eight physical boards may be accessed by up to 16 tasks. Note that every time you run a program, you start a new task.

ii. UEIDAQ allows multiple tasks to access the same board. For example, one task can use the A/D section of a board, while another uses the digital I/O section. Generally, UEIDAQ assumes that the programmer knows what they want, and simply "follows orders" to the best of its ability. It is up to the programmer to ensure that the order make sense.

iii. UEIDAQ does manage both Windows and its own resources such that no conflicts will occur at the driver level. Thus, a programmer can choose to deliberately access the same hardware from two different programs without fear of adversely affecting either Windows or UEIDAQ.

iv. UEIDAQ also intelligently handles board numbers. Two different program can refer to different boards by the same logical number, without conflict. For example, program 1 can refer to a WIN-30 at address 300 as board 0, and program 2 can also refer to another WIN-30 at location 700 as board 0. UEIDAQ can distinguish these boards from one another, and knows to which board a command from either program should be routed to.

v. In order to help UEIDAQ keep track of tasks, you should call the Daq_close function when your program terminates.

5.2.3. Obtaining a series of samples

The current version of UEIDAQ limits support to boards in the WIN-30 series. By far the most efficient way to acquire data using these boards is the Mibh_chan_h function. This function supports 1 MHz throughput even while other Windows programs are running, including programs in DOS boxes. You should not attempt to use any other function to acquire a series of samples under Windows.

The supplied demo programs show how to use this function under all supported languages.

5.2.4. Memory under Windows

If you are acquiring a series of samples, you will need to supply the driver with a memory buffer to place this in. In order to be able to run, this memory needs to be page locked. UEIDAQ supplies two kinds of functions :

i. Functions which take buffer addresses, such as Mibh_chan. This function requires as one of its parameters the address of a buffer. You should NOT use these functions unless you have already page locked the buffer. Note that a buffer can only be page locked from a DLL. Thus, unless you are writing a DLL, you should never use these functions.

ii. UEIDAQ also supplies equivalent functions to all "addressed" functions that take a Windows memory handles rather than a buffer address. All of these functions have names that end in "_h", for example Mibh_chan_h. You should always use these functions rather than addressed functions. UEIDAQ automatically locks such buffers prior to using them. You can either:

   a) Obtain memory yourself, and pass the memory's handle to the driver functions via the programming mechanism.

   b) Or you can use UEIDAQ helper functions described in the next section, if your programming environment doesn't support allocating Global memory handles, or you are unsure how to do this.

5.3. Windows Helper Functions

Some languages, such as Visual Basic, place limits on the size of arrays, or the size of array subscripts, or don't allow Windows memory to be obtained directly. As one of the main advantages of Windows is access to very large amounts of data, UEIDAQ for Windows provides special helper functions to allow such languages to acquire data which is as large as Windows itself supports.

These functions were specifically designed for Visual Basic V2, but are equally useful for any other language.

5.3.1. Basic operation

In essence, the helper functions operate by allowing programs to assign Windows global memory blocks. Each memory block has associated with it a handle, which uniquely identifies the block. This handle is a 16-bit quantity under Windows 3.1, and is represented as an Integer under Visual Basic. Be aware that Global memory blocks are NEVER passed "by value"; it is always the block's handle (in a sense, the address of the block) that is passed. Note that the handle as returned by the helper functions is a standard Windows HGLOBAL type handle; it can be passed to any Windows API function or DLL that requires such a handle.

These global memory handles can then be passed directly to the driver functions that operate on global memory blocks; These are driver functions that have "_h" suffixes, for example Mdh_chan_h.

Once such a block is assigned, you can access the block by using the read and write helper functions.

5.3.2. Block organization

From the perspective of a Visual Basic program, the memory blocks look like very large arrays of either Integer (2 byte) or Single(4 byte) samples. Whether a block consists of Integers or Singles is decided by the function call used to assign the block; it can't be changed later.

Note that only "Integer" blocks should be passed to any driver function that samples data. If you use a "Single" block, you will not be able to access the data that you sampled.

Blocks are organized (and accessed) on a channel and sample basis. Blocks with up to 32 768 channels, and 2 147 483 648 samples per channel are supported.

5.3.3. Initializing the helper system

The helper system MUST be initialized prior to any blocks being assigned. This is done via the vb_InitUEI_mem function. This function must be called once only prior to calling any other helper function.

5.3.4. Assigning memory

Memory is assigned by using either of the following two functions:

vb_allocateUEI_short	This function allocates a block of Integer values
vb_allocateUEI_float	This function allocates a block of Single values

5.3.5. Freeing memory

It is critically important that any memory allocated via the vb_allocate functions be freed prior to your program exiting. If this is not done, then the only way to recover this memory is to reboot Windows.

Memory is freed by using the following function

vb_freeUEI	This frees a memory block. It can be either a Integer or Single block.

5.3.6. Reading samples

Samples are read from the block by the following two functions:

vb_readUEI_short	This reads a single Integer value from an Integer block.
vb_readUEI_float	This reads a single float (or Single) value from a Single block.

5.3.7. Writing samples

Samples are written to the block by the following two functions:

vb_writeUEI_short	This writes a single Integer value to an Integer block.
vb_writeUEI_float	This writes a single Single value to a Single block.

5.3.8. Re-dimensioning the block

The size of an assigned block can be changed via the reallocate functions. If the number of channels remains the same, then previous data is preserved. The reallocate functions are the following :

vb_reallocateUEI_short	Changes the size of an Integer block.
vb_reallocateUEI_float	Changes the size of an Single block.

5.4. Visual C++

All supplied modules are directly compatible with Microsoft Visual C++ version 1.0. All that is required is to link with UEIDAQ.LIB module, which supplies definitions for UEIDAQ DLL. It is possible to directly include the C source code supplied into your programs, but if you do this, the buffer page locking will become ineffective.

Wdemo1.C is supplied with Visual C++ compatible make files, resource files etc.

5.4.1. Required files

PC30.H, UEIDAQ.DLL, UEIDAQ.LIB, UEIDAQV.386

5.4.2. Examples

Wdemo1

5.5. Borland C++

All supplied modules are directly compatible with Borland C++ version 3.1. All that is required is to ensure that your program imports the various functions from UEIDAQ DLL. It is possible to directly include the C source code supplied into your programs, but if you do this, the buffer page locking will become ineffective.

5.5.1. Required files

PC30.H, UEIDAQ.DLL, UEIDAQV.386

5.5.2. Examples

Wdemo1

5.6. Borland Pascal

Borland Pascal version 7.0 can directly utilize UEIDAQ DLL. This is done by simply including the line:

```
Uses UEIDAQ;
```

in your program. This includes UEIDAQ unit, which serves as a wrapper for the functions in UEIDAQ DLL

5.6.1. Required files

UEIDAQ.PAS, UEIDAQ.DLL, UEIDAQV.386

5.6.2. Examples

Wdemo2

5.7. Visual Basic

Visual Basic version 2.0 can directly utilize UEIDAQ DLL. This is done by simply including the lines contained in the file UEIDAQ.INC into the declarations section of your program. This provides declarations for the various functions in UEIDAQ DLL, and for the various constants used by the driver.

Note that Visual Basic does not allow arrays with array indexes of greater than 32K. To overcome this, you should use the Windows helper functions supplied with UEIDAQ. These are described above.

5.7.1. Required files

UEIDAQ.INC, UEIDAQ.DLL, UEIDAQV.386

5.7.2. Examples

Wdemo3

Chapter 6

6. Function Reference

This chapter contains full data on all available functions. Where examples are listed, this refers to the demonstration programs supplied on the distribution disks.

6.1. Ad_chan_cfg

Name :	Ad_chan_cfg **-** configures AD channels
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	int DRVENTRY AD_chan_cfg(int iChanNum, int iADRng, int iADMode, int iGain, int reserved1);
Borland Pascal Usage :	function AD_chan_cfg(iChanNum, iADRng, iADMode, iGain, reserved1 : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC'
	Function ad.chan.cfg %(ByVal ichannum%, BYVAL iadrng%, BYVAL iadmode%, BYVAL igain%, BYVAL reserved1%);
DOS FORTRAN Usage :	INCLUDE 'PC30.fi'
	INCLUDE 'PC30.fd'
	interface to function ad_chan_cfg (ichannum, int iadrng, int iadmode, int igain, int reserved1);
	integer*2 ichannum [VALUE]
	integer*2 iadrng [VALUE]

	integer*2 iadmode [VALUE]
	integer*2 igain [VALUE]
	integer*2 reserved1 [VALUE]
	end
VISUAL BASIC Usage :	Declare Function AD_chan_cfg Lib "ueidaq.dll" (ByVal ichannum%, ByVal iadrng%, ByVal iadmode%, ByVal igain%, ByVal reserved1%) As Integer
Target :	DOS, Windows 3.1
Description :	Configures AD channels for range, mode and gain.

iChanNum is the channel number; -1 configures all channels

iADRng is the A/D range it can be one of the following:

VRANGE5B	-5 to +5V	VRANGE5U	0 to +5V
VRANGE10B	-10 to +10V	VRANGE10U	0 to +10V
VRANGE25B	-2.5 to +2.5V	VRANGE25U	0 to +2.5V

-1 selects the boards default setting: -5 to +5 for the WIN-30.

iADMode selects single ended or differential mode:

VMODESNGL : single mode   VMODEDIF   : differential mode

-1 selects the boards default setting: single ended for the WIN-30Dx, 1 for differential inputs

iGain is the channel gain. This can be either 1,2,4,8 (WIN-30PGxH) or 10,100,1000 (WIN-30PGxL). For the WIN-30Dx the default is 1.

Return value :	None
Prior calls :	Set_base, Diag or config_bd
See Also :	set_osc, ad_clock
Windows Examples :	WDemo1.c
DOS Examples :	Demo20.c

6.2. Ad_clock

Name :	ad_clock - sets the clock divisor.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	void DRVENTRY ad_clock(unsigned clk_val);

Borland Pascal Usage :	procedure ad_clock(clk_val : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB ad.clock(BYVAL clk.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine ad_clock(clk_val) integer*2 clk_val [VALUE] end
Visual Basic Usage	Declare Sub ad_clock Lib "ueidaq.dll" (ByVal clk_val%)
Target :	DOS, Windows 3.1
Description :	Sets the A/D clock divider ratio. This division ratio applies only to the internal clock. The divider value is a 16-bit word, which may range from 2 to 65535. The clock divider works in conjunction with the A/D prescaler to set the frequency of A/D strobes. Example : If the prescaler is set to 10, a clk_val of 20 would give a total clock division ratio of 10 x 20 = 200 On a UEI-30B, UEI-30C, UEI-30D, UEI-30DS, UEI-30DS/4 or UEI-30PG this will give a sample rate of 2000000/(10 x 20) = 10 KHz For a WIN-30 series board, the sample rate would additionally depend on the oscillator setting. This can be either 10 MHz or 2 MHz. If set to 2 MHz, the resulting sample rate would be as in the previous case. If set to 10 MHz, the sampling rate would be: 10000000/(10 x 20) = 50 KHz
Return value :	None
Prior calls :	Set_base
See Also :	Ad_prescaler, set_osc
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo5.c, demo6.c, demo7.c, demo8.c, demo9.c, demo3.pas

6.3 Ad_cont_close

Name :	ad_cont_close- Close continuous acquisition system.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	int DRVENTRY ad_cont_close();
Borland Pascal Usage :	function ad_cont_close : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC'
	FUNCTION ad.cont.close%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi'
	INCLUDE 'PC30.fd'
	interface to function ad_cont_close
	integer*2 ad_cont_close
	end
Visual Basic Usage :	Declare Function ad_cont_close Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	Shuts down continuous acquisition. If DMA is used then The PC's DMA controller is disabled, the board's DMA subsystem disabled, and the data in the buffer aligned and checked for errors. Note that prior to calling this function, the position of the data in the buffer is unpredictable. This procedure must be called after each mdih_cont operation (single channel or gap-free dual channel), regardless of whether the operation was completed and whether DMA or RepString was enabled.
Return value :	ok_30 - operation completed.
	err_30 - error in operation.
	task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	mdih_chan, mdih_cont , DMA_close
Windows Examples :	Wdemo1.c
DOS Examples :	Demo20.c

6.4. Ad_in

Name :	ad_in - Gets a single A/D input reading.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY ad_in(int chan, int *in_val);

Borland Pascal Usage :	procedure ad_in(chan : integer, var in_val : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION ad.in%(BYVAL chan%, SEG in.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function ad_in (chan, in_val) integer*2 ad_in integer*2 chan [VALUE] integer*2 in_val [REFERENCE] end
Visual Basic Usage :	Declare Function ad_in Lib "ueidaq.dll" (ByVal chan%, in_val%) As Integer
Target :	DOS, Windows 3.1
Description :	This function obtains a sample from channel chan, and places the result in the integer variable in_val. Note that for the UEI-127, this returns only a single sample, even though the hardware actually performs conversions on all four channels. The extra three samples are simply discarded.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. err_30 - A/D hardware error. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	ad_in_1, ad_in_2.
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas

DOS Examples :	demo2.pas

6.5. Ad_in_1

Name :	ad_in_1 - Reads a single A/D value.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY ad_in_1(int chan, int *in_val);

Borland Pascal Usage :	procedure ad_in_1(chan : integer, var in_val : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION ad.in.1%(BYVAL chan%, SEG in.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function ad_in (chan, in_val) integer*2 ad_in integer*2 chan [VALUE] integer*2 in_val [REFERENCE] end
Visual Basic Usage :	Declare Function ad_in_1 Lib "ueidaq.dll" (ByVal chan%, in_val%) As Integer
Target :	DOS, Windows 3.1
Description :	This function is similar to ad_in, but is used to obtain a single A/D sample from an up to 256 channel system which uses a single layer of sub-multiplexed UEI-81 expansion boards. The lower 4 bits of digital output port A are used to address the extender boards. For UEI-30PG boards, the channel gain is selected by the lower 4 bits of the channel address. For UEI-127 boards, only a 64 channel system is possible with this technique.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. err_30 - A/D hardware error. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	ad_in, ad_in_1.

6.6. Ad_in_2

Name :	ad_in_2 - Reads a single A/D value.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	int DRVENTRY ad_in_2(int chan, int *in_val);

Borland Pascal Usage :	procedure ad_in_2(chan : integer, var in_val : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION ad.in.2%(BYVAL chan%, SEG in.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function ad_in_2 (chan, in_val) integer*2 ad_in integer*2 chan [VALUE] integer*2 in_val [REFERENCE] end
Visual Basic Usage :	Declare Function ad_in_2 Lib "ueidaq.dll" (ByVal chan%, in_val%) As Integer
Target :	DOS, Windows 3.1
Description :	This function is similar to ad_in, but is used to obtain a single A/D sample from an up to 4096 channel system which uses a double layer of sub-multiplexed UEI-81 expansion boards. The 8 bits of digital output port A are used to address the extender boards. For UEI-30PG boards, the channel gain is selected by the lower 4 bits of the channel address. For UEI-127 boards, only a 1024 channel system is possible with this configuration.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. err_30 - A/D hardware error. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	ad_in, ad_in_1.

6.7. Ad_prescaler

Name :	ad_prescaler - sets the clock prescaler.

Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	void DRVENTRY ad_clock(unsigned clk_val);

Borland Pascal Usage :	procedure ad_prescaler(clk_val : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB ad.prescaler(BYVAL clk.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine ad_prescaler(clk_val) integer*2 clk_val [VALUE] end
Visual Basic Usage :	Declare Sub ad_prescaler Lib "ueidaq.dll" (ByVal clk_val%)
Target :	DOS, Windows 3.1
Description :	Sets the clock prescaler ratio. This division ratio applies only to the internal clock. On UEI-30B, UEI-30C, UEI-30D, UEI-30DS, UEI-30DS/4 and UEI-30PG boards the input to the prescaler is a fixed 2 MHz master clock. On WIN-30 series boards, the input to the prescaler is software selectable to be either 2 MHz or 10 MHz. The prescaler value is a 16-bit word, which may range from 2 to 65535. The clock divider works in conjunction with the A/D prescaler and oscillator selection function to set the frequency of A/D strobes.
Return value :	None
Prior calls :	Set_base
See Also :	Ad_clock, Set_osc.
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo5.c, demo6.c, demo7.c, demo8.c, demo9.c, demo3.pas

6.8. Ch_list_load

Name :	Ch_list_load - Loads the channel list and block counter.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY ch_list_load(int c_list[], int burst_len);

Borland Pascal Usage :	procedure ch_list_load(var list : integer; burst_len : integer);

DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB ch.list.load(SEG c.list%, BYVAL burst.len%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine ch_list_load(c_list, burst_len) integer*2 c_list [REFERENCE] integer*2 burst_len [VALUE] end
Visual Basic Usage :	Declare Sub ch_list_load Lib "ueidaq.dll" (c_list%, ByVal burst_len%)
Target :	DOS, Windows 3.1
Description :	This procedure loads the hardware channel list with channel information from c_list. The block length is loaded into the block counter. If the block length is 0, then the A/D mode is set to 2 (replace). If the block length is non-zero, then the A/D mode is set to 1 (block trigger mode). The channel list is an integer array of maximum length 31, terminated by a channel number greater than 15. The ch_list_load function is used internally, and is not required prior to any other function described in this manual. It may however be of use to users wishing to write their own code.
Return value :	None
Prior calls :	Set_base
See Also :	mb_chan, mdb_chan.

6.9. Clean

Name :	Clean - Clears the board's A/D and D/A subsystems in preparation for a conversion.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY clean();

Borland Pascal Usage :	procedure clean;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB clean
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine clean end

Visual Basic Usage :	Declare Sub Clean Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	This procedure disables interrupts and DMA, and then waits long enough to ensure that no conversion is in progress. The error flag is then cleared. This procedure need not be called prior to the use of any of the other driver functions, but is supplied to assist users who wish to write programs which interact directly with the hardware.
Return value :	None
Prior calls :	Set_base, diag or set_type
See Also :	Dma_init.

6.10. Cntr_cfg

Name :	Cntr_cfg - Configures the uncommitted counter/timer.
Boards Sup ported :	WIN-1016/WIN-3016

C Usage :	#include <PC30.H>
	
	void DRVENTRY cntr_cfg(int mode);

Borland Pascal Usage :	procedure cntr_cfg(mode : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB cntr.cfg(BYVAL mode%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine cntr_cfg(mode) integer*2 mode [VALUE] end
Visual Basic Usage :	Declare Sub cntr_cfg Lib "ueidaq.dll" (ByVal mode%)
Target :	DOS, Windows 3.1
Description :	Configures the uncommitted counter into one of six modes. Valid mode settings are from 0 to 5.
Return value :	None
Prior calls :	Set_base, Init, Diag or Set_type
See Also :	cntr_read, cntr_write.

6.11. Cntr_read

Name :	Cntr_read - Reads the uncommitted counter/timer.
Boards Supported :	WIN-1016/WIN-3016

C Usage :	#include <PC30.H>
	
	unsigned DRVENTRY cntr_read(void);

Borland Pascal Usage :	function cntr_read: word;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION cntr.read%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function cntr_read integer*2 cntr_read end
Visual Basic Usage :	Declare Function cntr_read Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	Latches and reads the value in the uncommitted counter/timer. This can take on values from 0 to FFFF (hex).
Return value :	Counter contents.
Prior calls :	Set_base, Init, Diag or Set_type
See Also :	cntr_cfg, cntr_write.

6.12. Cntr_write

Name :	Cntr_write - Writes a value to the uncommitted counter/timer.
Boards Supported :	WIN-1016/WIN-3016

C Usage :	#include <PC30.H>
	
	void DRVENTRY cntr_write(unsigned val);

Borland Pascal Usage :	procedure cntr_write(val : word);
DOS QuickBasic Usage :	SUB cntr.write(BYVAL c.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi'

	INCLUDE 'PC30.fd' interface to subroutine cntr_write(val) integer*2 val [VALUE] end
Visual Basic Usage :	Declare Sub cntr_write Lib "ueidaq.dll" (ByVal c_val%)
Target :	DOS, Windows 3.1
Description :	Writes a value to the uncommitted counter/timer. This can take on values from 0 to FFFF (hex).
Return value :	None.
Prior calls :	Set_base, Init, Diag or Set_type
See Also :	cntr_cfg, cntr_read.

6.13. Clk_md

Name :	clk_md - Not used.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY clk_md(int clock_md);

Borland Pascal Usage :	procedure clk_md(clock_md : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB clk.md(BYVAL clock.md%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine clk_md(clock_md) integer*2 clock_md [VALUE] end
Visual Basic Usage :	Declare Sub clk_md Lib "ueidaq.dll" (ByVal clock_md%)
Target :	DOS, Windows 3.1
Description :	This procedure is a dummy procedure, supplied for compatibility with other driver packages.
Return value :	None
Prior calls :	Set_base, Init, Diag or Set_type

See Also :

Windows Examples :

DOS Examples :

6.14. Config_bd

Name :	config_bd - configures software configurable boards
Boards Supported :	All WIN boards

C Usage :

```
#include <PC30.H>
int DRVENTRY config_bd(int irq, int dma_l, int dma_sl, int int_source, int dma_source, int
ad_clk_src, int ad_rng, int ad_i_mode, int da_clk_src, int da_rng);
```

Borland Pascal Usage :

```
function config_bd(irq, dma_l, dma_sl, int_source, dma_source, ad_clk_src, ad_rng,
ad_i_mode, da_clk_src, da_rng : integer) : integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

Function config.bd %(ByVal irq%, ByVal dma.l%, ByVal dma.sl%, ByVal int.source%,
ByVal dma.source%, ByVal ad.clk.src%, ByVal ad.rng%, ByVal ad.i.mode%, ByVal
da.clk.src%, ByVal da.rng%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function config_bd (irq, dma_l, dma_sl, int_source, dma_source, ad_clk_src,
ad_rng, ad_i_mode, da_clk_src, da_rng)
integer*2 irq [VALUE]
integer*2 dma_l [VALUE]
integer*2 dma_sl [VALUE]
integer*2 int_source [VALUE]
integer*2 dma_source [VALUE]
integer*2 ad_clk_src [VALUE]
integer*2 ad_rng [VALUE]
integer*2 ad_i_mode [VALUE]
integer*2 da_clk_src [VALUE]
integer*2 da_rng [VALUE]
end
```

Visual Basic Usage :

```
Declare Function config_bd Lib "ueidaq.dll" (ByVal irq%, ByVal dma_l%, ByVal dma_sl%,
ByVal int_source%, ByVal dma_source%, ByVal ad_clk_src%, ByVal ad_rng%, ByVal
ad_i_mode%, ByVal da_clk_src%, ByVal da_rng%) As Integer
```

Target : DOS, Windows 3.1

Description : Configures boards that support programmable configuration. For the WIN-30 series, the parameters are as follows:

irq : Interrupt level (0, 3, 5, 7, 10, 11, 12, 14, 15)

	dma_l : Primary DMA level (0, 5, 6, 7)
	dma_sl : Secondary DMA level (0, 5, 6, 7)
	int_source : Interrupt source : 0 - A/D done, 1- DMA TC, 2-Smart cache
	dma_source : not used
	ad_clk_scr : A/D clock source : 0 - external, 1 - internal
	ad_rng : A/D input range : 0 - -5 to +5 V, 1 - 0 to 5V
	ad_i_mode : not used
	da_clk_src : D/A clock source : 0 - external, 1 - internal
	da_rng : not used
	Functions or board features not used should be set to 0. Note that this function cannot be called while another driver function is busy.
Return value :	None
Prior calls :	Set_base, Diag
See Also :	Set_osc.
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo5.c, demo6.c, demo7.c, demo8.c, demo9.c, demo3.pas

6.15. Daq_close

Name :	Daq_close - Closes the driver system.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	void DRVENTRY daq_close();
Borland Pascal Usage :	procedure daq_close;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC'
	SUB daq_close
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd'
	interface to subroutine daq_close end

Visual Basic Usage :	Declare Sub Daq_close Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	This procedure informs the driver that the calling program will not require any further access to any data acquisition hardware under the driver's control. This allows the driver to grant access to other programs. Calling this function is optional under DOS, but required under Windows.
Return value :	None
Prior calls :	None
See Also :	

6.16. Daq_version

Name :	daq_version - returns the driver version number.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY daq_version();

Borland Pascal Usage :	function daq_version : integer
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION daq_version%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function daq_version integer*2 version end
Visual Basic Usage :	Declare Function daq_version Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	Returns the version number of the driver, encoded as follows : High byte : Major version number. Low byte : Minor version number. For example, version 2.06 would return 0206 hex.
Return value :	Integer representing the version number.

Prior calls :	None.
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo1.pas, demo2.pas, demo3.pas, demo4.c, demo5.c, demo6.c

6.17. Da_clock

Name : da_clock - sets the D/A clock divisor.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>
void DRVENTRY da_clock(unsigned clk_val);
```

Borland Pascal Usage :

```
procedure da_clock(clk_val : integer);
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

SUB da.clock(BYVAL clk.val%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to subroutine da_clock(clk_val)
integer*2 clk_val [VALUE]
end
```

Visual Basic Usage :

```
Declare Sub da_clock Lib "ueidaq.dll" (ByVal clk_val%)
```

Target : DOS, Windows 3.1

Description : Sets the D/A clock divider ratio. This division ratio applies only to the internal clock.

The divider value is a 16-bit word, which may range from 2 to 65535. The clock divider works in conjunction with the D/A prescaler to set the frequency of D/A strobes.

Example : If the prescaler is set to 10, a clk_val of 20 would give a total clock division ratio of

10 x 20 = 200

On a UEI-30B, UEI-30C, UEI-30D, UEI-30DS, UEI-30DS/4 or UEI-30PG this will give a data rate of

2000000/(10 x 20) = 10 KHz

For a WIN-30 series board, the sample rate would additionally depend on the oscillator setting. This can be either 10 MHz or 2 MHz. If set to 2 MHz, the resulting sample rate would be as in the previous case. If set to 10 MHz, the sampling rate would be:

	10000000/(10 x 20) = 50 KHz
Return value :	None
Prior calls :	Set_base
See Also :	Config_bd.
Windows Examples :	
DOS Examples :	Demo5.c, demo6.c, demo7.c, demo8.c, demo9.c, demo3.pas

6.18. Da_out

Name : da_out - outputs a value to a D/A.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY da_out(int chan, int out_val);
```

Borland Pascal Usage :

```
function da_out(chan, out_val : integer) : integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION da.out%(BYVAL chan%, BYVAL out.val%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function da_out (chan, out_val)
integer*2 da_out
integer*2 chan [VALUE]
integer*2 out_val [VALUE]
end
```

Visual Basic Usage :

```
Declare Function da_out Lib "ueidaq.dll" (ByVal chan%, ByVal out_val%) As Integer
```

Target : DOS, Windows 3.1

Description : Out_val is sent to D/A chan. Values for chan are 0, 1, 2 or 3. Coding is offset binary. On UEI-30 series boards channels 0 and 1 are 12-bit converters, and channels 2 and 3 are 8-bit converters. On WIN-30 series boards channels 0 and 1 are 16-bit converters, and channels 2 and 3 are 12-bit converters. Note that for the 8-bit converters bits 11 thru 4 are of out_val are used to write to the converter. The lower four bits are discarded.

Return value :

ok_30 - operation performed

par_30 - Invalid channel number.

task_30 - Too many tasks accessing the data acquisition hardware.

Prior calls :	Set_base, diag or set_type
See Also :	Wfm_chan
Windows Examples :	
DOS Examples :	demo2.pas, demo3.pas

6.19. Diag

Name : diag - diagnostics.

Warning
The function Diag must be called prior to using any other driver function, in order to initialize the board type variable. Failure to do this will result in unpredictable results.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY diag();
```

Borland Pascal Usage :

```
function diag : integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION diag%
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function diag
integer*2 diag
end
```

Visual Basic Usage :

```
Declare Function diag Lib "ueidaq.dll" () As Integer
```

Target : DOS, Windows 3.1

Description : Executes diagnostic tests on the UEI-30. These tests do not require any specific jumper settings, but will destroy the contents of all the UEI-30 registers.

Return value :

0 - UEI-30 OK.

task_30 - Too many tasks accessing the data acquisition hardware.

Any other value - Board faulty or not found.

Prior calls : Set_base

See Also : Set_type

Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo1.pas, demo2.pas, demo3.pas, demo4.c, demo5.c, demo6.c

6.20. D_in

Name :	d_in - Digital input.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY d_in(int chan);

Borland Pascal Usage :	function d_in(chan : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION d.in%(BYVAL chan%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function d_in (chan) integer*2 d_in integer*2 chan [VALUE] end
Visual Basic Usage :	Declare Function d_in Lib "ueidaq.dll" (ByVal chan%) As Integer
Target :	DOS, Windows 3.1
Description :	Obtains the value on the digital input port. A chan value of 0 selects port A, 1 selects port B, and 2 port C. Note that the digital port must be configured by the d_mode function.
Return value :	Integer representing the digital input value.
Prior calls :	Set_base, Diag or Set_type
See Also :	d_out, d_mode.
Windows Examples :	Wdemo1.c

6.21. DMA_chk

Name :	dma_chk - Checks for DMA completion.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY dma_chk;

Borland Pascal Usage :	function dma_chk : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION dma.chk%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function dma_chk integer*2 dma_chk end
Visual Basic Usage :	Declare Function dma_chk Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	Checks whether a DMA operation has completed and, in the case of gap-free dual DMA channel operations, reprograms the DMA controller when required. In the case of polled gap-free DMA operations (as initiated by mdh_chan and mde_chan), it is the user's responsibility to call the dma_chk often enough to ensure that the DMA controller is reprogrammed before the channel in question is required. In practice, dma_chk should be called at least every 20 mS. If however the DMA operation was initiated by the mibh_chan function, DMA controller reprogramming is performed in the background by the driver system itself. In this case there are no timing requirement on calling dma_chk; it is simply a status function.
Return value :	ok_30 **-** operation completed. par_30 **-** Invalid DMA level. n_comp_30 **-** operation not yet completed. task_30 **-** Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	sd_chan, dma_init, mdb_chan, mdh_chan, mde_chan.

6.22. DMA_close

Name :	dma_close **-** Close the DMA system.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>

	int DRVENTRY dma_close();
Borland Pascal Usage :	function dma_close : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION dma.close%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function dma_close integer*2 dma_close end
Visual Basic Usage :	Declare Function dma_close Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	Shuts down the DMA system. The PC's DMA controller is disabled, the board's DMA subsystem disabled, and the data in the buffer aligned and checked for errors. Note that prior to calling this function, the position of the data in the buffer is unpredictable. This procedure must be called after each DMA operation (single channel or gap-free dual channel), regardless of whether the operation was completed.
Return value :	ok_30 - operation completed. err_30 - error in operation. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	sd_chan, dma_init, mdb_chan, mdh_chan, mde_chan.
Windows Examples :	
DOS Examples :	Demo6.c, demo7.c, demo8.c, demo9.c

6.23. DMA_init

Name :	dma_init - Initializes the DMA system.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY dma_init();

Borland Pascal Usage :	procedure dma_init;

DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB dma.init
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine dma_init end
Visual Basic Usage :	Declare Sub dma_init Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	This initializes internal storage prior to any DMA operations. Programs which use DMA should call this once, prior to any DMA operation.
Return value :	None.
Prior calls :	None.
See Also :	sd_chan, dma_chk, mdb_chan, mdh_chan, mde_chan.
Windows Examples :	
DOS Examples :	Demo6.c, demo7.c, demo8.c, demo9.c

6.24. D_out

Name :	d_out **-** Digital output.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> void DRVENTRY d_out(int chan, int out_val);
Borland Pascal Usage :	procedure d_out(chan, out_val : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB d.out(BYVAL chan%, BYVAL out.val%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine d_out(chan, out_val) integer*2 chan [VALUE] integer*2 out_val [VALUE] end

Required prior calls	Init, Diag or Set_type
Visual Basic Usage :	Declare Sub d_out Lib "ueidaq.dll" (ByVal chan%, ByVal out_val%)
Target :	DOS, Windows 3.1
Description :	Outputs out_val to the digital output port specified by chan. The MSB is ignored. Note that the digital port must be configured by the d_mode function.
Return value :	None
Prior calls :	Set_base, diag or set_type
See Also :	d_in, d_mode.
Windows Examples :	Wdemo1.c

6.25. D_mode

Name :	d_mode **-** Digital I/O mode setting.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY d_mode(int mode_a, int mode_b, int mode_c);

Borland Pascal Usage :	procedure d_mode(porta, portb, portc : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB d.mode(BYVAL porta%, BYVAL portb%, BYVAL portc%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine d_mode(mode_a, mode_b, mode_c) integer*2 mode_a [VALUE] integer*2 mode_b [VALUE] integer*2 mode_c [VALUE] end
Required prior calls	Init, Diag or Set_type
Visual Basic Usage :	Declare Sub d_mode Lib "ueidaq.dll" (ByVal mode_a%, ByVal mode_b%, ByVal mode_c%)
Target :	DOS, Windows 3.1
Description :	Sets the mode of the three digital I/O ports. Only simple mode 0 I/O is supported. If the mode variables are set to 1, then the port in question is an input. If the variable is a 0, then the port is an output.

	The UEI-126 and UEI-127 do not use this function; port 0 is fixed as an input, and port 1 as an output.
Return value :	None
Prior calls :	Set_base
See Also :	d_in, d_out.
Windows Examples :	Wdemo1.c

6.26. E_mem_size

Name :	E_mem_size - Extended memory available.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	long DRVENTRY e_mem_size(void);

Borland Pascal Usage :	function e_mem_size : longint;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION e.mem.size&
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function e_mem_size integer*4 e_mem_size end
Required prior calls	Init, Diag or Set_type
Target :	DOS
Description :	E_mem_size returns the number of samples which can be stored in extended memory.
Return value :	Number of words of extended memory available. This function checks both for BIOS allocated memory (INT 15) and for XMS memory.
Prior calls :	
See Also :	Mde_chan.

6.27. Get_base

Name :	Get_base - Retrieves the board base address, as set by the set_base function.

Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY get_base(void);

Borland Pascal Usage :	function get_base : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION get.base
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function get_base integer*2 get_base end
Visual Basic Usage :	Declare Function get_base Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	This function retrieves the board base address as set by the set_base function.
Return value :	Base address
Prior calls :	Set_base
See Also :	Set_base
Windows Examples :	
DOS Examples :	

6.28. Get_gain

Name :	Get_gain **-** Retrieves the channel gain as set by the set_gain function.
Boards Supported :	All WIN-PG boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY get_gain(index);

Borland Pascal Usage :	function get_gain(index : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION get.gain%(BYVAL index%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi'

```
INCLUDE 'PC30.fd'

interface to function get_gain(index)
integer*2 get_gain
integer*2 index
end
```

Visual Basic Usage : Declare Function get_gain Lib "ueidaq.dll" (ByVal index%) As Integer

Target : DOS, Windows 3.1

Description : This function retrieves the gain of the input channel referred to by index. Gain codes are described in section 3.2.

Return value : Gain code for the selected channel.

Prior calls : set_gain

See Also : Set_gain

Windows Examples :

DOS Examples :

6.29. Get_osc

Name : Get_osc - Retrieves the master oscillator value as set by the set_osc function.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY get_osc(void);
```

Borland Pascal Usage : function get_osc() : integer;

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION get.gain%
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function get_osc()
integer*2 get_osc
end
```

Visual Basic Usage : Declare Function get_osc Lib "ueidaq.dll" () As Integer

Target : DOS, Windows 3.1

Description : This function retrieves the frequency of the master oscillator for WIN-30 series boards.

Return value :	Master oscillator frequency in hundreds of KHz. Possible returns are 100 (10 MHz) and 20 (2 MHz)
Prior calls :	Set_base, diag
See Also :	Set_osc
Windows Examples :	
DOS Examples :	

6.30. Get_type

Name :	Get_type **-** Retrieves the board type, as determined by the Diag function, or set by the Set_type function.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY get_type(void);

Borland Pascal Usage :	function get_type : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION get.type
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function get_type integer*2 get_type end
Visual Basic Usage :	Declare Function get_type Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	This function gets the board type as detected by the Diag function or set by the Set_type function. Either Diag or Set_type must have been called prior to calling this function.
Return value :	7 – WIN-30D or WIN-10D 8 **-** WIN-30PGL or WIN-30PGSL 9 **-** WIN-30PGH or WIN-30PGSH
Prior calls :	Diag or set_type

See Also :	Diag
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo11.bas

6.31. Init

Name :	init - initializes the board.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY init();

Borland Pascal Usage :	procedure init;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB init
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine init end
Required prior calls	None
Visual Basic Usage :	Declare Sub init Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	Initializes the board to a known state. It should be called before any other functions (other than diagnostics).
Return value :	None
Prior calls :	Set_base, diag or set_type
See Also :	clean
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo2.pas, demo3.pas, demo5.c, demo6.c

6.32. Int_chk

Name :	int_chk - checks for interrupt operation completion.

Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY int_chk();

Borland Pascal Usage :	Not available.
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION int.chk%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function int_chk integer*2 int_chk end
Visual Basic Usage :	Declare Function int_chk Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	Checks whether an interrupt operation has completed, and cleans up the interrupt vectors etc.
Return value :	ok_30 - operation completed. err_30 - Hardware error. n_comp_30 - operation not yet completed. par_30 - no interrupt operation in progress. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	mi_chan, mibh_chan, wfmih_chan.
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo5.c

6.33. Int_close

Name :	int_close - Disable interrupts and restore interrupt vectors.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY int_close();

Borland Pascal Usage :	Not available.
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION int.close%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function int_close integer*2 int_close end
Visual Basic Usage :	Declare Sub int_close Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	Disable board interrupts, and restores the interrupt vectors.
Return value :	None.
Prior calls :	Set_base, diag or set_type
See Also :	mi_chan, int_chk.
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo5.c

6.34. Load_dsp

Name :	load_dsp - Downloads code to the on-board DSP processor.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	iint DRVENTRY load_dsp(int board_num, unsigned char __far *dsp_dat);

Borland Pascal Usage :	function load_dsp(board_num : integer; var dsp_dat : char) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION load.dsp%(BYVAL board.num%, dsp.dat%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function load_dsp (board_num, dsp_dat) integer*2 load_dsp integer*2 board_num [VALUE] integer*2 dsp_dat

end

Visual Basic Usage : Declare Function load_dsp Lib "ueidaq.dll" (ByVal board_num%, dspdat%) As Integer

Target : DOS, Windows 3.1

Description : The data pointed to dsp_dat is down loaded to the DSP processor on the board identified by board_num. The data must be BDS format. This is a proprietary UEI format.

Normally, this function is not required by users of the driver system. The Diag function, which all programs should call, automatically performs this function.

Return value : ok_30 **-** operation performed

par_30 **-** Board does not have a DSP processor.

task_30 **-** Too many tasks accessing the data acquisition hardware.

Prior calls : Set_base, diag or set_type

See Also :

Windows Examples :

DOS Examples :

6.35. Load_gain

Name : load_gain **-** Loads the channel gains into boards which support programmable gain.

Boards Supported : UEI-30PG

C Usage : #include <PC30.H>

void DRVENTRY load_gain();

Borland Pascal Usage : procedure load_gain;

DOS QuickBasic Usage : REM $INCLUDE: 'PC30.INC'

SUB load.gain

DOS FORTRAN Usage : INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to subroutine load_gain
end

Visual Basic Usage : Declare Sub load_gain Lib "ueidaq.dll" ()

Target : DOS, Windows 3.1

Description :	Loads the channel gain (as previously set by the Set_gain function) into boards which support programmable gain. This function need not be called prior to using any other driver function, as all the data acquisition functions include calls to this function automatically. It is included only to aid users wishing to write their own low-level code.
Return value :	None
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	Set_gain
Windows Examples :	
DOS Examples :	

6.36. Mb_chan

Name : mb_chan - Multiple channel block mode A/D input.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY mb_chan(int chan_list[], int n_samp, int burst, int *f_sample);
```

Borland Pascal Usage :

```
function mb_chan(var chan_list; n_samp : integer; burst : integer; var f_sample : integer) :
integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION mb.chan%(SEG chan.list%, BYVAL n.samp%, BYVAL burst%, SEG
f.sample%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function mb_chan(c_l, n_s, b_l, f_s)
integer*2 mb_chan
integer*2 c_l[REFERENCE]
integer*2 n_s[VALUE]
integer*2 b_l[VALUE]
integer*2 f_s[REFERENCE]
end
```

Visual Basic Usage :

```
Declare Function mb_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp%, ByVal
burst_len%, ByVal hgBuf%) As Integer
```

Target : DOS, Windows 3.1

Description : This function is provided for compatibility only; new programs should use mbh_chan.

This function obtains a sequence of samples from several channels, optionally in block (or burst) mode. A list of channels is given by the parameter chan_list, and the number of samples by the parameter n_samp. The parameter f_sample is the integer variable into which the first sample is written. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Once the last channel is sampled, the first channel is sampled again, until a total of n_samp conversion have been performed.

The burst parameter sets the block length. If this parameter is 0, normal mode is used. For boards which support simultaneous sampling, this takes effect in burst mode.

If the channel length is greater than 31, or the board is not a UEI-30B, UEI-30C UEI-30D, UEI-30DS, UEI-30DS/4, UEI-30PG or WIN-30, conventional channel mode is selected, and the burst parameter is ignored.

The UEI-127 always operates with a burst length equal to the length of the channel list, regardless of the burst parameter.

Return value :	ok_30 **-** operation performed
	par_30 **-** Invalid channel number.
	err_30 **-** A/D hardware error.
	task_30 **-** Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	m_chan, mb_chan.
Windows Examples :	
DOS Examples :	

6.37. Mbh_chan, Mbh_chan_h

Name :	mbh_chan **-** Multiple channel block mode A/D input for data lengths of greater than 32 Ksamples.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	int DRVENTRY mbh_chan(int chan_list[], int n_samp, int burst, int *f_sample);
Borland Pascal Usage :	function mbh_chan(var chan_list; n_samp : integer; burst : integer; var f_sample : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC'
	FUNCTION mb.chan%(SEG chan.list%, BYVAL n.samp%, BYVAL burst%, SEG f.sample%)

DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function mbh_chan(c_l, n_s, b_l, f_s) integer*2 mbh_chan integer*2 c_l[REFERENCE] integer*4 n_s[VALUE] integer*2 b_l[VALUE] integer*2 f_s[REFERENCE] end
Windows C Usage :	int DRVENTRY mbh_chan_h(int __far c_list[], int n_samp, int burst_len, HGLOBAL hgBuf);
Visual Basic Usage :	Declare Function mbh_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp%, ByVal burst_len%, ByVal hgBuf%) As Integer
Target :	DOS, Windows 3.1
Description :	Mbh_chan is very similar to mb_chan, but supports huge addressing. This function obtains a sequence of samples from several channels, optionally in block (or burst) mode. A list of channels is given by the parameter chan_list, and the number of samples by the parameter n_samp. The parameter f_sample is the integer variable into which the first sample is written. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Once the last channel is sampled, the first channel is sampled again, until a total of n_samp conversion have been performed. The burst parameter set the block length. If this parameter is 0, normal mode is used. For boards which support it, simultaneous sampling takes effect in burst mode. If the channel length is greater than 31, or the board is not a UEI-30B, UEI-30C UEI-30D, UEI-30DS, UEI-30DS/4, UEI-30PG or WIN-30, conventional channel mode is selected, and the burst parameter is ignored. Where possible, mbh_chan uses Smart Cache operation to speed up data acquisition on WIN-30 series boards, and can support 1 MHz sampling rates under these conditions. This is only possible if the channel length is less than 32. Mbh_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. err_30 - A/D hardware error. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	m_chan, mb_chan.

Windows Examples :

DOS Examples :

6.38. Mibh_chan, Mibh_chan_h

Name :	mibh_chan - Multiple channel block mode A/D input via interrupts and Smart Cache operation
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> int DRVENTRY mibh_chan(int chan_list[], int n_samp, int burst, int __huge *f_sample, int i_lvl);
Borland Pascal Usage :	function mibh_chan(var chan_list; n_samp : integer; burst : integer; var f_sample : integer, i_lvl : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION mibh.chan%(SEG chan.list%, BYVAL n.samp%, BYVAL burst%, SEG f.sample%, BYVAL i.lvl%))
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function mibh_chan(c_l, n_s, b_l, f_s, i_l) integer*2 mibh_chan integer*2 c_l[REFERENCE] integer*4 n_s[VALUE] integer*2 b_l[VALUE] integer*2 f_s[FAR,. REFERENCE] integer*2 i_l[VALUE] end
Windows C Usage :	int DRVENTRY mibh_chan_h(int __far c_list[], int n_samp, int burst_len, HGLOBAL hgBuf, int i_lvl);
Visual Basic Usage :	Declare Function mibh_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp&, ByVal burst_len%, ByVal hgBuf%, ByVal irq%) As Integer
Target :	DOS, Windows 3.1
Description :	This function obtains a sequence of samples from several channels, optionally in block (or burst) mode. A list of channels is given by the parameter chan_list, and the number of samples by the parameter n_samp. The parameter f_sample is the integer variable into which the first sample is written. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Once the last channel is sampled, the first channel is sampled again, until a total of n_samp conversion have been performed. This is a background mode task that uses the interrupt system to operate. Function int_chk is

called to determine whether the operation has completed. Note that calling mibh_chan only starts the operation. No actual data transfer takes place until after this routine completes. Note also that procedure int_close must be called to clean up the interrupt vectors etc., prior to program termination.

Values of 2, 3, 5, 7, 10, 11, 12, 14 and 15 are valid for the interrupt level. This level must be the same as set by jumpers on the UEI-30 board, or the configuration values set by a call to config_bd in the case of a WIN-30.

The burst parameter set the block length. If this parameter is 0, normal mode is used. For boards which support it, simultaneous sampling takes effect in burst mode.

If the channel length is greater than 31, or the board is not a UEI-30B, UEI-30C UEI-30D, UEI-30DS, UEI-30DS/4 or UEI-30PG, conventional channel mode is selected, and the burst parameter is ignored.

Mibh_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.

The UEI-127 always operates with a burst length equal to the length of the channel list, regardless of the burst parameter.

Return value :	ok_30 - operation performed
	par_30 - Invalid channel number.
	err_30 - A/D hardware error.
	task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	m_chan, mb_chan.
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo18.c

6.39. M_chan

Name :	m_chan - Multiple channel A/D input.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	int DRVENTRY m_chan(int chan_list[], int n_samp, int *f_sample);
Borland Pascal Usage :	function m_chan(var chan_list; n_samp : integer; var f_sample : integer) : integer;
DOS QuickBasic Usage :	not supported, use Mb_chan instead.

DOS FORTRAN Usage :	not supported, use Mb_chan instead.
Visual Basic Usage :	not available
Target :	DOS, Windows 3.1
Description :	This function obtains a sequence of samples from several channels. A list of channels is given by the parameter chan_list, and the number of samples by the parameter n_samp. The parameter f_sample is the integer variable into which the first sample is written. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Once the last channel is sampled, the first channel is sampled again, until a total of n_samp conversion have been performed.
	This procedure is implemented as a macro for C. It simply calls mb_chan, with a burst length of 0.
	Example : in order to sample channels 1, 5 and 6 twice, n_samp would be 6, and the channel list as follows :
	location 0 - 1
	location 1 - 5
	location 2 - 6
	location 3 - 16
	For the UEI-127, this function operates in burst mode.
Return value :	ok_30 - operation performed
	par_30 - Invalid channel number.
	err_30 - A/D hardware error.
	task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	mi_chan, mb_chan.
Windows Examples :	
DOS Examples :	Demo3.pas

6.40. Mdb_chan, Mdb_chan_h

Name :	mdb_chan - Multiple channel DMA block mode A/D input.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>

```
int DRVENTRY mdb_chan(int c_list[], int n_samp, int d_lvl, int burst_len, int *buf);
```

Borland Pascal Usage :

```
function mdb_chan(var c_list; n_samp : integer; dma_lvl : integer; burst : integer; var buf :
integer) : integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION mdb.chan%(SEG chan.list%, BYVAL n.samp%, BYVAL dma.lvl%, BYVAL
burst%, SEG buf%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function mdb_chan(c_l, n_s, d_lvl, b_l, f_s)
integer*2 mdb_chan
integer*2 c_l[REFERENCE]
integer*2 n_s[VALUE]
integer*2 d_lvl[VALUE]
integer*2 b_l[VALUE]
integer*2 f_s[REFERENCE]
end
```

Windows C Usage :

```
int DRVENTRY mdb_chan_h(int __far c_list[], int n_samp, int d_lvl, int burst_len,
HGLOBAL hgBuf);
```

Visual Basic Usage :

```
Declare Function mdb_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp%, ByVal
d_lvl%, ByVal burst_len%, ByVal hgBuf%) As Integer
```

Target : DOS

Description :

This function is similar to M_chan, but uses DMA, and can use block mode. It obtains a sequence of samples from several channels. A list of channels is given by the parameter chan_list, the number of samples by the parameter n_samp, and the parameter f_sample is the integer variable into which the first sample is written.

The channel list is an integer array of maximum length 31, terminated by a channel number greater than 15. Once the last channel is sampled, the first channel is sampled again, until a total of n_samp conversion have been done.

The parameter d_lvl sets the DMA level. This must correspond to the level for which the board is configured.

Parameter burst sets the block length. If you require normal rather than block mode triggering, this should be set to 0. For boards which support it, simultaneous sampling takes effect in burst mode.

Function dma_chk is used to check for DMA completion, and dma_close to align the buffer, check for error and shut the DMA subsystem down.

Mdb_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.

Mdb_chan requires that the buffer be large enough such that room for all of the samples is available in an area of the buffer which does not cross a 64K boundary. The simplest way to

	achieve this is use a buffer which is twice the required size. For example, to acquire 1000 samples, use a buffer of 2000 integer locations (4000 byte locations). Note that data may only be placed in the first part of the buffer by function dma_close.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	mi_chan, mb_chan, sd_chan, dma_chk, dma_close.
Windows Examples :	
DOS Examples :	Demo7.c

6.41. Mde_chan

Name :	mde_chan - Gap-free DMA into a extended memory.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY mde_chan(int c_list[], long n_samp, int d_lvl, int d_lvls, int burst_len);

Borland Pascal Usage :	function mde_chan(var c_list; n_samp : longint; dma_lvl : integer; dma_lvl_s : integer; burst : integer;) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION mde.chan%(SEG chan.list%, BYVAL n.samp&, BYVAL dma.lvl%, BYVAL dma.lvl.s%, BYVAL burst%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function mde_chan(c_l, n_s, d_lvl, d_lvls, b_l) integer*2 mde_chan integer*2 c_l[REFERENCE] integer*4 n_s[VALUE] integer*2 d_lvl[VALUE] integer*2 d_lvls[VALUE] integer*2 b_l[VALUE] end
Visual Basic Usage :	not available
Target :	DOS

Description :	Function mde_chan is similar to mdb_chan, but performs dual DMA channel gap-free DMA into extended memory. The amount of data which can be collected is limited only by extended memory size. The number of samples to be collected is a long(32-bit) integer. The function checks to see that sufficient unused extended memory is available. Note that the same memory as other extended memory functions such as disk caches etc. cannot be used. You can use the e_mem_size function to find out how much extended memory is available. Function dma_chk is used to check for DMA completion, and dma_close to shut the DMA subsystem down. After dma_close completes, the data is still in extended memory. You must use the xfer_dma_res function to transfer the data to normal memory, where it can be accessed. Note that error checking is also done by xfer_dma_res. C_list is the channel list (maximum length 31), n_samp is the number of samples to collect, d_lvl and d_lvls the primary and secondary DMA levels, and burst is the block length. For non-block triggered applications, set burst to 0. This function will use either BIOS allocated memory (INT 15) or XMS memory. The release_e_mem function must be called after each occasion that this function is called.

Remember to call release_e_mem after using this function.

Return value :	ok_30 - operation performed. par_30 - Invalid channel number. mem_30 - Insufficient extended memory. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	sd_chan, mdb_chan, dma_chk, dma_close, xfer_dma_res.
Windows Examples :	not applicable
DOS Examples :	Demo9.c

6.42. Mdh_chan, Mdh_chan_h

Name :	mdh_chan - Gap-free DMA into a huge array.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY mdh_chan(int c_list[], long n_samp, int d_lvl, int d_lvls, int burst_len, int huge *buf);

Borland Pascal Usage :	function mdh_chan(var c_list; n_samp : longint; dma_lvl : integer; dma_lvl_s : integer; burst : integer; var buf : integer) : integer;

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION mdh.chan%(SEG chan.list%, BYVAL n.samp&, BYVAL dma.lvl%, BYVAL
dma.lvl.s%, BYVAL burst%, SEG buf%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function mdh_chan(c_l, n_s, d_lvl, d_lvls, b_l, f_s)
integer*2 mdh_chan
integer*2 c_l[REFERENCE]
integer*4 n_s[VALUE]
integer*2 d_lvl[VALUE]
integer*2 d_lvls[VALUE]
integer*2 b_l[VALUE]
integer*2 f_s[FAR, REFERENCE]
end
```

Windows C Usage :

```
int DRVENTRY mdh_chan_h(int __far c_list[], long n_samp, int d_lvl, int d_lvls, int
burst_len, HGLOBAL hgBuf);
```

Visual Basic Usage :

```
Declare Function mdh_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp&, ByVal
d_lvl%, ByVal d_lvls%, ByVal burst_len%, ByVal hgBuf%) As Integer
```

Target : DOS

Description : Function mdh_chan is similar to mdb_chan, but performs dual DMA channel gap-free DMA. The amount of data which can be collected is limited only by memory size. The number of samples to be collected is a long(32-bit) integer, and the buffer is expected to be a huge buffer.

This function can transfer data to the limits of available normal memory.

Function dma_chk is used to check for DMA completion, and dma_close to align the buffer, check for errors and shut the DMA subsystem down.

C_list is the channel list (maximum length 31), n_samp is the number of samples to collect, d_lvl and d_lvls the primary and secondary DMA levels, and burst is the block length. For non-block triggered applications, set burst to 0. For boards which support it, simultaneous sampling takes effect in burst mode.

Mdh_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.

Note that mdh_chan does not require that the buffer not cross a 64K boundary, or be aligned in any way. For this reason mdh_chan is also useful for transferring into normal sized buffers, without the buffer alignment restriction applying to function mdb_chan.

Return value : ok_30 - operation performed

par_30 - Invalid channel number.

task_30 - Too many tasks accessing the data acquisition hardware.

Prior calls :	Set_base, diag or set_type, set_gain
See Also :	mi_chan, mb_chan, sd_chan, dma_chk, dma_close.
Windows Examples :	
DOS Examples :	Demo8.c

6.43. Mdih_chan, Mdih_chan_h

Name : mdih_chan - Gap-free DMA into a huge array, with background processing of DMA channel swaps.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY mdih_chan(int c_list[], long n_samp, int d_lvl, int d_lvls, int burst_len, int
huge *buf, int i_lvl);
```

Borland Pascal Usage :

```
function mdih_chan(var c_list; n_samp : longint; dma_lvl : integer; dma_lvl_s : integer;
burst : integer; var buf : integer, i_lvl : integer) : integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION mdih.chan%(SEG chan.list%, BYVAL n.samp&, BYVAL dma.lvl%, BYVAL
dma.lvl.s%, BYVAL burst%, SEG buf%, BYVAL i_lvl%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function mdih_chan(c_l, n_s, d_lvl, d_lvls, b_l, f_s, i_l)
integer*2 mdih_chan
integer*2 c_l[REFERENCE]
integer*4 n_s[VALUE]
integer*2 d_lvl[VALUE]
integer*2 d_lvls[VALUE]
integer*2 b_l[VALUE]
integer*2 f_s[FAR, REFERENCE]
integer*2 i_l[VALUE]
end
```

Windows C Usage :

```
int DRVENTRY mdih_chan_h(int __far c_list[], long n_samp, int d_lvl, int d_lvls, int
burst_len, HGLOBAL hgBuf, int i_lvl);
```

Visual Basic Usage :

```
Declare Function mdih_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp&, ByVal
d_lvl%, ByVal d_lvls%, ByVal burst_len%, ByVal hgBuf%, ByVal i_lvl%) As Integer
```

Target : DOS

Description :	Function mdih_chan is similar to mdb_chan, but performs dual DMA channel gap-free DMA, with buffer swaps handled automatically by the driver under interrupt control. Thus there are no requirements that dma_chk needs to be within a certain time period. The amount of data which can be collected is limited only by memory size. The number of samples to be collected is a long(32-bit) integer, and the buffer is expected to be a huge buffer. This function can transfer data to the limits of available normal memory. Values of 2, 3, 5, 7, 10, 11, 12, 14 and 15 are valid for the interrupt level. This level must be the same as set by jumpers on the UEI-30 board, or the configuration values set by a call to config_bd. Function dma_chk is used to check for DMA completion, and dma_close to align the buffer, check for errors and shut the DMA subsystem down. C_list is the channel list (maximum length 31), n_samp is the number of samples to collect, d_lvl and d_lvls the primary and secondary DMA levels, and burst is the block length. For non-block triggered applications, set burst to 0. For boards which support it, simultaneous sampling takes effect in burst mode. Mdih_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle. Note that mdih_chan does not require that the buffer not cross a 64K boundary, or be aligned in any way. For this reason mdih_chan is also useful for transferring into normal sized buffers, without the buffer alignment restriction applying to function mdb_chan.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type, set_gain
See Also :	mi_chan, mb_chan, sd_chan, dma_chk, dma_close.
Windows Examples :	
DOS Examples :	Demo17.c

6.44 Mdih_cont, Mdih_cont_h

Name :	mdih_cont - Continuous Gapfree DMAinto a huge array, with background processing of DMA channel swaps. - uses Smart Cache operations.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>

	int DRVENTRY mdih_cont(int c_list[], long lSectorSize, int iNumSectors, int burst_len, int iClockMode, int iTrigMode, short HUGEPOINTER *phaBuf);
Borland PASCAL Usage :	function mdih_cont(var c_list[]; lSectorSize : longint; iNumSectors, burst_len,
	iClockMode, iTrigMode : integer; var *phaBuf : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC'
	FUNCTION mdih.cont%(SEG chan.list%, BYVAL sec.siz&, BYVAL num.sec%, BYVAL burst%, SEG buf%, BYVAL clk.mode%, BYVAL trg.mode%, SEG pha.buf%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi'
	INCLUDE 'PC30.fd'
	interface to function mdih_cont(c_l, s_s, n_s, b_l, c_m, t_m, p_b)
	integer*2 mdih_cont
	integer*2 c_l[REFERENCE]
	integer*4 s_s[VALUE]
	integer*2 n_s[VALUE]
	integer*2 b_l[VALUE]
	integer*2 c_m[VALUE]
	integer*2 t_m[VALUE]
	integer*2 p_b[FAR, REFERENCE]
	end
Windows C Usage :	int DRVENTRY mdih_cont_h(int __far c_list[], long lSectorSize, int iNumSectors, int burst_len, int iClockMode, int iTrigMode, HGLOBAL PhaBuf);
Visual Basic Usage :	Declare Function mdih_cont_h Lib "ueidaq.dll" (chan_list%, ByVal sec_siz&, ByVal num_sec%, ByVal burst_len%, clk_mode%, ByVal trg_mode%, BYVAL hg_phBuf%) As Integer
Target :	DOS, Windows 3.1
Description :	Function mdih_cont is similar to mdih_chan, but acquires continuously, and samples by size and number of sectors, using dual channel gapfree DMA, or Rep String operations. The amount of data which can be collected is limited only by memory size. The buffer is expected to be a huge buffer. This function can transfer data to the limits of available normal memory. C_list is the channel list , SectorSize and NumSectors are the size and number of sectors as

defined by the user. Burst_Len sets the block length. For nonblock triggered applications, set burst to 0. For the WIN-30DS and WIN-30DS/4, simultaneous sampling takes effect in burst mode. iTrigMode is reserved for future use and should be set to 0. iClockMode sets external clocking. Set to 1 for internal clock. Mdih_cont_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle. Note that mdih_cont does not require that the buffer not cross a 64K boundary, or be aligned in any way. For this reason mdih_cont is also useful for transferring into normal sized buffers, without the buffer alignment restriction applied to function mdb_chan.

Return value :	ok_30	Operation performed
	mem_30	Too little memory to perform the requested operation.
	task_30	Too many tasks accessing the data acquisition hardware.

Prior calls : Set_base, diag or set_type, set_gain

See Also : mi_chan, mb_chan, sd_chan, ad_cont_close

Windows Examples : WDemo1.c

DOS Examples : Demo20.c

6.45. Mi_chan

Name : mi_chan - Multiple channel interrupt controlled input.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY mi_chan(int chan_list[], int n_samp, int *f_sample, int i_lvl);
```

Borland Pascal Usage : Not available.

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION mi.chan%(SEG chan.list%, BYVAL n.samp%, SEG f.sample%, BYVAL i.lvl%)
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function mi_chan (c_l, n_s, f_s, i_l)
integer*2 mi_chan
integer*2 c_l [REFERENCE]
integer*2 n_s [VALUE]
integer*2 f_s [REFERENCE]
integer*2 i_l [VALUE]
end
```

Visual Basic Usage :

```
Declare Function mi_chan Lib "ueidaq.dll" (chan_list%, ByVal n_samp%, f_sample%, ByVal i_lvl%) As Integer
```

Target : DOS, Windows 3.1

Description : This function is very similar to m_chan, but operates in interrupt mode. This is useful at low sampling rates, where the host PC can continue to perform other tasks while the data acquisition is done via the interrupt system. Function int_chk is called to determine whether the operation has completed. Note that calling mi_chan only starts the operation. No actual data transfer takes place until after this routine completes. Note that procedure int_close must be called to clean up the interrupt vectors etc., prior to program termination.

Values of 2, 3, 5, 7, 10, 11, 12, 14 and 15 are valid for the interrupt level. This level must be the same as set by jumpers or via software on the selected board.

This procedure does not make use of the channel list system, or burst mode operation.

Note that when the UEI-127 is used in conjunction with this function, only a single sample be clock cycle is taken, The other three samples taken by the UEI-127 are discarded.

Return value : ok_30 - operation started.

par_30 - Invalid channel number.

task_30 - Too many tasks accessing the data acquisition hardware.

Prior calls : Set_base, diag or set_type, set_gain

See Also : Int_chk, int_close, mb_chan.

Windows Examples :

DOS Examples : Demo5.c

6.46. Read_clock

Name : read_clock - Not used.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

int DRVENTRY read_clock();
```

Borland Pascal Usage :

```
function read_clock : integer;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

FUNCTION read.clock%
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to function read_clock
integer*2 read_clock
```

end

Visual Basic Usage : Declare Function read_clock Lib "ueidaq.dll" () As Integer

Target : DOS, Windows 3.1

Description : This procedure is a dummy procedure, supplied for compatibility with other driver packages.

Return value : None.

Prior calls : Set_base

See Also :

6.47. Release_e_mem

Name : release_e_mem **-** Releases XMS memory obtained by the mde_chan function.

Boards Supported : All WIN boards

C Usage :

```
#include <PC30.H>

void DRVENTRY release_e_mem();
```

Borland Pascal Usage :

```
procedure release_e_mem;
```

DOS QuickBasic Usage :

```
REM $INCLUDE: 'PC30.INC'

SUB release.e.mem
```

DOS FORTRAN Usage :

```
INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

interface to subroutine release_e_mem
end
```

Visual Basic Usage : not available

Target : DOS

Description : This function releases any XMS memory obtained by the mde_chan function. The function should be called after all required data has been transferred to normal memory by the xfer_dma_res function. If each call to mde_chan is not exactly matched by a corresponding call to this function, memory will be "lost" until the PC is rebooted.

Return value : None.

Prior calls : Mde_chan.

See Also : Mde_chan.

Windows Examples : not applicable

DOS Examples :	Demo9.c

6.48. Rtc_off

Name :	rtc_off - disables the PCs clock
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY rtc_off();

Borland Pascal Usage :	procedure rtc_off;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB rtc.off%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine rtc_off end
Visual Basic Usage :	Declare Sub rtc_off Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	Disables the host PC's real-time clock interrupt. This interrupt locks out program transfer and other interrupt routines for up to 200 uS. This can result in lost data and over-run errors at A/D sampling rates higher than 1 KHz when using polled I/O or interrupts to read A/D data. By disabling the real-time clock, higher sampling rates can be achieved.
Return value :	None
Prior calls :	None.
See Also :	rtc_on.
Windows Examples :	
DOS Examples :	Demo3.pas

6.49. Rtc_on

Name :	rtc_on - enables the PC's clock
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>

	void DRVENTRY rtc_on();
Borland Pascal Usage :	procedure rtc_on;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB rtc.on%
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine rtc_on end
Visual Basic Usage :	Declare Sub rtc_on Lib "ueidaq.dll" ()
Target :	DOS, Windows 3.1
Description :	Re-enables the host PC's real-time clock interrupt. This function is used to switch the real-time clock on again after it has been switched off by rtc_off. It is good practice to call this function as soon as possible after a data acquisition function terminates.
Return value :	None
Prior calls :	None.
See Also :	rtc_off.
Windows Examples :	
DOS Examples :	Demo3.pas

6.50. S_chan, S_chan_h

Name :	s_chan - Obtains a set of samples from a single channel.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> int DRVENTRY s_chan(int chan, int n_samp, int *f_sample);
Borland Pascal Usage :	function s_chan(chan, n_samp : integer; var f_sample : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION s.chan%(BYVAL chan%, BYVAL n.samp%, SEG f.sample%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function s_chan(chan, n_samp, f_sample)

```
integer*2 s_chan
integer*2 chan [VALUE]
integer*2 n_samp [VALUE]
integer*2 f_sample [REFERENCE]
end
```

Windows C Usage : int DRVENTRY s_chan_h(int chan, int n_samp, HGLOBAL hgBuf);

Visual Basic Usage : Declare Function s_chan_h Lib "ueidaq.dll" (chan%, ByVal n_samp%, ByVal hgBuf%) As Integer

Target : DOS, Windows 3.1

Description : This function obtains a sequence of samples from a single channel. The channel is given by parameter chan, the number of samples by the parameter n_samp, and the parameter f_sample is the integer variable into which the first sample is written. For example, if f_sample was the third location in an array, the second sample would go into location 4, the third into location 5 etc.

S_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.

Return value : ok_30 - operation performed

par_30 - Invalid channel number.

err_30 - A/D hardware error.

task_30 - Too many tasks accessing the data acquisition hardware.

Prior calls : Set_base, diag or set_type, set_gain

See Also : sd_chan.

6.51. Sd_chan, Sd_chan_h

Name : sd_chan - Obtains a set of samples from a single channel under DMA control.

Boards Supported : All WIN boards

C Usage : #include <PC30.H>

int DRVENTRY sd_chan(int chan, int n_samp, int d_lvl, int *buf);

Borland Pascal Usage : function sd_chan(chan, n_samp : integer; d_lvl : integer; var buf : integer) : integer;

DOS QuickBasic Usage : REM $INCLUDE: 'PC30.INC'

FUNCTION sd.chan%(BYVAL chan%, BYVAL n.samp%, BYVAL d.lvl%, SEG buf%)

DOS FORTRAN Usage : INCLUDE 'PC30.fi'
INCLUDE 'PC30.fd'

```
interface to function sd_chan(chan, n_s, d_lvl, buf)
integer*2 sd_chan
integer*2 chan [VALUE]
integer*2 n_s [VALUE]
integer*2 d_lvl [VALUE]
integer*2 buf [REFERENCE]
end
```

Windows C Usage :	int DRVENTRY sd_chan_h(int chan, int n_samp, int d_lvl, HGLOBAL hgBuf);
Visual Basic Usage :	Declare Function sd_chan_h Lib "ueidaq.dll" (chan%, ByVal n_samp%, ByVal d_lvl%, ByVal hgBuf%) As Integer
Target :	DOS

Description :

This function obtains a sequence of samples from a single channel. It is very similar to s_chan, with the exception that the user must specify a DMA level, and that it executes in the background. Function dma_chk is used to check for completion, and function dma_close is used to shut down the DMA system.

Valid setting for d_lvl are 1, 3, 5, 6 or 7. This value must correspond to the setting on the UEI-30 board.

Sd_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.

Sd_chan requires that the buffer be large enough such that room for all of the samples is available in an area of the buffer which does not cross a 64K boundary. The simplest way to achieve this is use a buffer which is twice the required size. For example, to acquire 1000 samples, use a buffer of 2000 integer locations (4000 byte locations). Note that data may only be placed in the first part of the buffer by function dma_close.

Note that function dma_init must be called prior to sd_chan.

Return value :

ok_30 **-** operation started.

par_30 **-** Invalid channel number or DMA level.

task_30 **-** Too many tasks accessing the data acquisition hardware.

Prior calls :	Set_base, diag or set_type, set_gain
See Also :	s_chan, dma_chk, dma_close.
Windows Examples :	
DOS Examples :	Demo6.c

6.52. Select_board

Name :	select_board **-** Selects the logical board number that all subsequent driver calls apply to.

Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY select_board(int b_num);

Borland Pascal Usage :	function select_board(b_num : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION select.board%(BYVAL bnum%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function select_board (b_num) integer*2 select_board integer*2 b_num [VALUE] end
Visual Basic Usage :	Declare Function select_board% Lib "ueidaq.dll" (ByVal num%)
Target :	DOS, Windows 3.1
Description :	This function selects the logical board number of the board that all subsequent driver calls apply to. A different board is selected only when another call to select_board is performed. The driver internally stores information on the base address, the type and the A/D gain values of each board. When DMA or interrupt operations are active, the driver also stores complete internal state information, allowing simultaneous operations to be performed on several boards.

Note that it is the user's responsibility to ensure that no two boards ever try to use the same DMA or interrupt channels. Failure to ensure this can result in damage to the boards in question.

Return value :	ok_30 - operation started. par_30 - Invalid board number; must be between 0 and 7. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	None
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo15.c

6.53. Set_base

Name :	set_base - Sets the base address of the board that the driver software is to control. For Windows NT this must be done in the registry. Refer to the WIN-10/30 hardware manual.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY set_base(int base);

Borland Pascal Usage :	procedure set_base(base : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB set.base(BYVAL base%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine set_base (b_addr) integer*2 b_addr [VALUE] end
Visual Basic Usage :	
Target :	DOS, Windows 3.1
Description :	This function sets the base address of the currently selected board.
Return value :	None.
Prior calls :	Declare Sub set_base Lib "ueidaq.dll" (ByVal base_addr%)
See Also :	Get_base
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo11.bas, demo12.bas

6.54. Set_gain

Name :	set_gain - Sets the gain of the selected channel.
Boards Supported :	All WIN-PG boards
C Usage :	#include <PC30.H> void DRVENTRY set_gain(int index, int value);
Borland Pascal Usage :	procedure set_gain(index, value : integer);

DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB set.gain(BYVAL index%, BYVAL value%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to subroutine set_gain (index, val) integer*2 index [VALUE] integer*2 val [VALUE] end
Visual Basic Usage :	Declare Sub set_gain Lib "ueidaq.dll" (ByVal index%, ByVal g_val%)
Target :	DOS, Windows 3.1
Description :	This function sets the channel gain global variable, gain_30. Index refers to the channel, and value is the gain code corresponding to the selected channel. Gain codes are discussed in chapter 3. Note that the set_gain function does not set the actual board gain, but only sets the global variable. The data acquisition functions load the gain codes from the global variable into the selected board when they are called. Note that this means that you can't use this function to change channel gains while a background (DMA or interrupt) data acquisition function is running. Any changes you make will only come into effect when the data acquisition function is called initially.
Return value :	None.
Prior calls :	set_gain
See Also :	Get_gain
Windows Examples :	
DOS Examples :	Demo2.pas, demo3.pas, demo5.c, demo6.c, demo7.c, demo8.c, demo9.c, demo12.bas, demo14.for

6.55. Set_osc

Name :	Set_osc - Sets the master oscillator value.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY set_osc(unsigned freq);

Borland Pascal Usage :	function set_osc(freq : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC'

	FUNCTION set.osc(BYVAL freq%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function set_osc(freq) integer*2 set_osc integer*2 freq end
Visual Basic Usage :	Declare Function set_osc Lib "ueidaq.dll" () As Integer
Target :	DOS, Windows 3.1
Description :	This function sets the frequency of the master oscillator for WIN-30 series boards. The master oscillator frequency is set in hundreds of KHz. Possible values are 100 (10 MHz) and 20 (2 MHz)
Return value :	ok_30 **-** operation performed. par_30 **-** invalid frequency setting. task_30 **-** Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag
See Also :	Get_osc
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	Demo2.pas, demo3.pas, demo4.c, demo5.c, demo6.c

6.56. Set_type

Name :	set_type **-** Sets the board type that the driver software is to control. This function is only required if for some reason the Diag function is not called.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	void DRVENTRY set_type(int type);

Borland Pascal Usage :	procedure set_type(type : integer);
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' SUB set.type(BYVAL type%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd'

	interface to subroutine set_type (type) integer*2 type [VALUE] end
Visual Basic Usage :	Declare Sub set_type Lib "ueidaq.dll" (ByVal bd_type%)
Target :	DOS, Windows 3.1
Description :	This function sets the board type. This function is only called if for some reason the Diag function is not called, or it is necessary to override the Diag function's board identification..
Return value :	None.
Prior calls :	Set_base
See Also :	
Windows Examples :	
DOS Examples :	

6.57. Wfm_chan, Wfm_chan_h

Name :	wfm_chan **-** Waveform generation via the D/A outputs.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY wfm_chan(int chan_list[], int n_samp, int n_cycle, int *f_sample);

Borland Pascal Usage :	function wfm_chan(var chan_list; n_samp : integer; n_cycle : integer; var f_sample : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION wfm.chan% ALIAS "wfm_chan" (SEG chan.list%, BYVAL n.samp%, BYVAL n.cycle%, SEG f.sample%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function wfm_chan (c_l, n_s, n_c, f_sample) integer*2 wfm_chan integer*2 c_l [REFERENCE] integer*2 n_s [VALUE] integer*2 n_c [VALUE] integer*2 f_sample [REFERENCE] end
Windows C Usage :	int DRVENTRY wfm_chan_h(int __far c_list[], int n_samp, int n_cycle, HGLOBAL hgBuf);

Visual Basic Usage :	Declare Function wfm_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp%, ByVal n_cycle%, ByVal hgBuf%) As Integer
Target :	DOS, Windows 3.1
Description :	This function takes a series of samples from the buffer, and outputs them to selected D/A converters. This will occur at the rate set by the prescaler and D/A divider, or the external clock. A list of channels is given by the parameter chan_list, and the number of samples by the parameter n_samp. The number of times that the function is to cycle through the buffer is given by the n_cycle parameter. The parameter f_sample is the integer variable from which the first sample is written. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Wfm_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	D/A clock.
Windows Examples :	
DOS Examples :	Demo16.c

6.58. Wfmih_chan, Wfmih_chan_h

Name :	wfmih_chan - Interrupt driven waveform generation via the D/A outputs.
Boards Supported :	All WIN boards

C Usage :	#include <PC30.H>
	
	int DRVENTRY wfmih_chan(int DRVBUF c_list[], long n_samp, long n_cycle, int __huge *buf, int i_lvl);

Borland Pascal Usage :	function wfmih_chan(var chan_list; n_samp : longint; n_cycle : longint; var f_sample : integer; i_lvl : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION wfmih.chan% ALIAS "wfmih_chan" (SEG chan.list%, BYVAL n.samp&, BYVAL n.cycle&, SEG f.sample%, BYVAL i.lvl%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd'

```
interface to function wfmih_chan (c_l, n_s, n_c, f_sample, i_l)
integer*2 wfmih_chan
integer*2 c_l [REFERENCE]
integer*4 n_s [VALUE]
integer*4 n_c [VALUE]
integer*2 f_sample [REFERENCE]
integer*2 i_l [VALUE]
end
```

Windows C Usage :	int DRVENTRY wfmih_chan_h(int __far c_list[], long n_samp, long n_cycle, HGLOBAL hgBuf, int i_lvl);
Visual Basic Usage :	Declare Function wfmih_chan_h Lib "ueidaq.dll" (chan_list%, ByVal n_samp&, ByVal n_cycle&, ByVal hgBuf%, BYVAL i_lvl) As Integer
Target :	DOS, Windows 3.1
Description :	This function takes a series of samples from the buffer, and outputs them to selected D/A converters. Data transfer occurs via interrupt operation in the background. This will occur at the rate set by the prescaler and D/A divider, or the external clock. A list of channels is given by the parameter chan_list, and the number of samples by the parameter n_samp. The number of times that the function is to cycle through the buffer is given by the n_cycle parameter. If the n_cycle value is set to 0, then the function continues until it is terminated by a call to Int_close. The parameter f_sample is the integer variable from which the first sample is written. The channel list is an integer array of any length, which must be terminated by a channel number greater than 15. Values of 2, 3, 5, 7, 10, 11, 12, 14 and 15 are valid for the interrupt level, i_lvl. This level must be the same as set by jumpers on the UEI-30 board, or the configuration values set by a call to config_bd in the case of a WIN-30. Wfmih_chan_h is a Windows specific version of the function; rather than a buffer address, it takes a Windows Global Handle. Function completion can be checked for via the Int_chk call. The operation must be terminated by the Int_close call. Note that is this function call is set for continuous operation (n_cycle = 0), and the update rate is too high for the host PC, the interrupt handler can absorb all CPU time, so effectively crashing the system.
Return value :	ok_30 - operation performed par_30 - Invalid channel number. task_30 - Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	D/A clock.
Windows Examples :	Wdemo1
DOS Examples :	Demo16.c

6.59. Xfer_dma_res

Name :	Xfer_dma_res **-** Transfers data from extended memory to normal memory, and checks for errors.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> int DRVENTRY xfer_dma_res(int *buf, long start, int len);
Borland Pascal Usage :	function xfer_dma_res(var buf; start : longint; length : integer) : integer;
DOS QuickBasic Usage :	REM $INCLUDE: 'PC30.INC' FUNCTION xfer.dma.res%(SEG buf%, BYVAL start&, BYVAL length%)
DOS FORTRAN Usage :	INCLUDE 'PC30.fi' INCLUDE 'PC30.fd' interface to function xfer_dma_res(buf, start, len) integer*2 xfer_dma_res integer*2 buf[REFERENCE] integer*4 start[VALUE] integer*2 len[VALUE] end
Visual Basic Usage :	not available
Target :	DOS
Description :	Data, starting at sample start, is transferred from extended memory to the array buf. The number of samples transferred is indicated by len. Note that each section transferred is checked individually for errors.
Return value :	ok_30 **-** operation performed err_30 **-** A/D hardware error. mem_30 **-** memory not available or memory error. task_30 **-** Too many tasks accessing the data acquisition hardware.
Prior calls :	Set_base, diag or set_type
See Also :	mde_chan.
DOS Examples :	demo9.c

6.60. Vb_InitUEI_mem

Name :	vb_InitUEI_mem **-** Initializes the memory helper system
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	void DRVENTRY vb_InitUEI_mem(void);
Borland Pascal Usage :	procedure vb_InitUEI_mem;
Visual Basic Usage :	Declare Sub vb_InitUEI_mem Lib "ueidaq.dll" ()
Target :	Windows 3.1
Description :	Initializes the Visual Basic memory helper system. Must be called prior to using any other memory helper function.
Return value :	None
Prior calls :	None
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.61. Vb_allocateUEI_short

Name :	vb_allocateUEI_short **-** Memory helper function; allocates an array of Integer values
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	HGLOBAL DRVENTRY vb_allocateUEI_short(unsigned int uiChannels, unsigned long ulSamples);
Borland Pascal Usage :	function vb_allocateUEI_short(uiChan : integer; ulSamples : longint) : word;
Visual Basic Usage :	Declare Function vb_allocateUEI_short Lib "ueidaq.dll" (ByVal uiChannels%, ByVal ulSamples&) As Integer
Target :	Windows 3.1
Description :	Allocates an array of Integer values. The number of values in the array is the product of uiChannels% and ulSamples&.

Any memory allocated by this function must be freed by the vb_freeUEI function.

Return value :	The integer returned by this function is the handle of the array. This must be used in calls that access the array. If the handle returned is 0, then not enough memory was available to satisfy the request.
Prior calls :	vb_InitUEI_mem
See Also :	vb_freeUEI_mem
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.62. Vb_allocateUEI_float

Name :	vb_allocateUEI_float - Memory helper function; allocates an array of Single values
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> HGLOBAL DRVENTRY vb_allocateUEI_float(unsigned int uiChannels, unsigned long ulSamples);
Borland Pascal Usage :	function vb_allocateUEI_float(uiChan : integer; ulSamples : longint) : word;
Visual Basic Usage :	Declare Function vb_allocateUEI_float Lib "ueidaq.dll" (ByVal uiChannels%, ByVal ulSamples&) As Integer
Target :	Windows 3.1
Description :	Allocates an array of Single values. The number of values in the array is the product of uiChannels% and ulSamples&.

Any memory allocated by this function must be freed by the vb_freeUEI function.

Return value :	The integer returned by this function is the handle of the array. This must be used in calls that access the array. If the handle returned is 0, then not enough memory was available to satisfy the request.
Prior calls :	vb_InitUEI_mem
See Also :	vb_freeUEI_mem
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.63. Vb_reallocateUEI_short

Name :	vb_reallocateUEI_short - Memory helper function; reallocates an array of Integer values.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> HGLOBAL DRVENTRY vb_reallocateUEI_short(HGLOBAL hgDat, unsigned int uiChannels, unsigned long ulSamples);
Borland Pascal Usage :	function vb_reallocateUEI_short(hgDat : word; uiChan : integer; ulSamples : longint) : word;
Visual Basic Usage :	Declare Function vb_reallocateUEI_short Lib "ueidaq.dll" (ByVal hgDat%, ByVal uiChannels%, ByVal ulSamples&) As Integer
Target :	Windows 3.1
Description :	Re-dimensions an array of Integer values. The number of values in the re-dimensioned array is the product of uiChannels% and ulSamples&. If the number of channels in this call is the same as in the original allocate call, then the existing data is preserved.

Any memory allocated by this function must be freed by the vb_freeUEI function.

Return value :	The integer returned by this function is the handle of the array. This must be used in calls that access the array. If the handle returned is 0, then not enough memory was available to satisfy the request.
Prior calls :	vb_InitUEI_mem
See Also :	vb_freeUEI_mem
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.64. Vb_reallocateUEI_float

Name :	vb_reallocateUEI_float - Memory helper function; reallocates an array of Single values.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> HGLOBAL DRVENTRY vb_reallocateUEI_float(HGLOBAL hgDat, unsigned int uiChannels, unsigned long ulSamples);
Borland Pascal Usage :	function vb_reallocateUEI_float(hgDat : word; uiChan : integer; ulSamples : longint) : word;
Visual Basic Usage :	Declare Function vb_reallocateUEI_float Lib "ueidaq.dll" (ByVal hgDat%, ByVal

	uiChannels%, ByVal ulSamples&) As Integer
Target :	Windows 3.1
Description :	Redimentions an array of Single values. The number of values in the re-dimensioned array is the product of uiChannels% and ulSamples&. If the number of channels in this call is the same as in the original allocate call, then the existing data is preserved.

Any memory allocated by this function must be freed by the vb_freeUEI function.

Return value :	The integer returned by this function is the handle of the array. This must be used in calls that access the array. If the handle returned is 0, then not enough memory was available to satisfy the request.
Prior calls :	vb_InitUEI_mem
See Also :	vb_freeUEI_mem
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.65. Vb_freeUEI

Name :	vb_freeUEI - Memory helper function; frees an array of memory.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	void DRVENTRY vb_freeUEI(HGLOBAL hgDat);
Borland Pascal Usage :	procedure vb_freeUEI(hgDat : word);
Visual Basic Usage :	Declare Function vb_freeUEI Lib "ueidaq.dll" (ByVal hgDat%)
Target :	Windows 3.1
Description :	Frees the memory allocated by either the vb_allocateUEI_short or vb_allocateUEI_float functions.
Return value :	None
Prior calls :	vb_InitUEI_mem
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.66. Vb_readUEI_short

Name :	vb_readUEI_short - memory helper function; reads an Integer value from a previously allocated array.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	short DRVENTRY vb_readUEI_short(HGLOBAL hgDat, unsigned int uiChan, unsigned long ulSample);
Borland Pascal Usage :	function vb_readUEI_float(hgDat : word; uiChan : integer; ulSample : longint) : single;
Visual Basic Usage :	Declare Function vb_readUEI_short Lib "ueidaq.dll" (ByVal hgDat%, ByVal uiChan%, ByVal ulSample&) As Integer
Target :	Windows 3.1
Description :	Reads the memory location identified by uiChan% and ulSample&. Array subscripts start at 0. The maximum values that uiChan% and ulSample& can take on is one less than the corresponding values in the allocate call. hgDat% identifies the array.
Return value :	The Integer value.
Prior calls :	vb_InitUEI_mem
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.67. Vb_readUEI_float

Name :	vb_readUEI_float - memory helper function; reads an Single value from a previously allocated array.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	float DRVENTRY vb_readUEI_float(HGLOBAL hgDat, unsigned int uiChan, unsigned long ulSample);
Borland Pascal Usage :	function vb_readUEI_short(hgDat : word; uiChan : integer; ulSample : longint) : integer;
Visual Basic Usage :	Declare Function vb_readUEI_float Lib "ueidaq.dll" (ByVal hgDat%, ByVal uiChan%, ByVal ulSample&) As float
Target :	Windows 3.1

Description :	Reads the memory location identified by uiChan% and ulSample&. Array subscripts start at 0. The maximum values that uiChan% and ulSample& can take on is one less than the corresponding values in the allocate call. hgDat% identifies the array.
Return value :	The Single value.
Prior calls :	vb_InitUEI_mem
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.68. Vb_writeUEI_short

Name :	vb_writeUEI_short **-** memory helper function; writes an Integer value to a previously allocated array.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H> void DRVENTRY vb_writeUEI_short(HGLOBAL hgDat, unsigned int uiChan, unsigned long ulSample, short iVal);
Borland Pascal Usage :	procedure vb_writeUEI_short(hgDat : word; uiChan : integer; ulSample : longint; iVal : integer);
Visual Basic Usage :	Declare Sub vb_writeUEI_short Lib "ueidaq.dll" (ByVal hgDat%, ByVal uiChan%, ByVal ulSample&, ByVal iVal%)
Target :	Windows 3.1
Description :	Writes the memory location identified by uiChan% and ulSample&. Array subscripts start at 0. The maximum values that uiChan% and ulSample& can take on is one less than the corresponding values in the allocate call. hgDat% identifies the array. IVal is the value written.
Return value :	None
Prior calls :	vb_InitUEI_mem
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.69. Vb_writeUEI_float

Name :	vb_writeUEI_float - memory helper function; writes an Single value to a previously allocated array.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	void DRVENTRY vb_writeUEI_float(HGLOBAL hgDat, unsigned int uiChan, unsigned long ulSample, float fVal);
Borland Pascal Usage :	
Visual Basic Usage :	Declare Sub vb_writeUEI_float Lib "ueidaq.dll" (ByVal hgDat%, ByVal uiChan%, ByVal ulSample&, ByVal fVal!)
Target :	Windows 3.1
Description :	Writes the memory location identified by uiChan% and ulSample&. Array subscripts start at 0. The maximum values that uiChan% and ulSample& can take on is one less than the corresponding values in the allocate call. hgDat% identifies the array. fVal is the value written.
Return value :	None
Prior calls :	vb_InitUEI_mem
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

6.70. Vxd_version

Name :	vxd_version - returns the Windows virtual device driver(Vxd) version number.
Boards Supported :	All WIN boards
C Usage :	#include <PC30.H>
	int DRVENTRY vxd_version();
Borland Pascal Usage :	function vxd_version : integer
Visual Basic Usage :	Declare Function vxd_version Lib "ueidaq.dll" () As Integer
Target :	Windows 3.1
Description :	Returns the version number of the virtual device driver, encoded as follows :

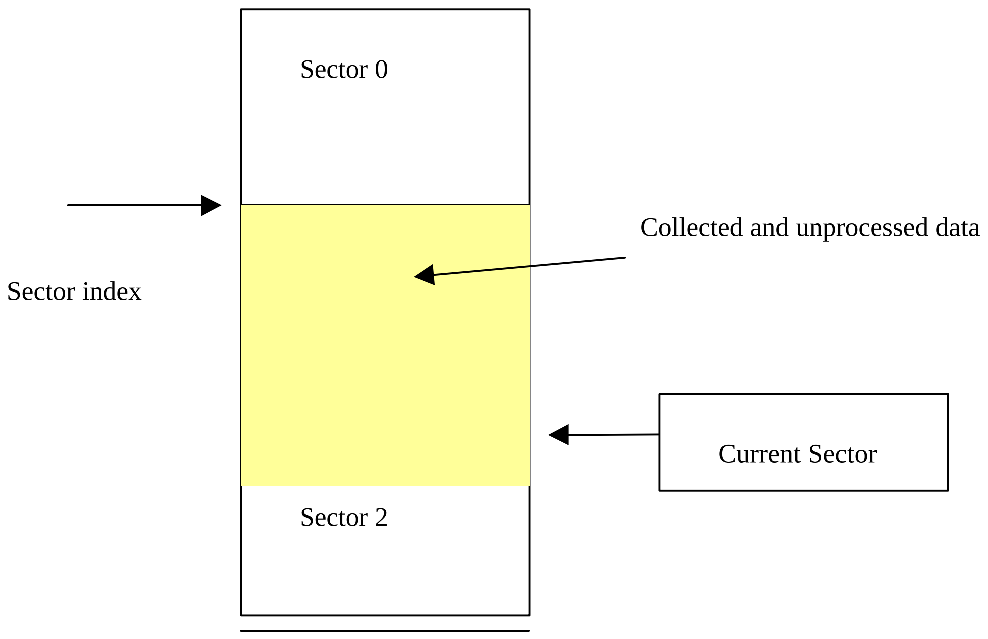
	High byte : Major version number.
	Low byte : Minor version number.
	For example, version 2.06 would return 0206 hex.
	If the driver is not installed, 0000H will be returned. Thus this function can be used as a check on the presence of the Vxd.
Return value :	Integer representing the version number.
Prior calls :	None.
See Also :	
Windows Examples :	Wdemo1.c, wdemo2.pas, wdemo3.bas
DOS Examples :	

Appendix A

A.1. How to program continuous AIN for WIN-30 boards

As for any other mode of operation, the user must first configure the board and AIN parameters such as conversion frequency and channel list.

For continuous AIN the user has to allocate a **short** memory buffer which will hold several sectors of equal size. The driver fills these sectors with data samples and reuses them after they are released by the application program.



Continuous data collection starts by calling the function **mdih_cont** and passing to it the number of sectors, sector length, and pointer to the buffer containing the sectors.

Continuous data collection starts by calling the function **mdih_cont** and passing to it the number of sectors, sector length, and pointer to the buffer containing the sectors.

Once the continuous operation started, the user has to check its status periodically. This is done by calling the function **query_current_sector**. It reports which sector is being written to. When the current sector number changes, the previous sector can be processed, for example, its content may be copied to another buffer or saved to a file. If everything goes normally, **query_current_sector** returns **n_comp_30**, return value **ok_30** in this case means buffer overrun. This problem may be solved by increasing the polling frequency or by increasing the number of sectors and their size.

After a sector is processed it must be released. this is done by calling the function **mark_sector_ready** which tells the driver that the sector is available for reuse.

To finish continuous analog input, call the function **ad_cont_close**.

In following code fragment 20 data sectors are collected contiunously.

```
long   lSectorSize = 2048;       // number of data points in one sector
int    iNumSectors = 5;          // number of sectors
int    * pChanList;              // channel list array
int     iSectorsCnt = 0;         // number of processed sectors
int     iSectorIndex = 0;        // index to the next sector to process
int    iTotalSectors = 20;       // total number of sectors to process

// allocate data buffer
short  * pBuffer = (short *) malloc (lSectorSize * iNumSectors * sizeof (short));

// initialize channel list
. . .


// Start continuous data acquisition

mdih_cont (pChanList, lSectorSize, iNumSectors,

        iBurst,  // burst length

        1,       // clock mode

        0,       // trigger mode

        pBuffer);




static int   iCurSector = 0;     // index to sector which is currently filled by  //
                                 the driver
static long  lSectorPos;


// timer procedure
. . .

int          iQueryRet;          // return value
short        *pSector;           // pointer to data sector

// Check to see what sector is being written to; when this changes, the previous //
sector can be processed

iQueryRet = query_current_sector(&iCurSector, &lSectorPos);

do {
      if (iCurSector != iSectorIndex)
      {
            pSector = pBuffer + iSectorIndex * lSectorSize;
            // The code for whatever processing is required goes here…
            . . .
            // Tell the acquisition engine that we are finished with this sector
```

```
                mark_sector_ready (iSectorIndex++);

                // And wrap round to the start of the buffer
                if (iSectorIndex >= iNumSectors) iSectorIndex = 0;
                    iSectorsCnt++;
        }
        // and query again
        iQuerySto = query_current_sector(&iCurSector, &lSectorPos);
}
while ((iQuerySto == n_comp_30) && (iSectorsCnt < iTotalSectors) &&
        (iCurSector != iSectorIndex));



ad_cont_close();
```

Appendix B

B.1. Reading Wdemo1 Packed Data Files

The sample program WDEMO1 is capable of saving data to disk in unpacked binary form. This means that each sample occupies the low twelve bits of a word; the high four bits of each word are unused. You may have a data viewer or editor capable of displaying blocks of 16-bit values as integers. If not, it is simple to write a program that will convert the file to ASCII. Something like the following should work: (this is an example only)

```
#include <io.h>
main()
{
  int f;
  f = open("datafile", O_BINARY | O_RDONLY)
  while (!eof(f)) {
    read(f, &i, 2); // load a single 2-byte sample into i
    printf("%d\n", i & 0xfff);  // print it on the screen
                         // chopping off the high four bits just
                         // to be sure
  }
  close(f);
}
```

If you have turned data packing on, by commenting out the indicated lines in the source code of WDEMO1, things are a bit messier. I refer you to the diagram in the WIN-30 manual for a description of the packing method used. You have to read three words at a time and unpack the high four bits of each word into a fourth twelve-bit sample. But that's only if you've altered the program to turn data packing on.

B.2. Reading Status for Windows Data Files

The following structure outlines what is required to read the .DAT file. This program uses Visual BASIC for Windows.

Setup variables:

Dim dat_loop As Long	(4 bytes)
Dim dat_loop2 As Long	(4 bytes)
Dim data_val	
conf_atod.hgR1 = vb_allocateUEI_float(1, 1)	
conf_atod.hgR2 = 0	

Read the file header:

dat_atod.FileId	[Char 32]
AtoD_Dat.slotnum	[Long 4 bytes]

```
        AtoD_Dat.CardModel      [Char 16]
        AtoD_Dat.CardName       [Char 24]
        AtoD_Dat.Date           [Char 8]
        AtoD_Dat.Time           [Char 8]
        AtoD_Dat.TriggerOffset  [Long 4 bytes]
        AtoD_Dat.Thruput        [Single 4 bytes]
        AtoD_Dat.NumSamples     [Long 4 bytes]
        AtoD_Dat.NumSets        [Long 4 bytes]
        AtoD_Dat.BackGroundColor        [Long 4 bytes]

For dat_loop = 0 To AtoD_Dat.NumSets - 1
        ChannelNum              [Long 4 bytes]
        Gain                    [Long 4 bytes]
        ScaleFactor             [Single 4 bytes]
        Units                   [String 4 bytes]
        CaptureEnabled          [Long 4 bytes]
        DisplayEnabled          [Long 4 bytes]
        DisplayColor            [Long 4 bytes]
Next dat_loop

conf_atod.hgR1 = vb_allocateUEI_float(1, 1)
conf_atod.hgR2 = vb_reallocateUEI_float(conf_atod.hgR1, AtoD_Dat.NumSets, AtoD_Dat.NumSamples)
```

<u>Read the data:</u>

```
For dat_loop = 0 To AtoD_Dat.NumSets - 1
        For dat_loop2 = 0 To AtoD_Dat.NumSamples - 1
                Read ad_value   [Single 4 bytes]
                vb_writeUEI_float conf_atod.hgR2, dat_loop, dat_loop2, dat_atod.Data
        Next dat_loop2
Next dat_loop
```

<u>Errata:</u>
UEI Function calls used:

```
        vb_allocateUEI_float
        vb_reallocateUEI_float
        vb_writeUEI_float
```

<u>DATA TYPES: (Extracted from Visual BASIC 5.0)</u>

Data type	Storage size	Description
BINARY	1 byte per character	Any type of data may be stored in a field of this type. No translation of the data (for example, to text) is made. How the data is input in a binary field dictates how it will appear as output.
BIT	1 byte	Yes and No values and fields that contain only one of two values.
BYTE	1 byte	An integer value between 0 and 255.
COUNTER	4 bytes	A number automatically incremented by the Microsoft Jet database engine whenever anew record is added to a table. In the Microsoft Jet database engine, the data type for this value is Long.
CURRENCY	8 bytes	A scaled integer between – 922,337,203,685,477.5808 and 922,337,203,685,477.5807.
DATETIME (See DOUBLE)	8 bytes	A date or time value between the years 100 and 9999.
GUID	128 bits	A unique identification number used with remote procedure calls.
SINGLE	4 bytes	A single-precision floating-point value with a range of – 3.402823E38 to – 1.401298E-45 for negative values, 1.401298E-45 to 3.402823E38 for positive values, and 0.
DOUBLE	8 bytes	A double-precision floating-point value with a range of – 1.79769313486232E308 to – 4.94065645841247E-324 for negative values, 4.94065645841247E-324 to 1.79769313486232E308 for positive values, and 0.
SHORT	2 bytes	A short integer between – 32,768 and 32,767.

LONG	4 bytes	A long integer between – 2,147,483,648 and 2,147,483,647.
LONGTEXT	1 byte per character	Zero to a maximum of 1.2 gigabytes.
LONGBINARY	As required	Zero to a maximum of 1.2 gigabytes. Used for OLE objects.
TEXT	1 byte per character	Zero to 255 characters.