ASO-ADC-16
User's Guide

Revision A
Printed February, 1993
Part No. 24460

WARNING

Keithley Data Acquisition assumes no liability for damages consequent to the use of this Product. This Product is not designed with components of a level of reliability that is suitable for use in life support or critical applications.

The information contained in this manual is believed to be accurate and reliable. However, Keithley Data Acquisition assumes no responsibility for its use; nor for any infringements or patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent rights of Keithley Data Acquisition.

Keithley Data Acquisition does not warrant that the Product will meet the Customer's requirements or will operate in the combinations which may be selected for use by the Customer or that the operation of the Program will be uninterrupted or error free or that all Program defects will be corrected.

Keithley Data Acquisition does not and cannot warrant the performance or results that may be obtained by using the Program. Accordingly, the Program and its documentation are sold "as is" without warranty as to their performance merchantability, or fitness for any particular purpose. The entire risk as to the results and performance of the program is assumed by you.

Keithley Data Acquisition • 440 Myles Standish Blvd. • Taunton, MA 02780
Telephone: (508) 880-3000 • Fax: (508) 880-0179

Contents

Introduction 1

1.1 About the ASO-ADC-16

The ASO-ADC-16 is the Advanced Software Option (ASO) for the ADC-16 analog input and digital I/O board. The ASO includes a set of software components that you can use, in conjunction with a programming language, to create application programs that execute the operations available on the ADC-16.

The two primary components of the ASO are the Function Call Driver and the File I/O Driver. These drivers represent two distinct methods of providing your application program with high-level access to the acquisition and control operations available on the ADC-16. The ASO also includes support files, example programs, and a configuration utility.

The Function Call Driver and the File I/O Driver are independent of each other; your application program will use one or the other, but not both. The two drivers are implemented differently and provide slightly different functionality. You should use whichever driver is appropriate for your programming skills and your application's requirements.

Function Call Driver

The Function Call Driver enables your program to define and execute board operations via calls to driver-provided functions. For example, your program can call the driver-provided **K_ADRead** function to execute a single-point, A/D input operation.

The ASO includes several different versions of the Function Call Driver. The .LIB and .TPU versions are provided for DOS application development. The Dynamic Link Library (DLL) is provided for Windows application development.

The ASO and this manual provide the necessary tools, example programs and information to develop Function Call Driver programs in the following languages:

- Borland C++ (version 2.0 and higher)
- Borland Turbo C (version 2.01)
- Borland Turbo Pascal (version 6.0)
- Borland Turbo Pascal for Windows (version 1.0)
- Microsoft C (version 5.1 and above)
- Microsoft Quick C for Windows (version 1.0)
- Microsoft Visual Basic (version 1.0 and higher)

File I/O Driver

The File I/O Driver enables your program to define, execute, and retrieve the results of board operations by writing (to the driver) driver -provided File I/O Commands. For example, your program can write the **Read Channel 1** command to execute a single-point, A/D input operation.

You can use the File I/O Driver to create DOS applications with any language that supports file I/O. The ASO and this manual provide the necessary tools, example programs and information to develop File I/O Driver programs in the following languages:

- Interpreted BASIC
- QuickBASIC
- Borland Turbo C
- Borland Turbo Pascal
- Microsoft C
- Microsoft Pascal

1.2 Prerequisites

The ASO is designed exclusively for use with the ADC-16. This manual assumes that you understand the information presented in the *ADC-16 User's Guide*. Additionally, you must complete the board installation and configuration procedures outlined in the *ADC-16 User's Guide* before you attempt any of the procedures described in this manual.

The fundamental goal of this manual is to provide you with the information you need to write ADC–16 application programs that use the ASO drivers. It is recommended that you proceed through this manual according to the sequence suggested by the table of contents; this will minimize the amount of time and effort required to develop your ASO-ADC-16 application programs.

1.3 Getting additional help

The following resources provide information about using the ASO:

- this manual
- the *ADC-16 User's Guide*
- the ASO example programs (these are copied to your system's hard disk during the installation procedure)
- the documentation for the programming language you are using

Call our Technical Support Department if you need additional assistance. A Technical Support Engineer will help you diagnose and solve your problem over the telephone.

> Keithley Data Acquisition – Technical Support
> 508–880–3000
> Monday – Friday, 8 A.M. – 7 P.M.

For the most efficient and helpful assistance, please compile the following information before calling our Technical Support Department:

ASO package	Version	_____
	Invoice/Order #	_____
ADC-16	Serial #	_____
	Base address setting	_____
	A/D full-scale setting	±3.2768 V ±5 V
STA-EX8	Number installed	_____
Computer	Manufacturer	_____
	CPU type	8088 286 386 486 Other
	Clock speed (MHz)	8 12 20 25 33 Other
	Math co-processor?	Yes No
	Amount of RAM	_____
	Video system	CGA Hercules EGA VGA
Compiler	Language	_____
	Manufacturer	_____
	Version	_____

1.4 Installing the ASO

The files on these ASO distribution diskettes are in compressed format. You must use the installation program included on the diskettes to install the ASO software. Since the aggregate size of the expanded ASO files is approximately 1.0 MB, check that there is at least this much space available on your PC's hard disk before you attempt to install the ASO.

Perform the following procedure to install the ASO software (note that it is assumed that the floppy drive is designated A:):

1. Make a back-up copy of the distribution diskette(s).
2. Insert ASO diskette #1 into the floppy drive
3. Type the following commands at the DOS prompt:

   A: [Enter↵]
   install [Enter↵]

The installation program prompts you for your installation preferences, including the name of the directory into which the ASO files will be copied. The installation program expands the files on the ASO diskette(s) and copies them into the directory you specified; refer to the file FILES.DOC in the ASO installation directory for the names and descriptions of these files.

The Function Call Driver 2

2.1 Available operations

The Function Call Driver provides functions through which an application program can perform the following operations:

Immediate-execution operations

- Single-value A/D input
- Single-value digital input
- Single-value digital output

Frame-based operations

- Multi-value, interrupt-mode A/D input
- Multi-value, synchronous-mode A/D input

Immediate-execution operations and frame-based operations are described in the following subsections.

Immediate-execution operations

The three immediate-execution operations and the Callable Function associated with each are as follows:

- Single-value A/D input: **K_ADRead**
- Single-value digital input: **K_DIRead**
- Single-value digital output: **K_DOWrite**

The calling arguments for these functions define the attributes of the associated operation. Upon receipt of a call to one of these functions, the driver immediately executes the associated operation.

Frame-based operations

The two frame-based operations and the Callable Function associated with each are as follows:

- Multi-value, interrupt-mode A/D input: **K_IntStart**
- Multi-value, synchronous-mode A/D input: **K_SyncStart**

The description of frame-based operations requires the introduction of a few new terms.

A *frame* is a data structure whose elements correspond to the defining attributes of a board operation. The driver uses two different types of frames: A/D and Digital Output frames. The driver maintains a pool of four A/D frames and four Digital Output frames.

The values of a frame's elements define the operation's attributes. For example, the elements contained in an A/D frame are as follows:

- Start Channel – defines the first channel in a scan
- Stop Channel – defines the last channel in a scan
- Gain element – defines the gain applied to all channels in the scan

The driver provides functions that set the value of one or more elements. For example, **K_SetG** sets the value of a frame's Gain element, and **K_SetStartStopChn** sets the values of a frame's Start Channel and Stop Channel elements.

A *frame handle* is a variable whose value identifies a frame. The sole purpose of a frame handle is to provide a mechanism through which different function calls can reference the same frame.

A *device handle* is a variable whose value identifies an installed board. The sole purpose of a device handle is to provide a mechanism through which different function calls can reference the same board.

A frame-based operation is so-called because the function that performs the operation uses a frame handle as its single calling argument. The frame handle identifies a frame whose element values are the operation's attributes. The values of all of a frame's elements must be set before that frame's handle can be used as a calling argument to a function that executes a frame-based operation.

2.2 Overview of programming with the Function Call Driver

The procedure to write a Function Call Driver program is as follows:

1. Define the application's requirements.
2. Write the program code.
3. Compile and link the program.

The subsections that follow describe the details of each of these steps.

Defining the application's requirements

Before you begin writing the program code, you should have a clear idea of the board operations you expect your program to execute. Additionally, you should determine the sequence in which these operations must be executed and the characteristics (number of channels, gains, and so on) that define each operation. You may find it helpful to review the list of available operations in Section 2.1 and to browse through the short descriptions of the Callable Functions in Section 3.1.

Writing the program code

Several sources of information relate to this step:

- Section 2.3 explains the programming tasks that are common to all Function Call Driver programs
- Section 2.4 describes the sequence of function calls required to execute each of the available operations
- Section 3.2 provides detailed information on individual functions
- The ASO includes several example source code files for Function Call Driver programs. The FILES.DOC file in the ASO installation directory lists and describes the example programs.

The phrase *general programming tasks*, as it is used in this chapter, refers to the programming tasks that every Function Call Driver program must execute. The task of obtaining a device handle, for example, qualifies as a general programming task, since the sequence of function calls required to execute any of the available board operations includes at least one function whose calling arguments include a device handle. Section 2.3 provides the details of the general-programming tasks.

Each available operation also has an associated set of tasks that the program must perform to execute the operation; these are referred to as *operation-specific programming tasks*. Section 2.4 provides the details of the operation-specific programming tasks for each available operation.

Compiling and linking the program

Refer to Section 2.5 for compile and link instructions and other language-specific considerations for each supported language.

2.3 General programming tasks

Every Function Call Driver program must execute the following programming tasks:

1. Identify a function/variable type definition file
   The method to identify this file is language-specific; refer to Section 2.5 for additional information.
2. Declare/initialize program variables
3. Call **ADC16_DevOpen** to initialize the driver
4. Call **ADC16_GetDevHandle** to initialize the board and get a device handle for the board

The tasks listed are the minimum tasks your program must complete before it attempts to execute any operation-specific tasks. Your application may require additional general-programming tasks. For example, if your program requires access to two boards, then it must call ADC16_GetDevHandle for each board.

2.4 Operation-specific programming tasks

This section describes the set of programming tasks that your program must perform to execute the following operations:

- Single-value A/D input
- Single-value digital input
- Single-value digital output
- Interrupt-mode A/D input using channel-gain array
- Synchronous-mode A/D input using channel-gain array
- Interrupt-mode A/D input using start/stop channels
- Synchronous-mode A/D input using start/stop channels

The set of tasks listed for each operation are valid only if the application program has already completed the general-programming tasks.

Single-value A/D input

To execute a single-value A/D input, your program must call **K_ADRead**. The calling arguments identify the board that executes the operation, the channel on which the value is acquired, the gain applied to that channel, and the buffer in which the value is stored.

Single-value digital input

To execute a single-value digital input, your program must call **K_DIRead**. The calling arguments identify the board that executes the operation, the channel on which the value is acquired, and the buffer in which the value is stored.

Single-value digital output

To execute a single-value digital output, your program must call **K_DOWrite**. The calling arguments identify the board that executes the operation, the channel on which the value is written, and the buffer from which the value is written.

Interrupt-mode A/D input using start/stop channels

Your program must perform the following tasks to execute an interrupt-mode A/D input operation whose channel-scanning sequence is given by the sequence's start and stop channels:

1. Allocate a buffer in which the driver stores the A/D values. Use **K_INTAlloc** if you want to allocate this buffer outside the program's memory area (you must use **K_INTAlloc** if you are writing an application that will execute in Windows standard mode).

2. Call **K_GetADFrame** to get the handle to an A/D frame.

3. Call **K_SetBuf** to assign the buffer address obtained in step 1 to the Buffer Address element in the frame associated with the frame handle obtained in step 2.

4. Call **K_SetStartStopG** or **K_SetStartStopChn** and **K_SetG** to assign values to the Start Channel, Stop Channel, and Gain elements in the frame associated with the frame handle obtained in step 2.

5. Call **K_INTStart** to start the operation.

6. Call **K_INTStatus** to monitor the status of the operation.

7. (Optional for C and Pascal programs)
   Call **K_MoveDataBuf** to transfer the acquired data from the buffer to a user-defined array.

8. If **K_INTAlloc** was used to allocate a buffer in step 1, call **K_INTFree** to deallocate the buffer.

9. Call **K_FreeFrame** to return the frame (associated with the frame handle from step 2) to the pool of available frames.

Interrupt-mode A/D input using channel-gain array

Your program must perform the following tasks to execute an interrupt-mode A/D input operation whose channel-scanning sequence is given by a channel-gain array:

1. Define and assign values to a channel-gain array. The format and other information pertaining to channel-gain arrays is listed under the reference entry for **K_SetChnGAry** on page 60.

2. Allocate a buffer in which the driver stores the A/D values. Use **K_INTAlloc** if you want to allocate this buffer outside the program's memory area (you must use **K_INTAlloc** if you are writing an application that will execute in Windows standard mode).

3. Call **K_GetADFrame** to get the handle to an A/D frame.

4. Call **K_SetBuf** to assign the buffer address obtained in step 2 to the Buffer Address element in the frame associated with the frame handle obtained in step 3.

5. Call **K_SetChnGAry** to assign the channel-gain array from step 1 to the Channel-Gain Array Address element in the frame associated with the frame handle obtained in step 3.

6. Call **K_INTStart** to start the operation.

7. Call **K_INTStatus** to monitor the status of the operation.

8. (Optional for C and Pascal programs)
   Call **K_MoveDataBuf** to transfer the acquired data from the buffer to a user-defined array.

9. If **K_INTAlloc** was used to allocate a buffer in step 2, call **K_INTFree** to deallocate the buffer.

10. Call **K_FreeFrame** to return the frame (associated with the frame handle from step 3) to the pool of available frames.

Synchronous-mode A/D input using start/stop channels

Your program must perform the following tasks to execute a synchronous-mode A/D input operation whose channel-scanning sequence is given by the sequence's start and stop channels:

1. Allocate a buffer in which the driver stores the A/D values. Use **K_INTAlloc** if you want to allocate this buffer outside the program's memory area.

2. Call **K_GetADFrame** to get the handle to an A/D frame.

3. Call **K_SetBuf** to assign the buffer address obtained in step 1 to the Buffer Address element in the frame associated with the frame handle obtained in step 2.

4. Call **K_SetStartStopG** or **K_SetStartStopChn** and **K_SetG** to assign values to the Start Channel, Stop Channel, and Gain elements in the frame associated with the frame handle obtained in step 2.

5. Call **K_SyncStart** to start the operation.

6. (Optional for C and Pascal programs)
   Call **K_MoveDataBuf** to transfer the acquired data from the buffer to a user-defined array.

7. If **K_INTAlloc** was used to allocate a buffer in step 1, call **K_INTFree** to deallocate the buffer.

8. Call **K_FreeFrame** to return the frame (associated with the frame handle from step 2) to the pool of available frames.

Synchronous-mode A/D input using channel-gain array

Your program must perform the following tasks to execute a synchronous-mode A/D input operation whose channel-scanning sequence is given by a channel-gain array:

1. Define and assign values to a channel-gain array. The format and other information pertaining to channel-gain arrays is listed under the reference entry for **K_SetChnGAry** on page 60.

2. Allocate a buffer in which the driver stores the A/D values. Use **K_INTAlloc** if you want to allocate this buffer outside the program's memory area.

3. Call **K_GetADFrame** to get the handle to an A/D frame.

4. Call **K_SetBuf** to assign the buffer address obtained in step 2 to the Buffer Address element in the frame associated with the frame handle obtained in step 3.

5. Call **K_SetChnGAry** to assign the channel-gain array from step 1 to the Channel-Gain Array Address element in the frame associated with the frame handle obtained in step 3.

6. Call **K_SyncStart** to start the operation.

7. (Optional for C and Pascal programs)
   Call **K_MoveDataBuf** to transfer the acquired data from the buffer to a user-defined array.

8. If **K_INTAlloc** was used to allocate a buffer in step 1, call **K_INTFree** to deallocate the buffer.

9. Call **K_FreeFrame** to return the frame (associated with the frame handle from step 3) to the pool of available frames.

2.5 Language-specific programming notes

This section provides specific programming guidelines for each of the supported languages. Additional programming information is available in the ASO example programs. Refer to the FILES.DOC file for names and descriptions of the ASO example programs.

Borland C++, Microsoft C and Borland Turbo C

Related files

ADC16.LIB
DASRFACE.LIB
USERPROT.H

Compile and Link instructions

Borland C++:

```
BCC -c -ml filename.c
TLINK c0l+filename,filename,,adc16+dasrface+cl;
```

Microsoft C:

```
CL /AL /c filename.c
LINK filename,,,ADC16+DASRFACE;
```

Turbo C:

```
TCC -c -ml filename.c
TLINK c0l+filename,filename,,adc16+dasrface+cl;
```

Example program

Execute a single A/D conversion

```
/* C include files */
#include "stdio.h"
#include "stdlib.h"

/* ADC-16 driver include file */
#include "userprot.h"

/* Local variables */
DDH ADC16;                              /* Device Handle */
char NumOfBoards;                       /* #boards in ADC16.CFG */
int Err;                                /* Function ret err flag */
long Advalue;                           /* Storage for A/D value */

/* Begin main module */
main()
{

/* Initialize the hardware/software */
if (( Err = ADC16_DevOpen( "ADC16.CFG", &NumofBoards )) !=0)
{
putch (7); printf( " Error %X during DevOpen ", Err );
exit(Err);
}
/* Establish communication with the driver */
/* through a device handle */
if ( ( Err = ADC16_GetDevHandle( 0, &ADC16 ) ) != 0 )
{
putch (7); printf("Error %X during GetDevHandle ",Err);
exit(Err);
}
/* Read channel 0 at gain 1; store sample in Advalue */
if ((Err = K_ADRead (ADC16, 0, 0, &ADvalue)) != 0)
{
putch(7); printf ("Error %X in K_ADRead operation ", Err);
exit(Err);
}

/* Display ADvalue */
printf ("A/D value from channel 0 is : %x\n", ADvalue);
}
```

Borland C++

If you want to compile a Borland C++ program as a standard C program, refer to the information presented in the previous section. If you want to compile your program as a Borland C++ program, refer to the information presented in the previous section with the following exceptions:

1. Use the supplied file USERPROT.BCP instead of USERPROT.H.
2. Specify the C++ compilation in one of the following two ways:
   a. Specify .CPP as the extension for your source file, or
   b. Use the BCC –P command line switch.

Borland Turbo Pascal

Compile and Link instructions

TPC *filename*.pas

Example program

Execute a single A/D conversion

```
Program TPEXAMPLE;
{ UNITS USED BY THIS PROGRAM }
Uses Crt, ADC16;
{ LOCAL VARIABLES }
Var
Devhandle : Longint; { Device Handle }
ConfigFile : String; { String to hold name of configuration file }
NumOfBoards : Integer;
BoardNumber : Integer;
Ertn : Word; { Error flag }
Gain : Byte; { Overall gain }
ADvalue : Longint; { Holds A/D sample }
Chan : Byte; { A/D channel }
{ BEGIN MAIN MODULE }
BEGIN
{ STEP 1: This step is mandatory; it initializes the
internal data tables according to the information
contained in the configuration file ADC16.CFG.
}
ConfigFile := 'ADC16.CFG' + #0;
Ertn := ADC16_DevOpen( ConfigFile[1], NumOfBoards );
IF Ertn <> 0 THEN
BEGIN
  writeln( 'Error ', Ertn, 'on Device open' );
  Halt(1);
END;
{ STEP 2: This step is mandatory; it establishes
communication with the driver through the
Device Handle.
}
```

```
BoardNumber := 0;
Ertn := ADC16_GetDevHandle( BoardNumber, Devhandle );
IF Ertn <> 0 THEN
BEGIN
  writeln( 'Error ', Ertn, ' getting Device Handle' );
  Halt(1);
END;
{ STEP 3: Read A/D sample from channel 0 at gain 1
(Gain Code 0) and store in local variable.
}
Chan := 0;
Gain := 0;
Ertn := K_ADRead(Devhandle, Chan, Gain, ADvalue);
IF Ertn <> 0 THEN
BEGIN
  writeln(^G, 'Error # ',Ertn, 'Occurred during K_ADRead call');
  Halt(1);
END;
writeln('A/D VALUE : ', ADvalue);
END.
```

Borland Turbo Pascal for Windows

Related files

ADC16TPW.INC
ADC16.DLL

Notes

If you use ADC16.DLL, the information presented for Borland Turbo Pascal applies here with the following additions:

- Use the compiler directive {$I ... } to include the supplied include file ADC16TPW.INC.
- Substitute 'WinCrt' for the 'Crt' unit; this is necessary in order that the console I/O procedures (writeln, readln, etc...) operate properly.

The following code fragment illustrates these additions:

```
Program TPW_EX;
{ UNITS USED BY THIS PROGRAM }
Uses WinCrt;
⋮
{ LOCAL VARIABLES }
Var
⋮
{ ADC16 function prototypes that reference .DLL }
{$I ADC16TPW.INC}
{ BEGIN MAIN MODULE }
BEGIN
⋮
```

If you use ADC16TPW.INC, the information presented for Borland Turbo Pascal applies here with the following exceptions:

- Substitute ADC16TPW.INC for the ADC16 unit.
- Substitute 'WinCrt' for the 'Crt' unit; this is necessary in order that the console I/O procedures (writeln, readln, etc...) operate properly.

The following code fragment illustrates these substitutions:

```
Program TPW_EX;
{ UNITS USED BY THIS PROGRAM }
Uses WinCrt, ADC16TPW;
  .
  .
{ LOCAL VARIABLES }
Var
  .
  .
{ BEGIN MAIN MODULE }
BEGIN
  .
  .
```

Microsoft Quick C for Windows

Related files

ADC16.DLL

Compile and Link instructions

1. Load *filename*.c into the Quick C for Windows environment.
2. Create a project file.
3. Select PROJECT ▸ BUILD to create a stand-alone .EXE that can be executed from within Windows.

Notes

The programming procedure required to call the Callable Functions from Quick C for Windows programs is identical to the procedure described for Microsoft C.

Microsoft Visual Basic for Windows

Related files

ADC16.DLL
ADC16EX.BAS

Notes

Before you begin coding your Visual Basic program, you must copy (from inside the Visual Basic environment) the contents of ADC16EX.BAS into your application's GLOBAL.BAS. Use the following procedure to add the contents of ADC16EX.BAS to GLOBAL.BAS (you should make a back-up copy of GLOBAL.BAS before you modify it):

1. Select FILE ▸ ADD FILE... from the Visual Basic main menu.
2. Select ADC16EX.BAS.
3. Highlight the contents of the entire ADC16EX.BAS file.
4. Select EDIT ▸ COPY to copy the contents of ADC16EX.BAS to the Windows clipboard.
5. Double-click on GLOBAL.BAS in the Project window.
6. Select EDIT ▸ PASTE.
7. Select FILE ▸ SAVE PROJECT.

Example program

Execute a single A/D conversion.

```
Sub Command1_Click ()
board% = 0
Cls

For x = 0 to 9' Clear our buffer
lbuffer(x) = 0
Next x

MyErr = ADC16_devopen("..\ADC16.CFG", board%)
If MyErr <> 0 Then
  MsgBox "ADC16_devopen Error", 48, "Error"
  GoTo exy1
End If

Print
Print "Scanning Channels "; strtch; "-"; stpch

MyErr = ADC16_getdevhandle(0, adc16)
If MyErr <> 0 Then
  MsgBox "ADC16_getdevhandle Error", 48, "Error"
  GoTo exy1
End If

Print
Print "AD Data :"
Print

For x = strtch to stpch
  MyErr = K_ADRead(adc16, x, Chgain, retval)
  lBuffer(x) = retval
  Print "    Channel "; x; " = "; Hex$(lBuffer(x))
Next x

Print
Print

exy1:

End Sub
```

Callable Functions 3

3.1 Functional grouping

The Callable Functions can be classified according to the functionality that each provides. This section lists each Callable Function as a member of one of the following groups:

- Initialization
- Memory management
- Frame management
- Frame-element management
- Frame-based operation control
- Immediate-execution operations
- Miscellaneous operations

This section provides short descriptions of each function; refer to Section 3.2 for additional information on each function.

Initialization

ADC16_DevOpen	Initialize and configure the driver.
ADC16_GetDevHandle	Obtain a device handle.
K_DASDevInit	Reset and initialize the device and driver.

Memory management

K_IntAlloc	Allocate a buffer suitable for an interrupt-mode A/D operation.
K_IntFree	De-allocate an interrupt buffer that was previously allocated with **K_IntAlloc**.
K_MoveDataBuf	Transfer acquired A/D samples between a memory buffer and an array.

Frame management

K_FreeFrame	Free the memory used by a frame and return the frame it to the pool of available frames.
K_GetADFrame	Obtain the handle to an A/D frame.
K_GetDOFrame	Obtain the handle to a digital output frame.

Frame-element management

K_ClearFrame	Set all the elements of an A/D frame to their default values.
K_GetBuf	Get the values of an A/D frame's Buffer Address and Number of Samples elements.
K_GetChn	Get the value of an A/D frame's Start Channel element.
K_GetChnGAry	Get the value of an A/D frame's Channel-Gain Array Address element.
K_GetDOCurVal	Get the value of a digital output frame's Digital Output Value element.

Frame-element management (cont'd)

K_GetG	Get the value of an A/D frame's Gain Code element.
K_GetStartStopChn	Get the values of an A/D frame's Start Channel and Stop Channel elements.
K_GetStartStopG	Get the values of an A/D frame's Start Channel, Stop Channel, and Gain Code elements.
K_InitFrame	Initialize a board's A/D circuitry and set an A/D frame's elements to their default values.
K_SetBuf	Set the values of an A/D frame's Buffer Address and Number of Samples elements.
K_SetChn	Set the value of an A/D frame's Start Channel element.
K_SetChnGAry	Set the value of a frame's Channel-Gain Array Address element.
K_SetG	Set the value of an A/D frame's Gain Code element.
K_SetStartStopChn	Set the values of an A/D frame's Start Channel and Stop Channel elements.
K_SetStartStopG	Set the values of an A/D frame's Start Channel, Stop Channel, and Gain Code elements.

Frame-based operation control

K_IntStart	Start an interrupt-mode A/D operation.
K_IntStatus	Determine the status of an interrupt-mode A/D operation.
K_IntStop	Abort an interrupt-mode A/D operation.
K_SyncStart	Start a synchronous-mode A/D operation.

Immediate-execution operations

K_ADRead	Read a single A/D value.
K_DIRead	Read a single digital value.
K_DOWrite	Write a single digital value.

Miscellaneous operations

K_GetErrMsg	Get the address of an error message string (available only as C-language function).
K_GetVer	Determine the driver revision and driver specification.

3.2 Function reference

This section contains reference entries for the Callable Functions. The entries appear one per page and in ascending alphabetical order (by function name). These reference entries provide the details associated with the use of each function.

This section is not a good resource for general and conceptual information about writing Function Call Driver programs. Moreover, much of the information presented here requires a thorough understanding of the concepts presented in Chapter 2. *Do not expect to write a Function Call Driver program merely by consulting the reference entries for the functions you expect to use in your program.*

The information related to the following topics pertains to several Callable Functions:

- the format of A/D values and the procedure to determine the voltage that produced a specific A/D value
- the gain codes the driver uses to represent gains and the A/D input ranges that correspond to each gain
- the return value for every call to a Callable Function

These topics are described in the next several paragraphs and referred to throughout the reference entries that follow.

A/D values and corresponding voltages

There are three Callable Functions through which your program can acquire A/D values: K_ADRead, K_IntStart, and K_SyncStart. Although the method to create/assign a storage buffer for the acquired value(s) is different for each of these functions, they all store the A/D value in the same format. Consequently, the interpretation of the A/D data is the same regardless of the function with which it was acquired.

The driver configuration file specifies two attributes that affect how you should interpret A/D values: the A/D Number Type and the A/D Full Scale Range. The possible values for these attributes are as follows:

- A/D Number Type: ***Sign/Magnitude*** or ***2's Complement***
- A/D Full Scale Range: ***±3.2767 V*** or ***±5.0 V***

The procedure to determine the voltage that produced a particular A/D value depends on the A/D Number Type. The two cases are presented below. The following variables are used in both cases:

- *range* is the maximum voltage in the range specified by the A/D Full Scale Range, which is either 3.2767 V or 5.0 V.
- *ADvalue* is the value acquired by the A/D operation

Case 1 A/D Number Type = Sign/Magnitude

If bit 15 = 0,

$$\text{voltage} = \frac{ADvalue \text{ AND } 7FFF}{-32{,}767} \times range$$

If bit 15 = 1,

$$\text{voltage} = \frac{ADvalue \text{ AND } 7FFF}{32{,}767} \times range$$

Case 2 A/D Number Type = 2's Complement

If bit 15 = 0,

$$\text{voltage} = \frac{ADvalue}{32{,}767} \times range$$

If bit 15 = 1,

$$\text{voltage} = \frac{(ADvalue)^{2's} \text{ AND } 7FFF}{-32{,}767} \times range$$

where $(ADvalue)^{2's}$ is the 2's complement of *ADvalue*.

Gain codes

The Function Call Driver uses gain codes to indicate gains. The valid gain codes are 0, 1, 2. The table below lists the gain that corresponds to each gain code. Additionally, this table shows the A/D input range for both settings of the A/D Full Scale Range (the A/D Full Scale Range is specified by the driver configuration file).

gain code	gain	A/D input range for ±3.2767 V full-scale range	A/D input range for ±5.0 V full-scale range
0	1	±3.2767 V	±5 V
1	10	±327.67 mV	±500 mV
2	100	±32.767 mV	±50 mV

Return values

Every call to a Callable Function returns an integer-type (16-bit) return value. A return value of 0 indicates that the function executed successfully; a non-zero return value indicates an error. The non-zero return values correspond to error codes; these error codes and their corresponding errors are listed in Appendix A. Your program should always check a function call's return value and, in the case of an error, perform an appropriate action.

ADC16_DevOpen

Purpose

Initialize and configure the driver.

Prototype

C

DASErr far pascal ADC16_DevOpen(char far * *cfgFile*,
char far * *numDevices*);

Pascal

Function ADC16_DevOpen(Var *cfgFile* : char;
Var *numDevices* : Integer) : Word;

Visual Basic for Windows

ADC16_DevOpen Lib "ADC16.dll" (ByVal *cfgFile$*,
numDevices As Integer) As Integer

Parameters

cfgFile	Driver configuration file
numDevices	Number of devices defined in *cfgFile*. Valid values: 1, 2

Notes

ADC16_DevOpen initializes the driver according to the information in *cfgFile*. On return, *numDevices* contains the number of devices for which *cfgFile* contains configuration information.

ADC16_DevOpen writes a zero value to OP0 and OP1; this turns off the ADC-16's relay 0 and relay 1.

Specify –1 for *cfgFile* to set the driver to its default configuration; the default configuration specifies that the device is set as follows:

Board number	0	1
Board name	ADC16	ADC16
Base address	300 Hex	308 Hex
Range	±3.2767 V	±3.2767 V
A/D Number Type	SignMagnitude	SignMagnitude
Interrupt level	A Hex	F Hex
Installed STA-EX8s	0	0

Purpose

Obtain a device handle.

Prototype

C

DASErr far pascal ADC16_GetDevHandle (int *devNumber*,
void far * far * *devHandle*);

Pascal

Function ADC16_GetDevHandle(*devNumber* : Integer;
Var *devHandle* : Longint) : Word;

Visual BASIC for Windows

ADC16_GetDevHandle Lib "ADC16.dll" (ByVal *devNumber* As Integer,
devHandle As Long) As Integer

Parameters

devNumber Device number. Valid values: **0**, **1**

devHandle Device handle

Notes

On return, *devHandle* contains the handle associated with the device identified by *devNumber*.

The value returned in *devHandle* is intended to be used exclusively as an argument to functions that require a device handle. Your program should not modify the value returned in *devHandle*.

The driver supports up to two devices; a unique handle is associated with each supported device.

In addition to obtaining a device handle, **ADC16_GetDevHandle** performs the following tasks:

- aborts all in-progress A/D operations
- writes a 0 to OP0 and OP1
- checks if device identified by *devHandle* is present
- checks if settings in configuration file match actual board settings
- initializes the board to its default state

K_ADRead

Purpose

Read a single A/D value.

Prototype

C

DASErr far pascal K_ADRead(DDH *devHandle*, unsigned char *chan*, unsigned char *gainCode*, void far * *ADvalue*);

Pascal

Function K_ADRead(*devHandle* : Longint; *chan* : Byte; *gainCode* : Byte; Var *ADvalue* : Longint) : Word;

Visual BASIC for Windows

K_ADRead Lib "ADC16.dll" (ByVal *devHandle* As Long, ByVal *chan* As Integer, ByVal *gainCode* As Integer, *ADvalue* As Long) As Integer

Parameters

devHandle	Handle to acquisition device
chan	Input channel. Valid values: **0**, **1**,..., **7(m+1)**, where **m** is the number of connected STA-EX8.
gainCode	Gain code. Valid values: **0** - 1x, **1** - 10x, **2** - 100x
ADvalue	Storage location of acquired A/D value

Notes

On return, *ADvalue* contains the value read from channel *chan* (at the gain indicated by *gain code*) of the device identified by *devHandle*.

See page 29 for the procedure to determine the voltage that produced the value returned in *ADvalue*.

See page 31 for the A/D voltage ranges that correspond to each gain.

Purpose

Set all the elements of an A/D frame to their default values.

Prototype

C

DASErr far pascal K_ClearFrame(FRAMEH *frameHandle*);

Pascal

Function K_ClearFrame(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_ClearFrame Lib "ADC16.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Frame handle

Notes

On return, the elements in the frame identified by *frameHandle* contain the following values:

Buffer Address	0
Start Channel	0
Stop Channel	0
Gain Code	0
Channel-Gain Array Address	0

K_DASDevInit

Purpose

Reset and initialize the device and driver.

Prototype

C

DASErr far pascal K_DASDevInit(DDH *devHandle*);

Pascal

Function K_DASDevInit(*devHandle* : Longint) : Word;

Visual BASIC for Windows

K_DASDevInit Lib "ADC16.dll" (ByVal *devHandle* As Long)
As Integer

Parameters

devHandle Device handle

Notes

K_DASDevInit performs the following tasks:

- Aborts all in-progress A/D operations
- Writes a 0 to OP0 and OP1
- Checks if device identified by *devHandle* is present
- Checks if settings in configuration file match actual board settings
- Initializes the board to its default state

Purpose

Read a single digital value.

Prototype

C

DASErr far pascal K_DIRead(DDH *devHandle*, unsigned char *chan*, void far * *DIvalue*);

Pascal

Function K_DIRead(*devHandle* : Longint; *chan* : Byte; Var *DIvalue* : Longint) : Word;

Visual Basic for Windows

K_DIRead Lib "ADC16.dll" (ByVal *devHandle* As Long, ByVal *chan* As Integer, *DIvalue* As Long) As Integer

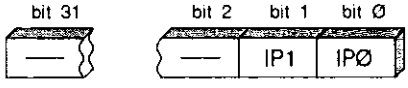
Parameters

devHandle	Device handle
chan	Digital input channel. Valid value: **0**
DIvalue	Digital input value. Valid values: **0**, **1**, **2**, **3**

Notes

On return, *DIvalue* contains the digital value read from channel *chan* of the device identified by *devHandle*.

DIvalue is a 32-bit variable. The acquired digital value is stored in bits 0 and 1; the values in the remaining bits of *DIvalue* are not well-defined. The figure below illustrates the format of *DIvalue*.



K_DOWrite

Purpose

Write a single digital value.

Prototype

C

DASErr far pascal K_DOWrite(DDH *devHandle*, unsigned char *chan*, long *DOvalue*);

Pascal

Function K_DOWrite(*devHandle* : Longint; *chan* : Byte; *DOvalue* : Longint) : Word;

Visual Basic for Windows

K_DOWrite Lib "ADC16.dll" (ByVal *devHandle* As Long, ByVal *chan* As Integer, ByVal *DOvalue* As Long) As Integer

Parameters

devHandle	Device handle
chan	Digital output channel. Valid value: **0**
DOvalue	Digital output value. Valid values: **0**, **1**,..., **31**

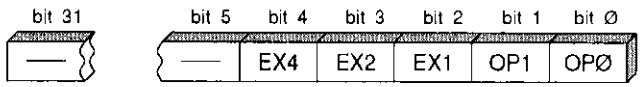
Notes

K_DOWrite outputs the value in *DOvalue* to channel *chan* on the device identified by *devHandle*.

DOvalue is a 32-bit variable; the significance of the bits in *DOvalue* depends on if there is a connection between the board and an STA-EX8:

If the board is not connected to an STA-EX8:

The output value comprises the values in bits 0 – 4; the values in bits 5 – 31 are not significant. This format is illustrated in the following figure:



If the board is connected to one or more STA-EX8:

The output value comprises the values in bits 0 and 1; the values in bits 2 – 31 are not significant. This format is illustrated in the following figure:

Purpose

Free the memory used by a frame and return the frame it to the pool of available frames.

Prototype

C

DASErr far pascal K_FreeFrame(FRAMEH *frameHandle*);

Pascal

Function K_FreeFrame(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_FreeFrame Lib "ADC16.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Frame handle

Notes

K_FreeFrame frees the memory used by the frame identified by *frameHandle*; the frame is then returned to the pool of available frames. The pool of available frames initially contains two A/D frames and two digital output frames.

K_GetADFrame

Purpose

Obtain the handle to an A/D frame.

Prototype

C

DASErr far pascal K_GetADFrame(DDH *devHandle*,
FRAMEH far * *frameHandle*);

Pascal

Function K_GetADFrame(*devHandle* : Longint;
Var *frameHandle* : Longint) : Word;

Visual Basic for Windows

K_GetADFrame Lib "ADC16.dll" (ByVal *devHandle* As Long,
frameHandle As Long) As Integer

Parameters

devHandle	Device handle
frameHandle	Handle to A/D frame

Notes

On return, *frameHandle* contains the handle to an A/D frame associated with the device identified by *devHandle*.

Purpose

Get the values of an A/D frame's Buffer Address and Number of Samples elements.

Prototype

C

DASErr far pascal K_GetBuf(FRAMEH *frameHandle*, void far * far * *bufAddr*, long far * *samples*);

Pascal

Function K_GetBuf(*frameHandle* : Longint; Var *bufAddr* : Integer; Var *samples* : Longint) : Word;

Visual Basic for Windows

K_GetBuf Lib "ADC16.dll" (ByVal *frameHandle* As Long, *bufAddr* As Long, *samples* As Long) As Integer

Parameters

frameHandle	Frame handle
bufAddr	Buffer Address
samples	Number of Samples

Notes

On return, the following parameters contain the value of an element in the frame identified by *frameHandle*:

- *bufAddr* contains the value of the Buffer Address element
- *samples* contains the value of the Number of Samples element

K_GetChn

Purpose

Get the value of an A/D frame's Start Channel element.

Prototype

C

DASErr far pascal K_GetChn(FRAMEH *frameHandle*, short far * *chan*);

Pascal

Function K_GetChn(*frameHandle* : Longint; Var *chan* : Word) : Word;

Visual Basic for Windows

K_GetChn Lib "ADC16.dll" (ByVal *frameHandle* As Long, *chan* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
chan	Start Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.

Notes

On return, *chan* contains the value of the Start Channel element in the frame identified by *frameHandle*.

Purpose

Get the value of an A/D frame's Channel-Gain Array Address element.

Prototype

C

DASErr far pascal K_GetChnGAry(FRAMEH *frameHandle*,
void far * far * *chanGainArray*);

Pascal

Function K_GetChnGAry(*frameHandle* : Longint;
Var *chanGainArray* : Integer) : Word;

Visual Basic for Windows

K_GetChnGAry Lib "ADC16.dll" (ByVal *frameHandle* As Long,
chanGainArray As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
chanGainArray	Channel-Gain Array Address

Notes

On return, *chanGainArray* contains the value of the Channel-Gain Array Address element in the frame identified by *frameHandle*.

Refer to **K_SetChnGAry** for a description of Channel-Gain arrays.

K_GetDOCurVal

Purpose

Get the value of a digital output frame's Digital Output Value element.

Prototype

C

DASErr far pascal K_GetDOCurVal(FRAMEH *frameHandle*,
long far * *DOvalue*);

Pascal

Function K_GetDOCurVal(*frameHandle* : Longint;
Var *DOvalue* : Longint) : Word;

Visual Basic for Windows

K_GetDOCurVal Lib "ADC16.dll" (ByVal *frameHandle* As Long,
DOvalue As Long) As Integer

Parameters

frameHandle	Handle to digital output frame
DOvalue	Digital Output Value

Notes

On return, *DOvalue* contains the value of the Digital Output Value element in the frame identified by *frameHandle*. This value represents the value that was specified as the *DOvalue* parameter for the most recent call to **K_DOWrite**; it is not necessarily the value currently at the digital output port.

Purpose

Obtain the handle to a digital output frame.

Prototype

C

DASErr far pascal K_GetDOFrame(DDH *devHandle*,
FRAMEH far * *frameHandle*);

Pascal

Function K_GetDOFrame(*devHandle* : Longint;
Var *frameHandle* : Longint) : Word;

Visual Basic for Windows

K_GetDOFrame Lib "ADC16.dll" (ByVal *devHandle* As Long,
ByVal *frameHandle* As Long) As Integer

Parameters

devHandle	Device handle
frameHandle	Handle to digital output frame

Notes

On return, *frameHandle* contains the handle to a digital output frame associated with the device identified by *devHandle*.

Since the driver does not support frame-based digital output operations, **K_GetDOFrame** serves a very specific and limited purpose in ADC-16 Function Call Driver programs. **K_GetDOCurVal** requires the handle to a digital output frame as one of its calling arguments, and the only way to obtain a handle to a digital output frame is through **K_GetDOFrame**. Consequently, if you want to use **K_GetDOCurVal**, you must first call **K_GetDOFrame**; this is the only circumstance in which your program should call **K_GetDOFrame**.

K_GetErrMsg

Purpose

Get the address of an error message string. This function is available only as a C-language function.

Prototype

C

DASErr far pascal K_GetErrMsg(DDH *devHandle*, short *msgNum*, char far * far * *errMsg*);

Parameters

devHandle	Device handle
msgNum	Error message number
errMsg	Error message string

Notes

On return, *errMsg* contains a pointer to a string that corresponds to *msgNum* for the device identified by *devHandle.*

Refer to Appendix A for error numbers and error messages.

Purpose

Get the value of an A/D frame's Gain Code element.

Prototype

C

DASErr far pascal K_GetG(FRAMEH *frameHandle*, short far * *gainCode*);

Pascal

Function K_GetG(*frameHandle* : Longint; Var *gainCode* : Word) : Word;

Visual Basic for Windows

K_GetG Lib "ADC16.dll" (ByVal *frameHandle* As Long, *gainCode* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
gainCode	Gain Code. Valid values: **0** = 1x, **1** = 10x, 2 = 100x

Notes

On return, *gainCode* contains the value of the Gain Code element in the frame identified by *frameHandle*.

See page 31 for the A/D voltage ranges that correspond to each gain.

K_GetStartStopChn

Purpose

Get the values of an A/D frame's Start Channel and Stop Channel elements.

Prototype

C

DASErr far pascal K_GetStartStopChn(FRAMEH *frameHandle*,
short far * *start*, short far * *stop*);

Pascal

Function K_GetStartStopChn(*frameHandle* : Longint; Var *start* : Word;
Var *stop* : Word) : Word;

Visual Basic for Windows

K_GetStartStopChn Lib "ADC16.dll" (ByVal *frameHandle* As Long,
start As Integer, *stop* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
start	Start Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.
stop	Stop Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.

Notes

On return, the following parameters contain the value of an element in the frame identified by *frameHandle*:

- *start* contains the value of the Start Channel element
- *stop* contains the value of the Stop Channel element

Purpose

Get the values of an A/D frame's Start Channel, Stop Channel, and Gain Code elements.

Prototype

C

DASErr far pascal K_GetStartStopG(FRAMEH *frameHandle*, short far * *start*, short far * *stop*, short far * *gainCode*);

Pascal

Function K_GetStartStopG(*frameHandle* : Longint; Var *start* : Word; Var *stop* : Word; Var *gainCode* : Word) : Word;

Visual Basic for Windows

K_GetStartStopG Lib "ADC16.dll" (ByVal *frameHandle* As Long, *start* As Integer, *stop* As Integer, *gainCode* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
start	Start Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.
stop	Stop Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.
gainCode	Gain Code. Valid values: **0** = 1x, **1** = 10x, **2** = 100x

Notes

On return, the following parameters contain the value of an element in the frame identified by *frameHandle*:

- *start* contains the value of the Start Channel element
- *stop* contains the value of the Stop Channel element
- *gainCode* contains the value of the Gain Code element

See page 31 for the A/D voltage ranges that correspond to each gain.

K_GetVer

Purpose

Determine the driver revision and driver specification.

Prototype

C

DASErr far pascal K_GetVer(DDH *devHandle*, short far * *spec*,
short far * *version*);

Pascal

Function K_GetVer(*devHandle* : Longint; Var *spec* : Word;
Var *version* : Word) : Word;

Visual Basic for Windows

K_GetVer Lib "ADC16.dll" (ByVal *devHandle* As Long, *spec* As Integer,
version As Integer) As Integer

Parameters

devHandle	Device handle
spec	Driver specification
version	Driver version

Notes

On return, *spec* contains the revision number of the Keithley DAS Driver Specification to which the driver conforms; *version* contains the driver's version number.

spec and *version* are two-byte integers; the high byte contains the major revision level and the low byte contains the minor revision level (in the version number **2.1**, for example, the major and minor revision levels are **2** and **1**, respectively).

Use the following equations to extract the major and minor revision levels from the values returned in either *spec* and *version*:

$$\text{major revision level} = \frac{\textit{returned value}}{256}$$

$$\text{minor revision level} = \textit{returned value} \text{ MOD } 256$$

where *returned value* represents either *spec* or *version*.

Purpose

Initialize a board's A/D circuitry and set an A/D frame's elements to their default values.

Prototype

C

DASErr far pascal K_InitFrame(FRAMEH *frameHandle*);

Pascal

Function K_InitFrame(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_InitFrame Lib "ADC16.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_InitFrame initializes the A/D circuitry on the ADC-16 that is associated with the frame identified by *frameHandle.*

If an interrupt-mode A/D operation is not active, then **K_InitFrame** checks the validity of the board number associated with the frame identified by *frameHandle* and then enables A/D operations.

If an interrupt-mode A/D operation is active, then **K_InitFrame** returns an error that indicates that the board is busy.

K_IntAlloc

Purpose

Allocate a buffer suitable for an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntAlloc(FRAMEH *frameHandle*, DWORD *samples*, void far * far * *intAddr*, WORD far * *memHandle*);

Pascal

Function K_IntAlloc(*frameHandle* : Longint ; *samples* : LongInt; Var *intAddr* : Longint ; Var *memHandle* : Word) : Word;

Visual Basic for Windows

K_IntAlloc Lib "ADC16.dll" (ByVal *frameHandle* As Long, ByVal *samples* As Long, *intAddr* As Long, *memHandle* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
samples	Number of samples. Valid values: **0**, **1**,..., **32,767**
intAddr	Address of interrupt buffer
memHandle	Handle to interrupt buffer

Notes

On return, *intAddr* contains the address of a buffer that is suitable for an interrupt-mode A/D operation of *samples* samples; *memHandle* contains a handle to the buffer that this function allocates.

Purpose De-allocate an interrupt buffer that was previously allocated with **K_IntAlloc**.

Prototype

C
DASErr far pascal K_IntFree(WORD *memHandle*);

Pascal
Function K_IntFree(*memHandle* : Word) : Integer;

Visual Basic for Windows
K_IntFree Lib "ADC16.dll" (ByVal *memHandle* As Integer) As Integer

Parameters *memHandle* Handle to interrupt buffer

Notes **K_IntFree** de-allocates the interrupt buffer identified by *memHandle*.

K_IntStart

Purpose

Start an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntStart(FRAMEH *frameHandle*);

Pascal

Function K_IntStart(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_IntStart Lib "ADC16.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_IntStart starts the interrupt-mode A/D operation defined in the frame identified by *framehandle.*

See page 29 for a description of the format in which the driver stores the acquired values.

See page 13 for a discussion of the programming tasks associated with interrupt-mode A/D operations.

Purpose

Determine the status of an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntStatus(FRAMEH *frameHandle*, short far * *status*, long far * *samples*);

Pascal

Function K_IntStatus(*frameHandle* : Longint; Var *status* : Word; Var *samples* : Longint) : Word;

Visual Basic for Windows

K_IntStatus Lib "ADC16.dll" (ByVal *frameHandle* As Long, *status* As Integer, *samples* As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
status	Code that indicates status of interrupt operation. Valid values: **0** = Interrupt-mode A/D operation idle **1** = Interrupt-mode A/D operation active
samples	Number of samples already transferred to interrupt buffer

Notes

On return, *status* contains a code that indicates the status of the Interrupt operation defined by the frame identified by *frameHandle; samples* contains the number of samples already transferred to the Interrupt buffer at the time the function was called.

K_IntStop

Purpose

Abort an interrupt-mode A/D operation.

Prototype

C

DASErr far pascal K_IntStop(FRAMEH *frameHandle*, short far * *status*, long far * *samples*);

Pascal

Function K_IntStop(*frameHandle* : Longint; Var *status* : Word; Var *samples* : Longint) : Word;

Visual Basic for Windows

K_IntStop Lib "ADC16.dll" (ByVal *frameHandle* As Long, *status* As Integer, *samples* As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
status	Code that indicates status of interrupt operation. Valid values: **0** = Interrupt operation idle **1** = Interrupt operation active **2** = Data overrun (see note below)
samples	Number of samples already transferred to interrupt buffer

Notes

K_IntStop aborts the interrupt operation defined by the frame identified by *frameHandle*. On return, *status* contains a code that indicates what the status was when the function was called; *samples* contains the number of samples already transferred to the interrupt buffer when the function was called.

Data overrun occurs if data is lost when the transfer of data between the board and the PC's memory is slower than the rate at which the board is acquiring data.

K_IntStop does nothing if an interrupt-mode A/D operation is not in progress.

Purpose

Transfer acquired A/D samples between a memory buffer and an array.

Prototype

C

DASErr far pascal K_MoveDataBuf(int far * *dest*, int far * *source*, unsigned int *samples*);

Pascal

Function K_MoveDataBuf(*dest* : Longint; *source* : Longint; *samples* : Word) : Integer;

Visual Basic for Windows

K_MoveDataBuf Lib "ADC16.dll" (*dest* As Any, *source* As Any, ByVal *samples* As Integer) As Integer

Parameters

dest	Address of destination buffer
source	Address of source buffer
samples	Number of samples to transfer

Notes

K_MoveDataBuf moves *samples* samples from the buffer at *source* to the buffer at *dest*.

Although this function is valid for all of the supported languages, it is intended primarily for use with those languages (such as Visual Basic) that do not provide a convenient method to access memory directly. This function is not needed in languages (such as C) that provide access to memory buffers through pointers.

K_SetBuf

Purpose

Set the values of an A/D frame's Buffer Address and Number of Samples elements.

Prototype

C

DASErr far pascal K_SetBuf(FRAMEH *frameHandle*, void far * *bufAddr*, long *samples*);

Pascal

Function K_SetBuf(*frameHandle* : Longint; *bufAddr* : Longint; *samples* : Longint) : Word;

Visual Basic for Windows

K_SetBuf Lib "ADC16.dll" (ByVal *frameHandle* As Long, *bufAddr* As Any, ByVal *samples* As Long) As Integer

Parameters

frameHandle	Handle to A/D frame
bufAddr	Buffer Address
samples	Number of Samples

Notes

K_SetBuf assigns values to the following elements in the frame identified by *frameHandle*:

- the Buffer Address element is assigned the value in *bufAddr*
- the Number of Samples element is assigned the value in *samples*

Purpose

Set the value of an A/D frame's Start Channel element.

Prototype

C

DASErr far pascal K_SetChn(FRAMEH *frameHandle*, short *chan*);

Pascal

Function K_SetChn(*frameHandle* : Longint; *chan* : Word) : Word;

Visual Basic for Windows

K_SetChn Lib "ADC16.dll" (ByVal *frameHandle* As Long,
ByVal *chan* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
chan	Start Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.

Notes

K_SetChn sets the value of the Start Channel element to *chan* in the frame identified by *frameHandle*.

K_SetChnGAry

Purpose

Set the value of a frame's Channel-Gain Array Address element.

Prototype

C

DASErr far pascal K_SetChnGAry(FRAMEH *frameHandle*,
void far * *chanGainArray*);

Pascal

Function K_SetChnGAry(*frameHandle* : Longint;
Var *chanGainArray* : Integer) : Word;

Visual Basic for Windows

K_SetChnGAry Lib "ADC16.dll" (ByVal *frameHandle* As Long,
chanGainArray As Integer) As Integer

Parameters

frameHandle Handle to A/D frame

chanGainArray Channel-Gain Array Address

Notes

K_SetChnGAry sets the value of the Channel-Gain Array Address element to *chanGainArray* in the frame identified by frameHandle.

A Channel-Gain Array defines two characteristics of an A/D operation:

- the sequence in which the input channels are sampled and,
- the gain applied to each channel in that sequence.

A Channel-Gain Array can define up to 256 randomly sequenced channel-gain pairs. Adjacent pairs can specify the same channel (with equal or unequal gains). The figure below illustrates the required format of a channel gain array.

Byte	0	1	2	3	4	5	...	2N-1	2N
Value	N		chan	gain	chan	gain	...	chan	gain
	# of pairs		pair 1		pair 2		...	pair N	

The gain must be specified as a gain code. Refer to **K_SetStartStopG** on page 63 for valid gain codes and channel numbers.

Purpose

Set the value of an A/D frame's Gain Code element.

Prototype

C

DASErr far pascal K_SetG(FRAMEH *frameHandle*, short *gainCode*);

Pascal

Function K_SetG(*frameHandle* : Longint; *gainCode* : Word) : Word;

Visual Basic for Windows

K_SetG Lib "ADC16.dll" (ByVal *frameHandle* As Long,
ByVal *gainCode* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
gainCode	Gain Code. Valid values: **0** = 1x, **1** = 10x, **2** = 100x

Notes

K_SetG sets the Gain Code element to *gainCode* in the frame identified by *frameHandle*.

See page 31 for the A/D voltage ranges that correspond to each gain.

K_SetStartStopChn

Purpose

Set the values of an A/D frame's Start Channel and Stop Channel elements.

Prototype

C

DASErr far pascal K_SetStartStopChn(FRAMEH *frameHandle*, short *start*, short *stop*);

Pascal

Function K_SetStartStopChn(*frameHandle* : Longint; *start* : Word; *stop* : Word) : Word;

Visual Basic for Windows

K_SetStartStopChn Lib "ADC16.dll" (ByVal *frameHandle* As Long, ByVal *start* As Integer, ByVal *stop* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
start	Start Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.
stop	Stop Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.

Notes

K_SetStartStopChn assigns values to the following elements in the frame identified by *frameHandle*:

- the Start Channel element is assigned the value in *start*
- the Stop Channel element is assigned the value in *stop*

Use **K_SetChnGAry** to specify a non-sequential channel-scanning sequence.

Purpose

Set the values of an A/D frame's Start Channel, Stop Channel, and Gain Code elements.

Prototype

C

DASErr far pascal K_SetStartStopG(FRAMEH *frameHandle*, short *start*, short *stop*, short *gainCode*);

Pascal

Function K_SetStartStopG(*frameHandle* : Longint; *start* : Word; *stop* : Word; *gainCode* : Word) : Word;

Visual Basic for Windows

K_SetStartStopG Lib "ADC16.dll" (ByVal *frameHandle* As Long, ByVal *start* As Integer, ByVal *stop* As Integer, ByVal *gainCode* As Integer) As Integer

Parameters

frameHandle	Handle to A/D frame
start	Start Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.
stop	Stop Channel. Valid values: **0**, **1**,...,**7(m+1)**, where **m** is the number of connected STA-EX8.
gainCode	Gain Code. Valid values: **0** = 1x, **1** = 10x, 2 = 100x

Notes

K_SetStartStopG assigns values to the following elements in the frame identified by *frameHandle*:

- the Start Channel element is assigned the value in *start*
- the Stop Channel element is assigned the value in *stop*
- the Gain Code element is assigned the value in *gainCode*

Use **K_SetChnGAry** to specify different gains for different channels or to specify an un-ordered channel-scanning sequence.

See page 31 for the A/D voltage ranges that correspond to each gain.

K_SyncStart

Purpose

Start a synchronous-mode A/D operation.

Prototype

C

DASErr far pascal K_SyncStart(FRAMEH *frameHandle*);

Pascal

Function K_SyncStart(*frameHandle* : Longint) : Word;

Visual Basic for Windows

K_SyncStart Lib "ADC16.dll" (ByVal *frameHandle* As Long) As Integer

Parameters

frameHandle Handle to A/D frame

Notes

K_SyncStart starts the synchronous-mode A/D operation defined in the frame identified by *framehandle.*

See page 29 for a description of the format in which the driver stores the acquired values.

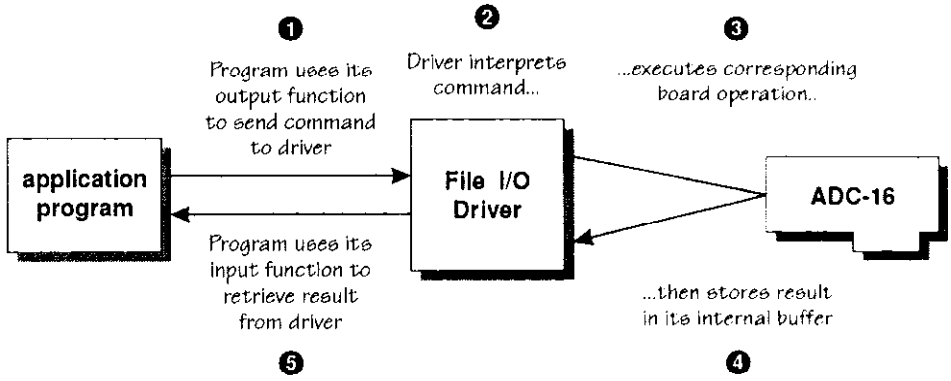
See page 15 for a discussion of the programming tasks associated with synchronous-mode A/D operations.

File I/O Driver 4

4.1 Overview

The File I/O Driver serves as an interface between your application program and the board's acquisition & control operations. The driver has its own set of File I/O Commands. Each of these English-like commands corresponds to a board operation. Your program can use these commands to perform a variety of acquisition & control operations.

The driver acts like a file device; consequently, your program can use its own file I/O functions (for example, INPUT and PRINT if you are programming in BASIC) to communicate with the driver. To execute a board operation, your program outputs a File I/O Command to the driver. The driver interprets the command, executes the corresponding operation, and stores the result in its internal buffer. Your program can then input this result from the driver.



Driver components

The File I/O Driver consists of two components: the driver program (MADC16.EXE) and one of the Virtual Instrument programs (VI.EXE or VITASK.EXE). You can use either of the Virtual Instrument programs. These two programs differ in the amount of memory each uses and in their ability to provide access to the Pop Up Control Panel (refer to the *ADC-16 User's Guide* for a complete description of the Pop Up Control Panel).

VI.EXE

VI.EXE uses approximately 51 K of RAM. If your program requires access to the Pop Up Control Panel, you must load VI.EXE.

VITASK.EXE

VITASK.EXE uses approximately 21 K of RAM. If your program does not require access to the Pop Up Control Panel, you can load either VITASK.EXE or VI.EXE.

4.2 Loading and unloading the driver

As described in the previous section, the driver consists of the driver program (MADC16.EXE) and one of the Virtual Instrument programs (VI.EXE or VITASK.EXE). The order in which you load these programs is significant.

To load the driver, load the driver programs in the following order:
VI.EXE or VITASK.EXE
MADC16.EXE.

To unload the driver, unload the driver programs in the following order:
MADC16.EXE
VI.EXE or VITASK.EXE.

To load or unload the driver, you must execute two separate DOS command lines (one for either VI.EXE or VITASK.EXE, one for MADC16.EXE). There are two ways to execute these command lines:

- You can enter the command lines at the DOS prompt, or
- You can create a batch file that contains the command lines and then run the batch file.

In either case, make sure that you execute the commands in the correct order.

Command line syntax

The command line syntax descriptions presented in this section use the following typographic conventions:

- [ ] – Entries enclosed between square brackets are mandatory. Do not include the brackets in the command line.
- { } – Entries enclosed between curly brackets are optional. To include the optional entry in the command line, specify only what is between the brackets (do not include the brackets in the command line).
- ( ) – Entries enclosed in parentheses represent the valid values for a command argument. The valid values are separated by commas. Specify only one of the valid values from the group (do not include the parentheses in the command line).
- The case of the letters in an entry is not significant; entries can be specified in uppercase, lowercase, or mixed case.
- Entries shown in **boldface** type must be specified exactly as shown (except for case).
- Entries shown in *italic* describe the type of entry that should be specified. For example, if the entry is given as *filename*, then the entry you specify must be a valid filename (TEST.DAT, for example).

VI syntax

{*drive*[:]}{*path*}**VI** {[**mono**]} {[**/HK**=*key*]} {[**/MK**=*key*]} {[**/SK**=*key*]} {[**/U**]}

[**mono**]
Specifies that VI will run in Monochromatic mode. If **mono** is not specified, VI will assume that it is running on a color monitor.

[**/HK**=*key*]
(Help Key) Specifies the key that invokes the Pop Up Control Panel Help screen. *key* must be one of the following:

A, **B**, ..., **Z**
0, **1**, ..., **9**
F1, **F2**, ..., **F10**
Tab, **Esc**, or **?**

or any of the above preceded by **Ctrl** or **Ctrl Alt**.

examples:
/HK=F2 specifies [F2] as the Help Key
/HK=Alt Tab specifies [Alt] – [Tab] as the Help Key

[**/MK**=*key*]
(Mode Select Key) Specifies the key that switches the Pop Up Control Panel to Keyboard Control Mode. See explanation of [**/HK**=] above for valid *key* values.

[**/SK**=*key*]
(Instrument Select Key) Specifies the key that cycles through multiple ADC-16 boards. See explanation of [**/HK**=] above for valid *key* values.

[**/U**]
Unloads VI from memory.

VITASK syntax

{*drive*[:]}{*path*}**VITASK** {[**/U**]}

[**/U**]
Unloads VI from memory.

MADC16 syntax

{*drive*[:]}{*path*}**MADC16** {**/F**=*cfgFile*} {**/PK**=*key*} {**/Name**=*boardName*} {**/U**}

{**/F**=*cfgFile*}
Specifies the board configuration file.

{**/PK**=*key*}
(Pop Up Control Panel Key) Specifies the key that invokes the Pop Up Control Panel. See the description of [**/HK**=] under VI syntax for the valid *key* values. The default Pop Up Control Panel Key is **Alt F6**.

{**/Name**=*boardName*}
(Board Name) Assigns a user-specified name to the board that is at the address specified in the driver configuration file. You must use **/Name**= when you have two boards installed and you want to simultaneously display the Pop Up Control Panel for both of them. *boardName* must contain one to eight characters; any character that is valid for a DOS file name can be used.

{**/U**}
Unloads MADC16 from memory.

Example

Suppose the following conditions exist:

- your program needs access to the Pop Up Control Panel
- you want the driver configured according to the information in a configuration file named CUSTOM.CFG
- VI.EXE, MADC16.EXE, and CUSTOM.CFG are in the C:\ADC16 directory

The following command lines load the File I/O Driver appropriately for the conditions listed above:

```
C:\ADC16\VI
C:\ADC16\MADC16 /F=C:\ADC16\CUSTOM.CFG
```

4.3 Language-specific programming notes

This section provides specific programming guidelines for each of the supported languages. Additional programming information is available in the ASO example programs. Refer to the FILES.DOC file for names and descriptions of the ASO example programs.

Borland Turbo C

Supported versions

2.0 and higher

Opening the driver

The code listed below shows the correct procedure to open the driver and clear its internal buffer. This code references the PrintError error handler defined under **Trapping Errors**.

```
/* Open Driver for reading and writing */
   ADC16 = fopen( "$ADC16","r+");
/* Check for Error */
   if (ADC16 == NULL) PrintError();
/* Clear the driver's internal buffer */
   fprintf( ADC16, "Clear" );
   fflush (ADC16);
   if (errno !=0) return(1);
```

Sending commands/ Retrieving results

The following notes provide general guidelines for sending commands and retrieving results with Borland Turbo C:

- Use fprintf( ) to send commands to the driver.
- Use fgets( ) to retrieve results from the driver.
- Call rewind( ) between successive calls to fprintf( ) and fputs( ).
- Call fclose( ) and freopen( ) between successive calls to fgets( ) and fprintf( ).
- Call fflush( ) after an fprintf( ) to insure that the command sent by fprintf( ) is flushed from the DOS buffer.

The following code demonstrates how to send a command and retrieve the results:

```
/* Repeat until Status=0 (DONE) */
do
   {
   /* Check log status */
   fprintf( ADC16, "Read Logstat" );
   fflush (ADC16);
   /* If error print it, then exit with error */
   if (errno != 0)
   {
      PrintError();
      exit(1)
   }
   /* Rewind required between successive input and output */
   rewind(ADC16);
   /* If error on read then exit */
   if (!fgets (Str, 80, ADC16)) exit(1);
   /* Convert data to integer */
   sscanf( Str,"%d",Status )
}
while( Status != 0 );
```

Trapping errors

The following code defines an error handler:

```
void PrintError()
{
/* Rewind required between successive input and output */
   rewind(ADC16);
/* Get error number */
   if (!fgets (Str, 80, ADC16))
/* Convert data to integer */
   sscanf( Str,"%d",ErrNum )
/* Get error number */
   if (!fgets (Str, 80, ADC16))
/* Convert data to integer */
   sscanf( Str,"%s",ErrStr1 )
/* Get error number */
   if (!fgets (Str, 80, ADC16))
/* Convert data to integer */
   sscanf( Str,"%s",ErrStr2 )
/* Print error results */
   printf( "Error Number \n%x ", ErrNum);
   printf( "\nError About %s", ErrStr1 );
   printf( "\nTotal Line %s",  ErrStr2 );
}
```

Microsoft C

Supported versions

4.0 and higher

Opening the driver

The code listed below shows the correct procedure to open the driver and clear its internal buffer. This code references the PrintError error handler defined under **Trapping Errors.**

```
/* Open Driver for reading and writing */
   ADC16 = fopen( "$ADC16","r+");
/* Check for Error */
   if (ADC16 == NULL) PrintError();
/* Clear the driver's internal buffer */
   fprintf( ADC16, "Clear" );
   if (fflush (ADC16) == EOF) return(1);
```

Sending commands/ Retrieving results

The following notes provide general guidelines for sending commands and retrieving results with Microsoft C:

- Use fprintf( ) to send commands to the driver.
- Use fgets( ) to retrieve results from the driver.
- Call rewind( ) between successive calls to fprintf( ) and fputs( ) and between successive calls to fputs( ) and fprintf( ).
- Call fflush( ) after fprintf( ) to insure that the command sent by fprintf( ) is flushed from the DOS buffer.

The following code demonstrates how to send a command and retrieve the results:

```
/* Repeat until Status=0 (DONE) */
do
   {
   /* Check log Status */
   fprintf( ADC16, "Read Logstat" );
   /* If error print it, then exit with error */
   if (fflush (ADC16) == EOF)
   {
   PrintError();
   exit(1)
   }
   /* Rewind required between successive input and output */
   rewind(ADC16);
   /* If error on read then exit */
   if (!fgets (Str, 80, ADC16)) exit(1);
/* Convert data to integer */
   sscanf( Str,"%d",Status )
}
while( Status != 0 );
```

Trapping errors

The following code defines an error handler:

```
void  PrintError()
{
/* Rewind required between successive input and output */
   rewind(ADC16);
/* Get error number */
   if (!fgets (Str, 80, ADC16))
/* Convert data to integer */
   sscanf( Str,"%d",ErrNum )
/* Get error number */
   if (!fgets (Str, 80, ADC16))
/* Convert data to integer */
   sscanf( Str,"%s",ErrStr1 )
/* Get error number */
   if (!fgets (Str, 80, ADC16))
/* Convert data to integer */
   sscanf( Str,"%s",ErrStr2 )
/* Print error results */
   printf( "Error Number \n%x ", ErrNum);
   printf( "\nError About %s", ErrStr1 );
   printf( "\nTotal Line %s",  ErrStr2 );
}
```

Borland Turbo Pascal

Supported versions

4.0 and higher

Opening the driver

The code listed below shows the correct procedure to open the driver and clear its internal buffer. This code references the GetError error handler defined under **Trapping Errors**.

```
(* Main *)
   BEGIN
   Assign(ADC16IN, '$ADC16');
   Assign(ADC16OUT, '$ADC16');
(* Input, PASCAL has no read/write text files *)
   Reset(ADC16IN);
(* Output, PASCAL has no read/write text files *)
   Rewrite(ADC16OUT);
```

Sending commands/ Retrieving results

The following notes provide general guidelines for sending commands and retrieving results with Borland Turbo Pascal:

- Use Writeln( ) to send commands to the driver.
- Use Readln( ) to retrieve results from the driver.
- All strings used for retrieving data from the driver must be declared as STRING[255].

The following code demonstrates how to send a command and retrieve the results:

```
   Status := 1;
(* Wait for status to be DONE *)
   WHILE Status <> 0 DO
   BEGIN
   writeln(ADC16OUT,'Read Logstat');
   IF (IOResult <> 0) THEN  GetError ;
(* Status was declared as integer *)
   ReadLn(ADC16IN,Status);
   END
```

Trapping errors

The following code defines an error handler:

```
PROCEDURE GetError ;
   BEGIN
   readln(ADC16IN,ErrNum);
   readln(ADC16IN,AStr);
   readln(ADC16IN,BStr);
   writeln('Driver Error Has Occurred !!');
   writeln('MADC16 Error Number => ',ErrNum);
   writeln('Error => ',BStr);
   writeln('On Command Line of => ',Astr);
   Halt(1)
   END;
```

Microsoft Pascal

Supported versions

3.0 and higher

Opening the driver

Microsoft Pascal programs communicate with the driver via a file handle of the Pascal type TEXT. This type of file handle allows files to simultaneously be open for input and output. Consequently, only one file handle is required and should be ASSIGNed for both input and output.

The code listed below shows the correct procedure to open the driver and clear its internal buffer. This code references the GetError error handler defined under **Trapping Errors.**

```
(* Main *)
   BEGIN
(* Open device driver for I/O random access *)
   Assign(ADC16 ,'$ADC16');
(* Direct Mode insures flush after WriteLn *)
   ADC16.MODE := DIRECT;
(* Rewrite opens and rewinds the file *)
   Rewrite(ADC16);
```

Sending commands/ Retrieving results

The following notes provide general guidelines for sending commands and retrieving results with Microsoft Pascal:

- Use Writeln( ) to send commands to the driver.
- Use Readln( ) to retrieve results from the driver.
- All strings used for retrieving data from the driver must be declared as STRING[255].

The following code demonstrates how to send a command and retrieve the results:

```
   Status := 1;
(* Wait for status to be DONE *)
   WHILE status <> 0 DO
   BEGIN
(* Rewinds file and flushes previous contents *)
   Seek(ADC16,1);
(* Clear I/O error flag before all file ops *)
   ADC16.ERRS := 0 ;
(* Trap Errors Instead of Exit To Dos *)
   ADC16.TRAP := TRUE ;
   writeln(ADC16,'Read Logstat');
   Seek(ADC16,1);
   IF (ADC16.ERRS <> 0) THEN
   GetError ;
(* Rewrite opens and rewinds the file *)
   Rewrite(ADC16);
   ADC16.ERRS := 0 ;
   ADC16.TRAP := TRUE ;
   Seek(ADC16,1) ;
(* Status was declared as integer *)
   ReadLn(ADC16,Status);
   Rewrite(ADC16)
   END
```

Trapping errors

The following code defines an error handler:

```
PROCEDURE GetError;
BEGIN
(* Rewinds file and flushes previous contents *)
   Seek(ADC16,1);
   ADC16.ERRS := 0 ;
(* Clear I/O Error Flag Before All File Ops. *)
(* Trap errors instead of exit To DOS *)
   ADC16.TRAP := TRUE;
(* Read error number string from driver *)
   readln(ADC16,EN);
(* Read original command line from driver *)
   readln(ADC16,AString);
(* Read Error Description From Driver *)
   readln(ADC16,BString);
   writeln(chr(7));  (* BELL *)
   writeln('Driver Error Has Occurred !!');
   writeln('ADC-16 Error Number => ',EN);
   writeln('Error => ',BString);
   writeln ('On Command Line => ',Astring);
   Abort('Program terminated due to error. . .',0,0)
```

Interpreted BASIC

Supported versions

All

Opening the driver

The following code shows the correct procedure to open the driver and clear its internal buffer:

```
150  ' Give Line number to goto if an error occurs
200  ON ERROR GOTO 5000
250  ' Establish File Token #1 with $ADC16 for output
260  ' All commands will be output using Token #1
300  OPEN "$ADC16" FOR OUTPUT AS #1
350  ' Clear ADC16 File I/O return buffer
400  PRINT #1, "CLEAR"
450  ' Establish File Token #2 with $ADC16 for input
460  ' All inputs will be read using Token #2
500  OPEN "$ADC16" FOR INPUT AS #2
```

Sending commands/ Retrieving results

The following notes provide general guidelines for sending commands and retrieving results with Interpreted BASIC:

- Use PRINT to send commands to the driver.
- Use INPUT to retrieve results from the driver.

The following code demonstrates how to send a command and retrieve the results:

```
1000  PRINT #1,"READ LOGSTAT"          ' Send command which will
                                       ' fill Device's Return Buffer
                                       ' with Status.
1010  INPUT #2, ST$                    ' Read status into string.
1020  IF VAL(ST$)<>0 goto 1010         ' If the Value of Status is
                                       ' not zero, then AD is busy.
                                       ' Wait for status to be done.
```

Trapping errors

The following code defines an error handler:

```
                                       ' START OF ERROR HANDLER
5000  Beep                             ' Signal error.
5010  IF ERR=75 GOTO 5060              ' GWBASIC may return
5020  IF ERR=68 GOTO 5060              ' 75, 68, or 57 for
5030  IF ERR=57 GOTO 5060              ' a SYNTAX error.
5040  IF ERR = 62 GOTO 5130            ' Error 62 is an attempt
                                       ' to read from a device that
                                       ' has no data to read.
5050  Print ERR : RESUME               ' If none of the above, then
                                       ' error is not from driver.
5060  INPUT #2,EN                      ' Read driver error #.
5070  LINE INPUT #2,A$                 ' Read part of line that
                                       ' contained error.
5080  LINE INPUT #2, B                 ' Read entire line as
                                       ' received.
5090  PRINT "Error number ";EN         ' Print info received from
                                       ' from syntax error.
5100  PRINT "ERROR - "B$
5110  PRINT "On command line of ";A$
5120  STOP                             ' Stop execution
5130  PRINT "Data Not Available"       ' Print error 62 message
5140  STOP
```

QuickBASIC

Supported versions

All

Opening the driver

The following code shows the correct procedure to open the driver and clear its internal buffer:

```
ON ERROR GOTO ErrHandler            ' Give line number to goto
                                    ' if an error occurs.
OPEN "$ADC16" FOR OUTPUT AS #1      ' Establish file token #1
                                    ' with $ADC16 for output.
                                    ' All commands will be output
                                    ' using token #1.
PRINT #1, "CLEAR"                   ' Clear ADC16 file I/O
                                    ' return buffer.
OPEN "$ADC16" FOR INPUT AS #2       ' Establish file token #2
                                    ' with $ADC16 for input.
                                    ' All inputs will be read
                                    ' using token #2.
```

Sending commands/ Retrieving results

The following notes provide general guidelines for sending commands and retrieving results with Quick BASIC:

- Use PRINT to send commands to the driver.
- Use INPUT to retrieve results from the driver.

The following code demonstrates how to send a command and retrieve the results:

```
WaitForDone:
PRINT #1,"READ AD STATUS"           ' Send command which will
                                    ' fill device return buffer
                                    ' with Status.
INPUT #2, ST$                       ' Read Status into string.
IF VAL(ST$)<>0 goto WaitForDone     ' If the value of status
                                    ' is not zero then AD
                                    ' is busy; wait for Status
                                    ' to be done.
```

Trapping errors

The following code defines an error handler:

```
ErrHandler:
   Beep                                  ' Signal error.
   IF ERR=75 GOTO SyntaxError            ' QuickBASIC may return
   IF ERR=68 GOTO SyntaxError            ' 75, 58, or 67 for
   IF ERR=57 GOTO SyntaxError            ' a SYNTAX error.
   IF ERR = 62 GOTO DataOutError         ' Error 62 is an attempt to
                                         ' read from a device that has
                                         ' no data to read.
   Print ERR : RESUME                    ' If none of the above, then
                                         ' error is not from driver.

SyntaxError:
   INPUT #2,EN                           ' Read driver error #.
   LINE INPUT #2,A$                      ' Read part of line that
                                         ' contained error.
   LINE INPUT #2, B$                     ' Read entire line as
                                         ' received.
   PRINT "Error number ";EN              ' Print info received from
                                         ' from syntax error.
   PRINT "ERROR - "B$
   PRINT "On command line of ";A$
   STOP                                  ' Stop execution.

DataOutError:
   PRINT "Data Not Available"            ' Print error 62 message.
   STOP
```

File I/O Commands 5

5.1 Functional grouping

The File I/O Commands can be logically grouped according to the functionality that each provides. This section lists each command as a member of one of the following groups:

- Setup and Initialization
- A/D operations
- Pop Up Control Panel

Setup and Initialization

Clear
Clears all data that the driver has prepared for your program's next input operation.

A/D operations

ADStart
Enables A/D acquisition.

ADStop
Disables A/D acquisition.

Read ADType
Returns the A/D transfer mode in which the next A/D operation will execute.

A/D operations (cont'd)

Read Channel
Returns the A/D value acquired on a specified channel.

Read Gain
Returns a code that indicates the current global gain.

Read Level
Returns the current interrupt level.

Read {Mode/LogFile/Date/Block/LogStat}
Returns the conditions that define the next **StartLog**.

Read Range
Returns the current full-scale A/D range.

Read Startchannel
Returns the channel number of the first channel in the current channel scan.

Read Stopchannel
Returns the channel number of the last channel in the current channel scan.

Set ADType
Specifies the A/D transfer mode in which the next A/D operation will execute.

Set Gain
Sets the global gain.

Set Level
Sets the interrupt level to be used for the next A/D operation.

Set {Mode/LogFile/Date/Block/LogStat}
Sets conditions that define the next **StartLog**.

A/D operations (cont'd)

Set Startchannel
Specifies the first channel that will be scanned during the next A/D operation.

Set Stopchannel
Specifies the last channel that will be scanned during the next A/D operation.

StartLog
Writes the current A/D data into a file.

StopLog
Stops current logging operation.

Pop Up Control Panel

Hide
Hides the Pop Up Control Panel.

Lock
Disables keyboard and mouse control of the Pop Up Control Panel.

Read Units
Returns the units that the Pop Up Control Panel will use to display data.

Set Units
Sets the units that the Pop Up Control Panel will use to display data.

Show
Causes the display of a specified panel of the Pop Up Control Panel.

Unlock
Enables keyboard and mouse control of the Pop Up Control Panel.

5.2 Command reference

The following notes describe the conventions and standard terminology used in the remainder of this chapter:

About Syntax entries

- The Syntax heading for each command lists two lines. The first line is the standard form of the command. The second line is the abbreviated form of the command. The abbreviated form shows the minimum characters in each keyword that must be present in order for the driver to recognize the command. The driver recognizes both forms; the abbreviated form is provided as a convenience.

- **Se** is shown as the minimum abbreviation for **Set** keyword. However, **Set** (or its abbreviation **Se**) can be omitted from any command whose standard form includes the **Set** keyword. For example, **Set ADType** can be specified as **ADType**.

- { } – Curly brackets enclose a set of command keywords from which one must be selected to define the command; keywords are separated by a backslash. For example, **Read {Mode/LogFile/Date/Block/LogStat}** represents five commands: **Read Mode**, **Read LogFile**, **Read Date**, **Read Block**, and **Read LogStat**.

- ( ) – Entries enclosed in parentheses are *constant* arguments (see note about *variable* arguments below). The constant arguments are separated by commas. The constant arguments must be specified exactly as they are shown. For example, **Set Units (ADcodes,Volts)** indicates that the **Set Units** command takes a single argument, and that argument must be either **ADcodes** or **Volts**.

- Variable arguments are shown in *italic* and describe the type of value that should be specified. For example, *filename* indicates that the argument should represent a valid filename.

Format of returned values

"Returns" means that the driver executes the command and stores the result in its internal buffer. Your program can retrieve this result from the driver by using one of your programming language's input functions.

All of these results are returned as ASCII text strings. Many of these text strings, however, represent decimal integers. You should write your program so that it interprets each result appropriately.

Gain Codes and A/D input ranges

The gain and the A/D full-scale range determine the A/D input range (the A/D full scale range is specified by the driver configuration file). The table shown below lists the A/D input range that corresponds to each gain/full-scale range combination.

gain	A/D input range for ±3.2767 V full-scale range	A/D input range for ±5.0 V full-scale range
1	±3.2767 V	±5 V
10	±327.67 mV	±500 mV
100	±32.767 mV	±50 mV

ADStart

Syntax

ADStart
ADStart

Description

Enables A/D acquisition.

ADStop

Syntax

ADStop
ADStop

Description

Disables A/D acquisition.

Clear

Syntax

Clear
Cl

Description

Clears all data that the driver has prepared for your program's next input operation.

Notes

Since some versions of DOS do not call the driver when your program issues an **Open**, the driver might contain input-ready data that was prepared by a previous program. Consequently, you should issue a **Clear** immediately following any **Open**.

Hide

Syntax

Hide
Hi

Description

Hides the Pop Up Control Panel.

Notes

Hide is ignored if VITASK (instead of VI) was loaded immediately before MADC16 was loaded; refer to page 66 for a description of the differences between VI and VITASK.

Show cancels the effect of **Hide**.

Lock

Syntax

Lock
Lo

Description

Disables keyboard and mouse control of the Pop Up Control Panel.

Notes

Use **Unlock** to cancel the effect of **Lock**.

Read ADType

Syntax

Read ADType
Re ADT

Description

Returns a code that indicates the current A/D transfer mode.

Return codes

0 = Synchronous-mode
1 = Interrupt-mode

Notes

The current A/D transfer mode is the mode specified by the most recently issued **Set ADType**.

Read Channel

Syntax

Read Channel *channel*
Re Ch *channel*

Description

Returns the A/D value acquired on the channel specified by *channel*. The valid values for *channel* are **0**, **1**,...,**7**.

Notes

The current units and the current gain (as specified by the most recently issued **Set Units** and **Set Gain**, respectively) determine the implied units of the returned value as follows:

- If the current units = Volts and the current gain = 1, then the value is returned in units of Volts.
- If the current units = Volts and the current gain = 10 or 100, then the value is returned in units of MilliVolts.

Read Gain

Syntax

Read Gain
Re Ga

Description

Returns the current global gain.

Return codes

1, **10**, **100**. Refer to the table on page 85 for the A/D input ranges that correspond to each of these gains.

Read Level

Syntax

Read Level
Re Lev

Description

Returns the current interrupt level.

Notes

The current interrupt level is defined by the value of the *level* argument specified in the most recently issued **Set Level**.

Read {Mode/LogFile/Date/Block/LogStat}

Syntax

Read Mode
Re Mo

Read LogFile
Re Logfile

Read Date
Re Da

Read Block
Re Bl

Read LogStat
Re Logstat

Description

Read Mode
Returns the value of the *mode* argument specified in the most recently issued **Set Mode**. The value of *mode* represents the mode (New, Append or Overwrite) in which the data will be written to the data file by the next **StartLog**.

Read LogFile
Returns the *filename* argument specified in the most recently issued **Set LogFile**. *filename* represents the name of the data file that will be used by the next **StartLog**.

Read Date
Returns the *date* argument specified in the most recently issued **Set Date**. *date* indicates if date stamping is enabled.

Read Block
Returns the *block* argument specified in the most recently issued **Set Block**. *block* represents the number of data blocks that will be logged by the next **Start Log**.

Read LogStat
Returns the current log status.

Return codes

Read Mode
0 = New
1 = Overwrite
2 = Append

Read Date
0 = Date stamping Off
1 = Date stamping On

Read LogStat
0 = Logging Off
1 = Logging On

Read Range

Syntax

Read Range
Re Ran

Description

Returns a code that indicates the current A/D full-scale range.

Return codes

0 = ±3.2768 V
1 = ±5.0 V

Read Startchannel

Syntax

Read Startchannel
Re Sta

Description

Returns the channel number of the first channel in the current channel scan.

Notes

The first channel in the current scan is defined by the most recently issued **Set Startchannel**.

Read Stopchannel

Syntax

Read Stopchannel
Re Sto

Description

Returns the channel number of the last channel in the current channel scan.

Notes

The last channel in the current scan is defined by the most recently issued **Set Stopchannel**.

Read Units

Syntax

Read Units
Re Un

Description

(Applies only if the Pop Up Control Panel is visible). Returns a code that indicates the current display units on the Pop Up Control Panel.

Return codes

0 = A/D codes
1 = Volts

Set ADType

Syntax

Set ADType (Interrupt,Synchronous)
Se ADT (Int,Syn)

Description

Specifies the A/D transfer mode in which the next A/D operation will execute.

Notes

Interrupt mode allows for the acquisition and transfer of data using the interrupt level set by the most recently issued **Set Level**. The driver detects the interrupt that the ADC-16 issues at the conclusion of a conversion and then reads the acquired data into memory. Because of the driver and CPU involvement required to read the data into memory, the maximum conversion rate in interrupt mode is limited to approximately 5 Khz.

Operations that execute in interrupt mode must use a single gain for each channel in the scan. Interrupt-mode operations execute entirely in the background.

Synchronous mode operates in the foreground. When an AD command begins executing, no other board functions are available until the A/D operation terminates. The maximum conversion throughput available in synchronous mode is machine dependent.

Set Gain

Syntax

Set Gain *gain*
Se Ga *gain*

Description

Specifies *gain* as the current global gain.

Arguments

channel
0, 1,...,7

gain
1, 10, 100

Set Level

Syntax

Set Level (2,3,4,5,7,10,11,15)
Se Lev (2,3,4,5,7,10,11,15)

Description

Sets the interrupt level to be used for the next A/D operation.

Set {Mode/LogFile/Date/Block/LogRate}

Syntax

Set Mode (Append,New,Overwrite)
Se Mo (Ap,Ne,Ov)

Set LogFile *fileName*
Se Logfile *fileName*

Set Date (On,Off)
Se Da (On,Off)

Set Block *numBlocks*
Se Bl *numBlocks*

Set LogRate *rate*
Se Lograte *rate*

Description

Set Mode (Append,New,Overwrite)
Specifies the mode in which the data will be written to the file by the next **StartLog**.

Set LogFile *fileName*
Specifies *fileName* as the name of file that will be used by the next **StartLog**.

Set Date (On,Off)
Specifies if date stamping is enabled.

Set Block *numBlocks*
Specifies *numBlocks* as the number of blocks that will be saved by the next **StartLog**. The valid values for *numBlocks* are **0**, **1**,...,**99999**.

Set LogRate *rate*
Specifies *rate* as the number of seconds between successive A/D acquisitions. The valid values for *rate* are as follows:

- **2.0**, **2.1**,...,**99.9**, or
- **102**, **108**,...,**102 + 6m**,...,**5994** with **m** an integer in the range [0,982]

Set StartChannel

Syntax

Set StartChannel *channel*
Se Sta *channel*

Description

Specifies *channel* as the first channel to be scanned during the next A/D operation.

Set StopChannel

Syntax

Set StopChannel *channel*
Se Sto *channel*

Description

Specifies *channel* as the last channel to be scanned during the next A/D operation.

Set Units

Syntax

Set Units (ADcodes,Volts)
Se Un (ADco,Vo)

Description

Sets the units that the Pop Up Control Panel will use to display data.

Show

Syntax

Show (1,2)
Sh (1,2)

Description

Causes the display of the specified panel of the Pop Up Control Panel.

Arguments

1 = Main panel
2 = LogFile panel

Notes

This command is ignored if VITASK (instead of VI) was loaded immediately before MADC16 was loaded.

StartLog

Syntax

StartLog
Startlog

Description

Writes the current A/D data into a file according to the conditions specified by the most recently issued **Set {Mode/LogFile/Date/Block/LogStat}**.

StopLog

Syntax

StopLog
Stoplog

Description

Stops current logging operation.

Unlock

Syntax

Unlock
Un

Description

Enables keyboard and mouse control of the Pop Up Control Panel.

Notes

Use **Lock** to cancel the effect of **Unlock**.

Function Call Driver error messages

A

Error 6000H **Error In Configuration File**

Cause The configuration file supplied to ADC16_DevOpen() is corrupt or does not exist. If file is known to be good, then it probably contains one or more undefined keywords.

Solution Check if the file exists at the specified path. Check for illegal keywords in file; the best way to fix illegal keywords is to let the supplied ADC16CFG.EXE utility do it.

Error 6001H **Illegal Base Address in Configuration File**

Error 6004H **Error Opening Configuration File**

Error 6005H **Illegal Channel Number**

Cause The specified I/O operation channel is out of range. For A/D operations, the legal channel numbers are **0**, **1**,..., **7(m+1)**, where **m** is the number of STA-EX8s connected to the board. For digital operations, **0** is the only valid channel.

Solution Specify legal channel number.

Error 6006H **Illegal gain**

Cause The specified Analog Input (A/D) operation gain code is out of range. The allowed codes are: 0, 1, 2. Refer to the appropriate function call description for more detail.

Solution Specify legal gain code.

Error 6008H **Bad Number in Configuration File**

Cause An illegal specification of a number is detected in the Configuration file. Note that if specifying a hexadecimal number for the Base Address, that number must proceeded with '&H'.

Solution Check the number following 'Address' in the Configuration file.

Error 6009H **Incorrect Version Number**

Error 600AH **Configuration file not found**

Cause This error is returned by the ADC16_DevOpen() function whenever the specified configuration file is not found.

Solution Check the configuration file name (spelling!), path, etc...

Error 600CH **Error in returning INT Buffer**

Cause This error occurs during K_IntFree() whenever DOS returns an error in INT 21H function 49H.

Solution Make sure that the parameter passed to K_IntFree() was previously obtained via K_IntAlloc().

Error 600DH **Bad Frame handle**

Cause This error is usually returned by Frame Management or an Operation Function whenever an illegal Frame handle is passed to one of these functions.

Solution Check the Frame Handle.

Error 600eH **No more Frame Handles**

Error 600fH **Requested Int Buffer Too Large**

Error 6010H **Cannot Allocate Int Buff**

Error 6011H **Int Buffer Already allocated**

Error 6012H **Int Buffer De-Allocation Error**

Error 6013H **Int Buffer Never Allocated**

Error 7000H **No board name**

Cause ADC16_DevOpen() function did not find a board 'Name' in the specified configuration file.

Solution Make sure that a name is specified in your configuration file. The legal name is ADC16.

Error 7001H **Bad board name**

Cause ADC16_DevOpen() function found the board 'name' in the specified configuration file to be illegal. The legal name is ADC16.

Solution Check the keyword following 'Name' in your configuration file.

Error 7002H **Bad board number**

Cause ADC16_DevOpen() function found the 'Board' number in the specified configuration file to be illegal. The legal board numbers are 0 and 1.

Solution Check the number following 'Board' in your configuration file.

Error 7003H **Bad base address**

Cause ADC16_DevOpen() function found the board's base I/O 'Address' in the specified configuration file to be illegal. The legal address are 200H (512) through 3F0H (1008) in increments of 10H (16) inclusive.

Solution Check the number following 'Address' in your configuration file. NOTE that to specify a Hex number, the number must be preceded by '&H'.

Error 7005H **Bad Interrupt Level**

Cause ADC16_DevOpen() function found the Interrupt Level in the specified configuration file to be illegal. The legal Interrupt levels are 2, 3, 4, 5, 7, 10, 11, 15.

Solution Check the number following 'IntLevel' in your configuration file.

Error 7006H **Bad number of EXPs**

Cause ADC16_DevOpen() function found the number of EXPs in the specified configuration file to be illegal. The legal number of EXPs is 0 through 8 inclusive.

Solution Check the number following 'STAEX8' in your configuration file.

Error 7018H **No board name**

Cause ADC16_DevOpen() function found the board 'Name' in the specified configuration file to be illegal. The legal name is ADC16.

Solution Check the keyword following 'Name' in your configuration file.

Error 701bH **Resource Busy**

Error 8001H **Function not supported**

Cause A request is made to a function not supported by the ADC16 driver. This error should not occur in a standard release software.

Solution Contact Keithley Data Acquisition Technical Support.

Error 8002H **Function out of bounds**

Cause Illegal function number is specified. This error should not occur in a standard release software.

Solution Contact Keithley Data Acquisition Technical Support.

Error 8003H **Illegal board number**

Cause The ADC16 driver supports up to two boards: 0 and 1.

Solution Check the board number parameter in your call to ADC16_ GetDevHandle().

Error 8004H **Bad error**

Cause An illegal error number was passed to function K_GetErrMsg(). The legal error numbers are listed in this appendix.

Solution Check the error number.

Error 8005H **No board**

Cause This error is issued during K_DASDevInit() whenever the board presence test fails. This is normally caused by a conflict in the specified board I/O address and the actual I/O address the board is configured for. Also, this error is issued when the board is not present in the system.

Solution Check the board's base I/O address dip switch and make sure it matches the base address in your configuration file.

Error 8006H **A/D not initialized**
Error 8008H **Digital Input not initialized**
Error 8009H **Digital Output not initialized**

Cause An attempt to start the particular operation without first initializing the associated Frame.

Solution Use K_InitFrame() to initialize the particular frame you wish to use.

Error 801AH **Interrupts active**

Cause An attempt is made to start an Interrupt-based operation while another is already active.

Solution Stop current interrupt-mode operation first and retry.

Error 8020 **Bad Revision**

Cause Specified DAS revision number is not valid.

Error 8021 **Error – Resource Busy**

Cause Illegal handle for frame.

Error 8022 **Unknown error**

Error FFFFH **User aborted operation by pressing Ctrl – Break or Ctrl – C .**

File I/O Driver error messages

B

Error 850 **Illegal character encountered.**

Error 851 **Illegal ADC-16 Command**

Cause The driver does not recognize the command.

Solution Refer to the File I/O Command Reference to check the syntax and spelling of the command.

Error 852 **ADC-16 SEt Command Error**

Cause Keyword specified in a SET command is not valid.

Solution Refer to the File I/O Command Reference to check which keywords are valid for the command.

Error 853 **ADC-16 REad Command Error**

Cause Keyword specified in a READ command is not valid.

Solution Refer to the File I/O Command Reference to check which keywords are valid for the command.

Error 854 **Illegal ADType, Should be SYNChronous or INTerrupt**

Error 855 **Illegal Gain, should 1, 10, or 100.**

Error 856 **Illegal Interrupt level, should be 2, 3, 4, 5, 7, 10, 11, 15.**

Error 857 **Interrupt mode NOT Enabled.**

Error 858 **Illegal unit selection, should be ADCOdes or VOlts.**

Error 859 **Illegal Start Channel Selection, should be 0 to 7.**

Error 860 **Illegal Stop Channel Selection, should be 0 to 7.**

Error 861 **Error, STOP Device before using this command.**

Error 862 **Error, Channel must be 0 to 7.**

Error 863 **Error, START acquisitions before Reading Channel Data.**

Error 864 **Error, Data OVERFLOW or Channel NOT Enabled.**

Error 865 **Illegal Command MUST be SHow, SHow 1, or SHow 2.**

Error 866 **Log File Name Error.**

Error 867 **Illegal File Mode, MUST be NEw, OVerwrite or APpend**

Error 868 **Illegal Date Mode, MUST be OFf or ON.**

Error 869 **Illegal Number of Blocks, MUST be 1-99999.**

Error 870 **Illegal Log Rate Setting.**

Error 871 **Start Acquisitions before Logging.**