

DAS-4300 Series
Function Call Driver

USER'S GUIDE

**DAS-4300 Series
Function Call Driver
User's Guide**

Revision A - June 1995
Part Number: 94530

New Contact Information

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, OH 44139

Technical Support: 1-888-KEITHLEY
Monday – Friday 8:00 a.m. to 5:00 p.m (EST)
Fax: (440) 248-6168

Visit our website at <http://www.keithley.com>

The information contained in this manual is believed to be accurate and reliable. However, Keithley Instruments, Inc., assumes no responsibility for its use or for any infringements of 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 Instruments, Inc.

KEITHLEY INSTRUMENTS, INC., SHALL NOT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RELATED TO THE USE OF THIS PRODUCT. THIS PRODUCT IS NOT DESIGNED WITH COMPONENTS OF A LEVEL OF RELIABILITY SUITABLE FOR USE IN LIFE SUPPORT OR CRITICAL APPLICATIONS.

Refer to your Keithley Instruments license agreement for specific warranty and liability information.

All brand and product names are trademarks or registered trademarks of their respective companies.

© Copyright Keithley Instruments, Inc., 1995.

All rights reserved. Reproduction or adaptation of any part of this documentation beyond that permitted by Section 117 of the 1976 United States Copyright Act without permission of the Copyright owner is unlawful.

Keithley MetraByte Division

Keithley Instruments, Inc.

440 Myles Standish Blvd. Taunton, MA 02780

Telephone: (508) 880-3000 • FAX: (508) 880-0179

Preface

This manual describes how to write application programs for DAS-4301/8K boards using the DAS-4300 Series Function Call Driver. The DAS-4300 Series Function Call Driver supports the following DOS-based languages:

- Microsoft[®] C/C++ (Version 4.0 and higher)
- Borland[®] C/C++ (Version 1.0 and higher)

The DAS-4300 Series Function Call Driver supports the following Windows-based languages:

- Microsoft C/C++ (Version 7.0 and higher)
- Borland C/C++ (Version 4.0 and higher)
- Microsoft Visual Basic[®] for Windows (Version 3.0 and higher)

The manual is intended for application programmers using a DAS-4301/8K board in an IBM[®] PC AT[®] or compatible computer. It is assumed that users have read the *DAS-4300 Series User's Guide* to familiarize themselves with the board's features, and that they have completed the appropriate hardware installation and configuration. It is also assumed that users are experienced in programming in their selected language and that they are familiar with data acquisition principles.

The *DAS-4300 Series Function Call Driver User's Guide* is organized as follows:

- Chapter 1 contains the information needed to install the DAS-4300 Series Function Call Driver and to get help.
- Chapter 2 contains the background information needed to use the functions included in the DAS-4300 Series Function Call Driver.
- Chapter 3 contains programming guidelines and language-specific information related to using the DAS-4300 Series Function Call Driver.
- Chapter 4 contains detailed descriptions of the DAS-4300 Series Function Call Driver functions, arranged in alphabetical order.
- Appendix A contains a list of the error codes returned by DAS-4300 Series Function Call Driver functions.
- Appendix B contains instructions for converting counts to voltage and for converting voltage to counts.
- Appendix C present bandwidth charts for the supported input ranges.

An index completes this manual.

Table of Contents

Preface

1 Getting Started

2 Available Operations

System Operations	2-1
Initializing the Driver	2-2
Initializing a Board	2-2
Retrieving Revision Levels	2-3
Handling Errors	2-3
Analog Input Operations	2-4
Operation Mode	2-4
Memory Allocation and Management	2-5
Dimensioning a Local Array	2-5
Dynamically Allocating a Memory Buffer	2-5
Assigning the Starting Address	2-7
Channels	2-7
Input Ranges	2-7
Pacer Clocks	2-8
Internal Pacer Clock	2-8
External Pacer Clock	2-9
Triggers	2-10
Trigger Sources	2-10
Internal Trigger	2-10
External Analog Trigger	2-11
External Digital Trigger	2-12
Trigger Acquisition	2-13
Post-Trigger Acquisition	2-13
Pre-Trigger Acquisition	2-14
About-Trigger Acquisition	2-15

3 Programming with the Function Call Driver

How the Driver Works	3-1
Programming Overview	3-5
Preliminary Tasks	3-6
Analog Input Programming Tasks	3-6

C/C++ Programming Information	3-8
Dimensioning and Assigning a Local Array	3-8
Dynamically Allocating and Assigning a Memory Buffer	3-9
Allocating a Memory Buffer	3-9
Accessing the Data	3-10
Handling Errors	3-10
Programming in Microsoft C/C++ (for DOS)	3-11
Programming in Microsoft C/C++ (for Windows)	3-12
Programming in Borland C/C++ (for DOS)	3-13
Programming in Borland C/C++ (for Windows)	3-14
Microsoft Visual Basic for Windows	
Programming Information	3-15
Dimensioning and Assigning a Local Array	3-15
Dynamically Allocating and Assigning a Memory Buffer	3-16
Allocating a Memory Buffer	3-16
Accessing the Data	3-17
Handling Errors	3-17
Programming in Microsoft Visual Basic for Windows	3-18

4 Function Reference

K_ClearFrame	4-5
K_CloseDriver	4-6
K_ClrAboutTrig	4-7
K_DASDevInit	4-8
K_FreeDevHandle	4-9
K_FreeFrame	4-10
K_GetADFrame	4-11
K_GetClkRate	4-13
K_GetDevHandle	4-15
K_GetErrMsg	4-17
K_GetShellVer	4-18
K_GetVer	4-19
K_IntAlloc	4-21
K_IntFree	4-23
K_IntStart	4-24
K_IntStatus	4-25
K_IntStop	4-28
K_MoveBufToArray	4-30
K_OpenDriver	4-31
K_SetAboutTrig	4-33
K_SetADTrig	4-34
K_SetBuf	4-36

K_SetBufI	4-38
K_SetChn	4-40
K_SetClk	4-41
K_SetClkRate	4-42
K_SetDITrig	4-44
K_SetG	4-46
K_SetPostTrigDelay	4-48
K_SetTrig	4-50

A Error/Status Codes

B Data Formats

Converting Counts to Voltage	B-1
Converting Voltage to Counts	B-3

C Bandwidth Charts for Input Voltage Ranges

Index

List of Figures

Figure 2-1. Analog Trigger Conditions	2-12
Figure 2-2. Digital Trigger Conditions	2-13
Figure 3-1. Interrupt-Mode Operation	3-3
Figure 4-1. Status Word Settings	4-26
Figure C-1. ± 0.2 V Input Range (Gain Code 0)	C-1
Figure C-2. ± 0.25 V Input Range (Gain Code 1)	C-2
Figure C-3. ± 0.5 V Input Range (Gain Code 2)	C-2
Figure C-4. ± 1 V Input Range (Gain Code 3)	C-3
Figure C-5. ± 0.125 V Input Range (Gain Code 4)	C-3
Figure C-6. ± 0.15625 V Input Range (Gain Code 5)	C-4
Figure C-7. ± 0.3125 V Input Range (Gain Code 6)	C-4
Figure C-8. ± 0.625 V Input Range (Gain Code 7)	C-5
Figure C-9. ± 0.1 V Input Range (Gain Code 8)	C-5
Figure C-10. ± 0.125 V Input Range (Gain Code 9)	C-6
Figure C-11. ± 0.25 V Input Range (Gain Code 10)	C-6
Figure C-12. ± 0.5 V Input Range (Gain Code 11)	C-7
Figure C-13. ± 0.025 V Input Range (Gain Code 12)	C-7
Figure C-14. ± 0.03125 V Input Range (Gain Code 13)	C-8
Figure C-15. ± 0.0625 V Input Range (Gain Code 14)	C-8
Figure C-16. ± 0.125 V Input Range (Gain Code 15)	C-9

List of Tables

Table 2-1.	Supported Operations	2-1
Table 2-2.	Analog Input Ranges	2-8
Table 2-3.	Available Conversion Rates Using Internal Clock	2-9
Table 3-1.	A/D Frame Elements	3-4
Table 3-2.	Setup Functions for Interrupt-Mode Analog Input Operations	3-7
Table 4-1.	Functions	4-2
Table 4-2.	Data Type Prefixes.	4-4
Table A-1.	Error/Status Codes	A-1
Table B-1.	Some Span Values For Analog Input Data Conversion Equations	B-2

Table 2-1.	Supported Operations	2-1
Table 2-2.	Analog Input Ranges	2-8
Table 2-3.	Available Conversion Rates Using Internal Clock	2-9
Table 3-1.	A/D Frame Elements	3-4
Table 3-2.	Setup Functions for Interrupt-Mode Analog Input Operations	3-7
Table 4-1.	Functions	4-2
Table 4-2.	Data Type Prefixes.	4-4
Table A-1.	Error/Status Codes	A-1
Table B-1.	Some Span Values For Analog Input Data Conversion Equations	B-2

Figure 2-1.	Analog Trigger Conditions	2-12
Figure 2-2.	Digital Trigger Conditions	2-13
Figure 3-1.	Interrupt-Mode Operation	3-3
Figure 4-1.	Status Word Settings	4-26
Figure C-1.	± 0.2 V Input Range (Gain Code 0)	C-1
Figure C-2.	± 0.25 V Input Range (Gain Code 1)	C-2
Figure C-3.	± 0.5 V Input Range (Gain Code 2)	C-2
Figure C-4.	± 1 V Input Range (Gain Code 3)	C-3
Figure C-5.	± 0.125 V Input Range (Gain Code 4)	C-3
Figure C-6.	± 0.15625 V Input Range (Gain Code 5)	C-4
Figure C-7.	± 0.3125 V Input Range (Gain Code 6)	C-4
Figure C-8.	± 0.625 V Input Range (Gain Code 7)	C-5
Figure C-9.	± 0.1 V Input Range (Gain Code 8)	C-5
Figure C-10.	± 0.125 V Input Range (Gain Code 9)	C-6
Figure C-11.	± 0.25 V Input Range (Gain Code 10)	C-6
Figure C-12.	± 0.5 V Input Range (Gain Code 11)	C-7
Figure C-13.	± 0.025 V Input Range (Gain Code 12)	C-7
Figure C-14.	± 0.03125 V Input Range (Gain Code 13)	C-8
Figure C-15.	± 0.0625 V Input Range (Gain Code 14)	C-8
Figure C-16.	± 0.125 V Input Range (Gain Code 15)	C-9

1

Getting Started

The DAS-4300 Series Function Call Driver is a library of data acquisition and control functions (referred to as the Function Call Driver or FCD functions). It is part of the following two software packages:

- **DAS-4300 Series Standard Software package** - This is the software package that is shipped with DAS-4301/8K boards; it includes utility programs, running under DOS, that allow you to configure, calibrate, and test the DAS-4301/8K boards.
- **ASO-4300 software package** - This is the Advanced Software Option for DAS-4301/8K boards. You purchase the ASO-4300 software package separately from the board; it includes the following:
 - Libraries of FCD functions for Microsoft C/C++ (for DOS) and Borland C/C++ (for DOS).
 - Dynamic Link Libraries (DLLs) of FCD functions for Microsoft Visual Basic for Windows, Microsoft C/C++ (for Windows), and Borland C/C++ (for Windows).
 - Support files, containing program elements, such as function prototypes and definitions of variable types, that are required by the FCD functions.
 - Utility programs, running under DOS, that allow you to configure, calibrate, and test the DAS-4301/8K boards.
 - Language-specific example programs.

Before you use the Function Call Driver, make sure that you have installed the software, set up the board, and created a configuration file using the setup and installation procedures described in the *DAS-4300 Series User's Guide*.

If you need help installing or using the DAS-4300 Series Function Call Driver, call your local sales office or the Keithley Metrabyte Applications Engineering Department at:

(508) 880-3000

Monday - Friday, 8:00 A.M. - 6:00 P.M., Eastern Time

An applications engineer will help you diagnose and resolve your problem over the telephone.

Please make sure that you have the following information available before you call:

DAS-4301/8K board configuration	Serial #	_____
	Revision code	_____
	Base I/O address setting	_____
	Memory address setting	_____
	Interrupt level setting	_____
Computer	Manufacturer	_____
	CPU type	_____
	Clock speed (MHz)	_____
	KB of RAM	_____
	Video system	_____
	BIOS type	_____
	Memory manager	_____
Operating system	DOS version	_____
	Windows version	_____
	Windows mode	_____
Software package	Name	_____
	Serial #	_____
	Version	_____
	Invoice/Order #	_____

**Compiler
(if applicable)**

Language
Manufacturer
Version

Accessories

Type

2

Available Operations

This chapter contains the background information you need to use the FCD functions to perform operations on DAS-4301/8K boards. The supported operations are listed in Table 2-1.

Table 2-1. Supported Operations

Operation	Page Reference
System	page 2-1
Analog input	page 2-4

System Operations

This section describes the miscellaneous operations and general maintenance operations that apply to DAS-4301/8K boards and to the DAS-4300 Series Function Call Driver. It includes information on initializing a driver, initializing a board, retrieving revision levels, and handling errors.

Initializing the Driver

You must initialize the DAS-4300 Series Function Call Driver and any other Keithley DAS Function Call Drivers you are using in your application program. To initialize the drivers, use the **K_OpenDriver** function. You specify the driver you are using and the configuration file that defines the use of the driver. The driver returns a unique identifier for the driver; this identifier is called the driver handle.

You can specify a maximum of 30 driver handles for all the Keithley MetraByte drivers initialized from all your application programs. If you no longer require a driver and you want to free some memory or if you have used all 30 driver handles, you can use the **K_CloseDriver** function to free a driver handle and close the associated driver.

If the driver handle you free is the last driver handle specified for a Function Call Driver, the driver is shut down. (For Windows-based languages only, the DLLs associated with the Function Call Driver are shut down and unloaded from memory.)

Initializing a Board

The DAS-4300 Series Function Call Driver supports up to two boards. You must use the **K_GetDevHandle** function to specify the boards you want to use. The driver returns a unique identifier for each board; this identifier is called the device handle.

Device handles allow you to communicate with more than one board. You use the device handle returned by **K_GetDevHandle** in subsequent function calls related to the board.

You can specify a maximum of 30 device handles for all the Keithley MetraByte boards accessed from all your application programs. If a board is no longer being used and you want to free some memory or if you have used all 30 device handles, you can use the **K_FreeDevHandle** function to free a device handle.

To reinitialize a board during an operation, use the **K_DASDevInit** function. **K_GetDevHandle** and **K_DASDevInit** perform the following tasks:

- Abort all operations currently in progress that are associated with the board identified by the device handle.
- Verify that the board identified by the device handle is the board specified in the configuration file associated with the board.

Retrieving Revision Levels

If you are using functions from different Keithley DAS Function Call Drivers in the same application program or if you are having problems with your application program, you may want to verify which versions of the Function Call Driver, Keithley DAS Driver Specification, and Keithley DAS Shell are installed on your computer.

The **K_GetVer** function allows you to get both the revision number of the DAS-4300 Series Function Call Driver and the revision number of the Keithley DAS Driver Specification to which the driver conforms.

The **K_GetShellVer** function allows you to get the revision number of the Keithley DAS Shell (the Keithley DAS Shell is a group of functions that are shared by all DAS boards).

Handling Errors

Each FCD function returns a code indicating the status of the function. To ensure that your application program runs successfully, it is recommended that you check the returned code after the execution of each function. If the status code equals 0, the function executed successfully and your program can proceed. If the status code does not equal 0, an error occurred; ensure that your application program takes the appropriate action. Refer to Appendix A for a complete list of error codes.

Each supported programming language uses a different procedure for error checking. Refer to the following for information:

C/C++	page 3-10
Visual Basic for Windows	page 3-17

For C-language application programs only, the DAS-4300 Series Function Call Driver provides the **K_GetErrMsg** function, which gets the address of the string corresponding to an error code.

Analog Input Operations

This section describes the following:

- Analog input operation mode available.
- How to allocate and manage memory for analog input operations.
- How to specify the following for an analog input operation: a channel, a gain and range, a clock source, a trigger source, and the trigger acquisition type.

Operation Mode

DAS-4301/8K boards support interrupt mode only. In interrupt mode, the board acquires multiple samples from a single analog input channel. A hardware clock initiates A/D conversions. Once the analog input operation begins, control returns to your application program. The hardware continues to store the acquired data in its onboard memory until the specified number of samples is acquired, then transfers the data all at once to a user-defined array or buffer in the computer using an interrupt service routine.

Use the **K_IntStart** function to start an analog input operation in interrupt mode. Use the **K_IntStop** function to stop an interrupt-mode operation. Use the **K_IntStatus** function to determine the current status of an interrupt-mode operation.

The converted data is stored as counts. For information on converting counts to voltage, refer to Appendix B.

Memory Allocation and Management

Interrupt-mode analog input operations require a memory location in which to store acquired data. DAS-4301/8K boards can transfer up to 8,192 samples (bytes) to this memory location.

Note: Even though you can reserve a memory location greater than the board requires, to conserve memory it is recommended that you allocate only the required amount (maximum of 8,192 bytes).

The ways you can allocate and manage memory are described in the following sections.

Dimensioning a Local Array

The simplest way to reserve a memory location is to dimension an array within your application program's memory area. This is the recommended way to reserve memory for this driver. The advantage of this method is that the array is directly accessible to your application program. The limitation of this method is that local arrays occupy permanent memory areas; these memory areas cannot be freed to make them available to other programs or processes.

Since the DAS-4300 Series Function Call Driver stores data in 16-bit integers, you must dimension a local array as an integer data type.

Dynamically Allocating a Memory Buffer

You can allocate a memory buffer dynamically outside of your application program's memory area. The advantages of this method are as follows:

- The size of the buffer is limited by the amount of free physical memory available in your computer at run time.
- A dynamically allocated memory buffer can be freed to make it available to other programs or processes.

The limitation of this method is that, for Visual Basic for Windows, data in a dynamically allocated buffer is not directly accessible to your program. You must use the **K_MoveBufToArray** function to move the data from the dynamically allocated buffer to the program's local array; refer to page 3-16 for more information.

Use the **K_IntAlloc** function to dynamically allocate a memory buffer. You specify the operation requiring the buffer and the number of samples to store in the buffer (maximum of 8,192). The driver returns the starting address of the buffer and a unique identifier for the buffer (this identifier is called the memory handle). When the buffer is no longer required, you can free the buffer for another use by specifying this memory handle in the **K_IntFree** function.

Notes: For DOS-based languages, the area used for dynamically allocated memory buffers is referred to as the far heap; for Windows-based languages, this area is referred to as the global heap. These heaps are areas of memory left unoccupied as your application program and other programs run.

For DOS-based languages, the **K_IntAlloc** function uses the DOS Int 21H function 48H to dynamically allocate far heap memory. For Windows-based languages, the **K_IntAlloc** function calls the **GlobalAlloc** API function to allocate the desired buffer size from the global heap.

For Windows-based languages, dynamically allocated memory is guaranteed to be fixed and locked in memory.

Assigning the Starting Address

After you dimension your array or allocate your buffer, you must assign the starting address of the array or buffer and the number of samples to store in the array or buffer. Each supported programming language requires a particular procedure for assigning a starting address. Refer to the following for information:

Language	Memory Location	Function	See...
C/C++	Array or Buffer	K_SetBuf	page 3-9
Visual Basic for Windows	Array	K_SetBufI	page 3-15
	Buffer	K_SetBuf	page 3-16

Channels

The DAS-4301/8K board provides two analog input channels; the software refers to the analog input signal from the Ch A connector as channel 0 and the analog input signal from the Trg/Ch B connector as channel 1. You can perform an analog input operation on a single channel at a time. To acquire samples from both channels, you must alternate between the two channels after an acquisition is done.

You can acquire a single sample or multiple samples from a single analog input channel. Use the **K_SetChn** function to specify the channel.

Input Ranges

Each channel on the DAS-4301/8K board can measure signals in one of 16, software-selectable bipolar analog input ranges.

Table 2-2 lists the analog input ranges supported by DAS-4301/8K boards and the gain code associated with each range. Use the **K_SetG** function to specify the gain code. Refer to Appendix C to understand the effect of the input range on the bandwidth of the DAS-4301/8K board.

Table 2-2. Analog Input Ranges

Analog Input Range	Gain Code	Analog Input Range	Gain Code
±25 mV	12	±200 mV	0
±31.25 mV	13	±250 mV	1, 10
±62.5 mV	14	±312.5 mV	6
±100 mV	8	±0.5 V	2, 11
±125 mV	4, 9, 15	±0.625 V	7
±156.25 mV	5	±1.0 V	3

Pacer Clocks

The pacer clock determines the period between A/D conversions. Use the **K_SetClk** function to specify an internal or an external pacer clock. The internal pacer clock is the default pacer clock.

The internal and external pacer clocks are described in the following subsections; refer to the *DAS-4300 Series User's Guide* for more information.

Internal Pacer Clock

When you start an analog input operation (using **K_IntStart**), conversions are performed at a rate of 20 Gsamples/s divided by a count value of 20, 40, 80, 200, 400, 800, 1600, 3200, 6400, 12800, or 25600. Use the **K_SetClkRate** function to specify the count value. Each count represents .05 ns between conversions.

Table 2-3 lists the conversion rates, sample periods, and count values for the internal pacer clock.

Table 2-3. Available Conversion Rates Using Internal Clock

Conversion Rate	Sample Period	Count Value
1 Gsamples/s	1 ns	20
500 Msamples/s	2 ns	40
250 Msamples/s	4 ns	80
100 Msamples/s	10 ns	200
50 Msamples/s	20 ns	400
25 Msamples/s	40 ns	800
12.5 Msamples/s	80 ns	1600
6.25 Msamples/s	160 ns	3200
3.125 Msamples/s	320 ns	6400
1.5625 Msamples/s	640 ns	12800
0.78250 Msamples/s	1280 ns	25600

Note: If you enter a count value that is not one of those listed in Table 2-3, the driver uses the next fastest rate. For example, if you enter a count value of 50, the driver uses a count value of 40 to perform the faster conversion rate. To determine the actual count value used, use the **K_GetClkRate** function.

External Pacer Clock

When you start an analog input operation (using **K_IntStart**), conversions are armed. At the next rising edge of the external pacer clock (and at every subsequent rising edge of the external pacer clock), a conversion is initiated.

Do not use a conversion rate less than 100 Msamples/s when using an external pacer clock or an error will result.

Triggers

A trigger is an event that occurs based on a specified set of conditions. The operation must have a start trigger that determines when the acquisition starts. In addition, you can choose the optional about trigger to determine when the acquisition stops.

You can define operations that acquire data after the trigger event occurs (post-trigger acquisition), operations that acquire data before a trigger event (pre-trigger acquisition), and operations that acquire data before and after a trigger event (about-trigger acquisition). For post-trigger acquisitions, you can also specify a post-trigger delay. If you specify an about trigger, the operation stops when a specified number of samples has been acquired after the occurrence of the about-trigger event.

When the trigger event occurs, a TTL-level, signal is output on the Trg IO connector. The signal is edge sensitive with a positive polarity.

The following sections describe the supported trigger sources and the ways to acquire data using triggers.

Trigger Sources

Use **K_SetTrig** to specify an internal or an external trigger source. External triggers can be either analog triggers or digital triggers. The trigger event is not significant until the operation the trigger governs has been started (using **K_IntStart**).

The internal trigger, external analog trigger, and external digital trigger are described in the following subsections.

Internal Trigger

An internal trigger is a software trigger. The trigger event occurs when you start the operation using the **K_IntStart** function. Note that there is a slight delay between the time you start the operation and the time the trigger event occurs. The internal trigger is the default trigger source.

External Analog Trigger

You can select the digitized analog input signal from the Ch A connector (referred to in software as analog trigger channel 0) or from the Trg/Ch B connector (referred to in software as analog trigger channel 1) as the trigger signal. You program the trigger level as a count value in 256 steps (-128 to +127). If you use analog trigger channel 0 or 1, you must acquire data from the same channel you are using as the trigger channel. For example, if you specify 0 as the trigger channel, you must also acquire analog input data from the Ch A connector by specifying 0 in **K_SetChn**.

You can also select the ± 10 V trigger input signal from the Trg/Ch B connector (referred to in software as analog trigger channel 2) as the trigger signal. You program the trigger level as a count value in 256 steps (-127 to 128) from -10 V to +9.922 V. If you use analog trigger channel 2, you can acquire analog input data from either the Ch A connector or from the Trg/Ch B connector.

Note: Even though the resolution of the ± 10 V range is 12 bits in hardware, the driver accepts only 8 bits, resulting in the loss of some accuracy.

The trigger conditions for external analog triggers are illustrated in Figure 2-1 and described as follows:

- **Positive-Edge Trigger** - A trigger event occurs the first time the trigger signal changes from a voltage that is less than the trigger level to a voltage that is greater than the trigger level.
- **Negative-Edge Trigger** - A trigger event occurs the first time the trigger signal changes from a voltage that is greater than the trigger level to a voltage that is less than the trigger level.

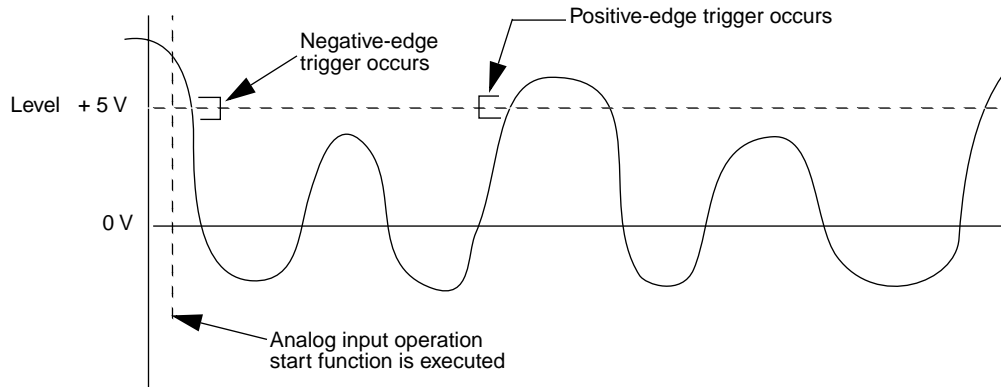


Figure 2-1. Analog Trigger Conditions

Use the **K_SetADTrig** function to specify the analog input channel to use as the trigger channel, the trigger level, and the trigger polarity. The trigger sensitivity is always edge for the DAS-4301/8K board.

Refer to Appendix B for information on how to convert a voltage to a count value.

External Digital Trigger

The digital trigger signal is connected to the Trg IO connector of the DAS-4301/8K board. Use **K_SetDITrig** to specify the digital input channel to use as the trigger channel and whether you want the trigger event to occur on the rising edge of the digital trigger signal (positive-edge trigger) or on a falling edge of the digital trigger signal (negative-edge trigger). The trigger sensitivity is always edge for DAS-4301/8K boards. The trigger events are illustrated in Figure 2-2.

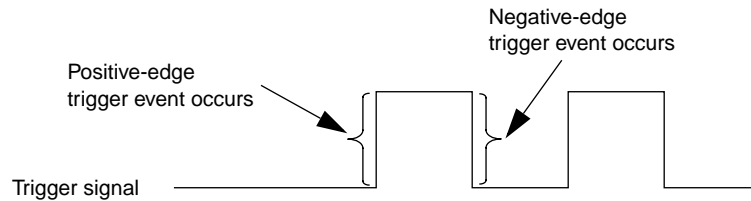


Figure 2-2. Digital Trigger Conditions

Trigger Acquisition

The maximum number of samples you can collect for each trigger event is 8,192. The minimum number of samples you can collect for each trigger event depends on the conversion rate as follows:

- **Conversion rate of 100 Msamples/s and slower:** minimum of 1 sample in increments of 1 sample.
- **Conversion rate of 250 Msamples/s and faster:** minimum of 10 samples in increments of 10 samples.

The ways you can acquire data using triggers are described in the following subsections.

Post-Trigger Acquisition

Use post-trigger acquisition in applications where you want to collect data after a specific trigger event. You specify a start trigger only; the start trigger determines when the operation starts and can be either an internal, an external analog, or an external digital trigger. To stop the operation, use the **K_IntStop** function. If desired, you can specify the number of samples to wait between when the trigger event occurs and when the data is collected by using the **K_SetPostTrigDelay** function.

At a conversion rate of 100 Msamples/s or slower, the post-trigger delay can range from 0 to 65,536 samples (in increments of 1); at a conversion rate of 250 Msamples/s or faster, the post-trigger delay can range from 0 to 655,360 samples (in increments of 10).

To specify post-trigger acquisition, perform the following steps:

1. Specify the start trigger.
 - Use **K_SetTrig** to specify an internal or an external trigger source (specify external for an analog or digital trigger).
 - If you specify an external start trigger in **K_SetTrig**, define the start trigger conditions using **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger).
2. If you specified an external analog or digital start trigger, use **K_ClrAboutTrig** to disable the about trigger.
3. To specify the number of post-trigger samples to wait after the trigger event occurs and before data is collected, use **K_SetPostTrigDelay**.

Pre-Trigger Acquisition

Use pre-trigger acquisition in applications where you want to collect data before a specific trigger event. The operation starts when your application program calls the **K_IntStart** function. The about trigger can be either an external analog or external digital trigger; the operation stops when the about-trigger event occurs.

Note: The memory buffer must fill up with data at least once before the board can accept a trigger event.

To specify pre-trigger acquisition, perform the following steps:

1. Use **K_SetTrig** to specify an external about-trigger source (external analog or external digital trigger).
2. Use **K_SetAboutTrig** to enable the about trigger and to set the number of samples to 1.

Note: The minimum number of samples that you can specify in **K_SetAboutTrig** is 1.

3. Specify the trigger conditions for the about trigger.
 - If the about trigger is an external analog trigger, use **K_SetADTrig** to specify the trigger conditions for the about trigger.
 - If the about trigger is an external digital trigger, use **K_SetDITrig** to specify the trigger conditions for the about trigger.

About-Trigger Acquisition

Use about-trigger acquisition in applications where you want to collect data both before and after a specific trigger event. The operation starts when your application program calls the **K_IntStart** function. The about trigger can be either an external analog or external digital trigger; the operation stops after a specified number of samples has been acquired after the about-trigger event occurs.

Note: The memory location must fill up with data at least once before the board can accept a trigger event.

To specify about-trigger acquisition, perform the following steps:

1. Use **K_SetTrig** to specify an external about-trigger source (external analog or external digital trigger).
2. Use **K_SetAboutTrig** to enable the about trigger and to specify the desired number of post-trigger samples.
3. Specify the trigger conditions for the about trigger.
 - If the about trigger is an external analog trigger, use **K_SetADTrig** to specify the trigger conditions for the about trigger.
 - If the about trigger is an external digital trigger, use **K_SetDITrig** to specify the trigger conditions for the about trigger.

After the about-trigger acquisition is completed, the software automatically ensures that the post-trigger samples are the last samples in the array or buffer.

For example, suppose you set the the number of post-trigger samples in **K_SetAboutTrig** to 4,160 and start the about-trigger acquisition. Pre-trigger data is collected and begins to fill the 8,192 byte buffer. Once the buffer is filled, the board can accept a trigger; pre-trigger data continues to be collected and overwrites the data in the buffer until the trigger event occurs. When a valid trigger event occurs, the board collects 4,160 post-trigger samples then stops the acquisition. The number of pre-trigger samples in the buffer is 8,192 minus 4,160 or 4,032.

3

Programming with the Function Call Driver

This chapter contains an overview of the structure of the DAS-4300 Series Function Call Driver, as well as programming guidelines and language-specific information to assist you when writing application programs with the DAS-4300 Series Function Call Driver.

How the Driver Works

When writing application programs, you can use functions from one or more Keithley MetraByte DAS Function Call Drivers. You initialize each driver according to a particular configuration file. If you are using more than one driver or more than one configuration file with a single driver, the driver handle uniquely identifies each driver or each use of the driver.

You can program one or more boards in your application program. You initialize each board using a device handle to uniquely identify each board. Each device handle is associated with a particular driver.

The driver uses frames to perform operations. A frame is a data structure whose elements define the attributes of the operation. Each frame is associated with a particular board, and therefore, with a particular driver.

Frames help you create structured application programs. You set up the attributes of the operation in advance, using a separate function call for each attribute, and then start the operation at an appropriate point in your program. Frames are useful for operations that have many defining attributes, since providing a separate argument for each attribute could make a function's argument list unmanageably long.

You indicate that you want to perform an analog input operation by getting an available frame for the driver. The DAS-4300 Series Function Call Driver provides analog input frames, called A/D (analog-to-digital) frames. You use the **K_GetADFrame** function to access an available A/D frame. The driver returns a unique identifier for the frame; this name is called the frame handle.

You then specify the attributes of the operation by using setup functions to define the elements of the frame associated with the operation. For example, to specify the channel on which to perform an analog input operation, you would use the **K_SetChn** setup function.

You use the frame handle you specified when accessing the frame in all setup functions and other functions related to the operation. This ensures that you are defining the same operation.

When you are ready to perform the operation you have set up, you can start the operation by referencing the appropriate frame handle. Figure 3-1 illustrates the syntax of the interrupt-mode operation function **K_IntStart**.

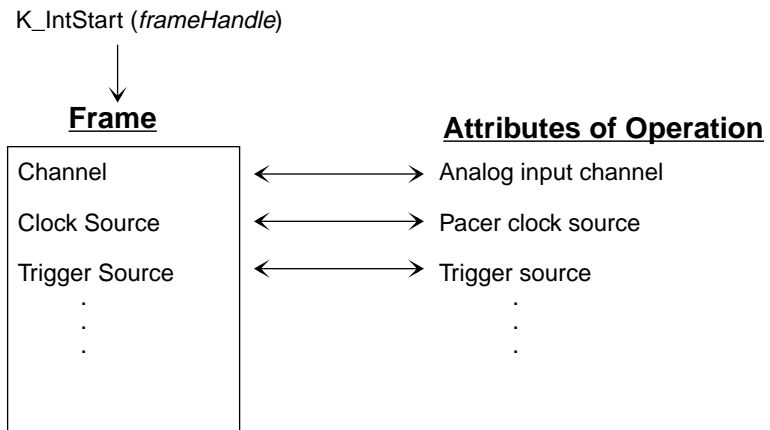


Figure 3-1. Interrupt-Mode Operation

If you want to perform an interrupt-mode operation and all frames have been accessed, you can use the **K_FreeFrame** function to free a frame that is no longer in use. You can then redefine the elements of the frame for the next operation.

When you access a frame, the elements are set to their default values. You can also use the **K_ClearFrame** function to reset all the elements of a frame to their default values.

Table 3-1 lists the elements of a DAS-4300 Series A/D frame. This table also lists the default value of each element and the setup function used to define each element.

Table 3-1. A/D Frame Elements

Element	Default Value	Setup Function
Buffer ¹	0 (NULL)	K_SetBuf
Number of Samples	0	K_SetBuf
Channel	0	K_SetChn
Input range	0 (± 200 mV)	K_SetG
Clock Source	Internal A/D pacer clock	K_SetClk
Pacer Clock Rate	0	K_SetClkRate
Trigger Source	Internal	K_SetTrig
Trigger Type	Digital	K_SetADTrig K_SetDITrig
Trigger Channel	0 (for analog trigger)	K_SetADTrig
	0 (for digital trigger)	Not applicable ²
Trigger Polarity	Positive	K_SetADTrig K_SetDITrig
Trigger Sensitivity	Edge (for analog)	Not applicable ²
	Edge (for digital)	Not applicable ²
Trigger Level	0	K_SetADTrig
About-Trigger Acquisition	Disabled	K_SetAboutTrig K_ClrAboutTrig ³
Post-Trigger Delay	0	K_SetPostTrigDelay

Notes

¹ This element must be set.

² The default value of this element cannot be changed.

³ Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

Note: The DAS-4300 Series Function Call Driver provides many other functions that are not related to controlling frames or defining the elements of frames. These functions include initialization functions, memory management functions, and miscellaneous functions.

For information about using the FCD functions in your application program, refer to the following sections of this chapter. For detailed information about the syntax of FCD functions, refer to Chapter 4.

Programming Overview

To write an application program using the DAS-4300 Series Function Call Driver, perform the following steps:

1. Define the application's requirements. Refer to Chapter 2 for a description of the board operations supported by the Function Call Driver and the functions that you can use to define each operation.
2. Write your application program. Refer to the following for additional information:
 - Preliminary Tasks, the next section, which describes the programming tasks that are common to all application programs.
 - Analog Input Programming Tasks on page 3-6, which describes operation-specific programming tasks and the sequence in which these tasks must be performed.
 - Chapter 4, which contains detailed descriptions of the FCD functions.
 - The example programs in the ASO-4300 software package. The FILES.TXT file in the installation directory lists and describes the example programs.
3. Compile and link the program. Refer to the language-specific programming information (page 3-11 to page 3-14 for C/C++ or page 3-18 for Visual Basic for Windows), or to the EXAMPLES.TXT file in the installation directory for compile and link statements and other language-specific considerations for each supported language.

Preliminary Tasks

For every Function Call Driver application program, you must perform the following preliminary tasks:

1. Include the function and variable type definition file for your language. This file is included in the ASO-4300 software package.
2. Declare and initialize program variables.
3. Use **K_OpenDriver** to initialize the driver.
4. Use **K_GetDevHandle** to specify the board you want to use and to initialize the board. If you are using more than one board, use the board initialization function once for each board you are using.

After completing the preliminary tasks, perform the analog input programming tasks described in the following section.

Analog Input Programming Tasks

For an interrupt-mode analog input operation, perform the following tasks:

1. Use the **K_GetADFrame** function to access an A/D frame.
2. Dimension a local array within your program's memory area or use the **K_IntAlloc** function to allocate a buffer dynamically outside your program's memory area.
3. *If you are programming in Visual Basic for Windows and are using a local array, use the **K_SetBufI** function to assign the starting address of the array and to specify the number of samples in the array.*

*Otherwise, use the **K_SetBuf** function to assign the starting address of the array or buffer and to specify the number of samples in the array or buffer.*

4. Use the appropriate setup functions to specify the attributes of the operation. The setup functions are listed in Table 3-2.

Note: When you access a new A/D frame, the frame elements contain default values. If the default value of a particular element is suitable for your operation, you do not have to use the setup function associated with that element. Refer to Table 3-1 on page 3-4 for a list of the default values of A/D frame elements.

Table 3-2. Setup Functions for Interrupt-Mode Analog Input Operations

Attribute	Setup Function(s)
Channel	K_SetChn
Input Range	K_SetG
Clock Source	K_SetClk
Pacer Clock Rate ¹	K_SetClkRate
Trigger Source	K_SetTrig
Trigger Type	K_SetADTrig K_SetDITrig
Trigger Channel	K_SetADTrig K_SetDITrig
Trigger Polarity	K_SetADTrig K_SetDITrig
Trigger Level	K_SetADTrig
About-Trigger Acquisition	K_SetAboutTrig K_ClrAboutTrig
Post-Trigger Delay	K_SetPostTrigDelay

Refer to Chapter 2 for background information about the setup functions; refer to Chapter 4 for detailed descriptions of the setup functions.

5. Use the **K_IntStart** function to start the interrupt-mode operation.
6. Use the **K_IntStatus** function to monitor the status of the interrupt-mode operation.

7. Use the **K_IntStop** function to stop the interrupt-mode operation when the appropriate number of samples has been acquired.
8. *If you are programming in Visual Basic for Windows and you are using a dynamically allocated memory buffer, use the **K_MoveBufToArray** function to transfer the acquired data from the allocated buffer to a local array that your program can use.*
9. *If you used a dynamically allocated memory buffer, use the **K_IntFree** function to free the memory.*

C/C++ Programming Information

The following sections contain information you need to dimension an array or dynamically allocate a memory buffer when programming in C or C++, as well as language-specific information for Microsoft C/C++ and Borland C/C++ for DOS and Windows.

Note: When programming in C/C++, proper typecasting may be required to avoid C/C++ type-mismatch warnings.

Dimensioning and Assigning a Local Array

You can use a single, local array for an interrupt-mode analog input operation. The following code fragment illustrates how to dimension an array of 8,192 samples for the frame defined by hFrame and how to use **K_SetBuf** to assign the starting address of the array.

```
. . .  
int Data[8192];    //Dimension array of 8,192 samples  
. . .  
wDasErr = K_SetBuf (hFrame, Data, 8192);  
. . .
```

Refer to the example programs on disk for more information.

Dynamically Allocating and Assigning a Memory Buffer

This section provides code fragments that describe how to dynamically allocate and assign a memory buffer when programming in C or C++ and how to access the data in the buffer. Refer to the example programs on disk for more information.

Note: If you are programming in Windows Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you install the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer; refer to the *DAS-4300 Series User's Guide* for information on the Keithley Memory Manager.

Allocating a Memory Buffer

You can use a single, dynamically allocated memory buffer for an interrupt-mode analog input operation. The following code fragment illustrates how to use **K_IntAlloc** to allocate a buffer of size Samples for the frame defined by hFrame and how to use **K_SetBuf** to assign the starting address of the buffer.

```
. . . .
void far *AcqBuf;           //Declare pointer to buffer
WORD hMem;                 //Declare word for memory handle
. . . .
wDasErr = K_IntAlloc (hFrame, Samples, &AcqBuf, &hMem);
wDasErr = K_SetBuf (hFrame, AcqBuf, Samples);
. . . .
```

The following code illustrates how to use **K_IntFree** to later free the allocated buffer, using the memory handle stored by **K_IntAlloc**.

```
. . . .
wDasErr = K_IntFree (hMem);
. . . .
```

Accessing the Data

You access the data stored in a dynamically allocated buffer through C/C++ pointer indirection. For example, assume that you want to display the first 10 samples of the buffer described in the previous section (AcqBuf). The following code fragment illustrates how to access and display the data.

```
. . .
int huge *pData;           //Declare a pointer called pData
. . .
pData = (int huge *) AcqBuf; //Assign pData to buffer
for (i = 0; i < 10; i++)
    printf ("Sample #%d %X", i, *(pData+i));
. . .
```

Note: Declaring pData as a huge pointer allows the program to directly access all data within the computer's memory buffer, regardless of the buffer size.

Handling Errors

It is recommended that you always check the returned value (wDasErr in the previous examples) for possible errors. The following code fragment illustrates how to check the returned value of the **K_GetDevHandle** function.

```
. . .
if ((wDASErr = K_GetDevHandle (hDrv, BoardNum, &hDev)) != 0)
{
    printf ("Error %X during K_GetDevHandle", wDASErr);
    exit (1);
}
. . .
```

Programming in Microsoft C/C++ (for DOS)

To program in Microsoft C/C++ (for DOS), you need the following files; these files are provided in the ASO-4300 software package.

File	Description
DAS4300.LIB	Linkable driver
DASRFACE.LIB	Linkable driver
DASDECL.H	Include file when compiling in C (.c programs)
DASDECL.HPP	Include file when compiling in C++ (.cpp programs).
USE4300.OBJ	Linkable object

To create an executable file in Microsoft C/C++ (for DOS), use the following compile and link statements. Note that *filename* indicates the name of your application program.

Type of Compile	Compile and Link Statements ¹
C	CL /c <i>filename.c</i> LINK <i>filename</i> +use4300.obj,,,das4300+dasrface;
C++	CL /c <i>filename.cpp</i> LINK <i>filename</i> +use4300.obj,,,das4300+dasrface;

Notes

¹ These statements assume a large memory model; in DOS, only the large memory model is acceptable.

Programming in Microsoft C/C++ (for Windows)

To program in Microsoft C/C++ (for Windows), including Microsoft Visual C++, you need the following files; these files are provided in the ASO-4300 software package.

File	Description
DASHELL.DLL	Dynamic Link Library
DASSUPRT.DLL	Dynamic Link Library
DAS4300.DLL	Dynamic Link Library
DASDECL.H	Include file when compiling in C (.c programs)
DASDECL.HPP	Include file when compiling in C++ (.cpp programs)
DASIMP.LIB	DAS Shell Imports

To create an executable file in Microsoft C/C++ (for Windows), use the following compile and link statements. Note that *filename* indicates the name of your application program.

Type of Compile	Compile and Link Statements
C	CL /c <i>filename.c</i> LINK <i>filename</i> ,,dasimp, <i>filename.def</i> ; RC -r <i>filename.rc</i> RC -30 <i>filename.res</i>
C++	CL /c <i>filename.cpp</i> LINK <i>filename</i> ,,dasimp, <i>filename.def</i> ; RC -r <i>filename.rc</i> RC -30 <i>filename.res</i>

To create an executable file in the Microsoft C/C++ (for Windows) environment, perform the following steps:

1. Create a project file by choosing New from the Project menu.
2. Add all necessary files to the project make file by choosing Edit from the Project menu. Make sure that you include *filename.c* (or *filename.cpp*), *filename.rc*, *filename.def*, and DASIMP.LIB, where *filename* indicates the name of your application program.
3. From the Project menu, choose Rebuild All FILENAME.EXE to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Programming in Borland C/C++ (for DOS)

To program in Borland C/C++ (for DOS), you need the following files; these files are provided in the ASO-4300 software package.

File	Description
DAS4300.LIB	Linkable driver
DASRFACE.LIB	Linkable driver
DASDECL.H	Include file when compiling in C (.c programs)
DASDECL.HPP	Include file when compiling in C++ (.cpp programs)
USE4300.OBJ	Linkable object

To create an executable file in Borland C/C++ (for DOS), use the following compile and link statements. Note that *filename* indicates the name of your application program.

Type of Compile	Compile and Link Statements ¹
C	BCC -ml <i>filename.c</i> use4300.obj das4300.lib dasrface.lib
C++	BCC -ml <i>filename.cpp</i> use4300.obj das4300.lib dasrface.lib

Notes

¹ These statements assume a large memory model; in DOS, only the large memory model is acceptable.

Programming in Borland C/C++ (for Windows)

To program in Borland C/C++ (for Windows), you need the following files; these files are provided in the ASO-4300 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library
DASSUPRT.DLL	Dynamic Link Library
DAS4300.DLL	Dynamic Link Library
DASDECL.H	Include file when compiling in C (.c programs)
DASDECL.HPP	Include file when compiling in C++ (.cpp programs)
DASIMP.LIB	DAS Shell Imports

To create an executable file in Borland C/C++ (for Windows), use the following compile and link statements. Note that *filename* indicates the name of your application program.

Type of Compile	Compile and Link Statements
C	BCC -c <i>filename.c</i> TLINK <i>filename,,,dasimp, filename.def</i> ; BRC -r <i>filename.rc</i> BRC -30 <i>filename.res</i>
C++	BCC -c <i>filename.cpp</i> TLINK <i>filename,,,dasimp, filename.def</i> ; BRC -r <i>filename.rc</i> BRC -30 <i>filename.res</i>

To create an executable file in the Borland C/C++ (for Windows) environment, perform the following steps:

1. Create a project file by choosing New from the Project menu.
2. Inside the Project window, select the project name and click on the right mouse button.
3. Select the Add node option and add all necessary files to the project make file. Make sure that you include *filename.c* (or *filename.cpp*),

filename.rc, *filename.def*, and *DASIMP.LIB*, where *filename* indicates the name of your application program.

4. From the Options menu, select Project.
5. From the Project Options dialog box, select Linker\General and make sure that you turn OFF both the Case sensitive link and Case sensitive exports and imports options.
6. From the Project menu, choose Build All to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Microsoft Visual Basic for Windows Programming Information

The following sections contain information you need to dimension an array or dynamically allocate a memory buffer when programming in Microsoft Visual Basic for Windows, as well as language-specific information for Microsoft Visual Basic for Windows.

Dimensioning and Assigning a Local Array

You can use a single, local array for an interrupt-mode analog input operation. The following code fragment illustrates how to dimension an array of 8K samples for the frame defined by *hFrame* and how to use **K_SetBufI** to assign the starting address of the array.

```
. . . .
Global Data(8191) As Integer      ' Allocate array
. . . .
wDasErr = K_SetBufI (hFrame, Data(0), 8192)
. . . .
```

Refer to the example programs on disk for more information.

Dynamically Allocating and Assigning a Memory Buffer

This section provides code fragments that describe how to dynamically allocate and assign a memory buffer and how to access the data in the buffer when programming in Microsoft Visual Basic for Windows. Refer to the example programs on disk for more information.

Note: If you are programming in Windows Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer. Refer to your *DAS-4300 Series User's Guide* for more information about the Keithley Memory Manager.

Allocating a Memory Buffer

You can use a single, dynamically allocated memory buffer for an interrupt-mode analog input operation. The following code fragment illustrates how to use **K_IntAlloc** to allocate a buffer of size `Samples` for the frame defined by `hFrame` and how to use **K_SetBuf** to assign the starting address of the buffer.

```
. . . .
Global AcqBuf As Long    ' Declare pointer to buffer
Global hMem As Integer  ' Declare integer for memory handle
. . . .
wDasErr = K_IntAlloc (hFrame, Samples, AcqBuf, hMem)
wDasErr = K_SetBuf (hFrame, AcqBuf, Samples)
. . . .
```

The following code illustrates how to use **K_IntFree** to later free the allocated buffer, using the memory handle stored by **K_IntAlloc**.

```
. . . .
wDasErr = K_IntFree (hMem)
. . . .
```


Accessing the Data

In Microsoft Visual Basic for Windows, you cannot directly access analog input samples stored in a dynamically allocated memory buffer. You must use **K_MoveBufToArray** to move a subset (up to 32,766 samples) of the data into a local array as required. The following code fragment illustrates how to move 100 samples from the buffer described in the previous section (AcqBuf) to a local array.

```
. . .
Dim Buffer(1000) As Integer    ' Declare local array
. . .
wDasErr = K_MoveBufToArray (Buffer(0), AcqBuf, 100)
. . .
```

Handling Errors

It is recommended that you always check the returned value (wDasErr in the previous examples) for possible errors. The following code fragment illustrates how to check the returned value of the **K_GetDevHandle** function:

```
. . .
wDASErr = K_GetDevHandle (hDrv, BoardNum, hDev)
If (wDASErr <> 0) Then
    MsgBox "K_GetDevHandle Error: " + Hex$ (wDASErr),
        MB_ICONSTOP, "DAS-4300 SERIES ERROR"
    End
End If
. . .
```

Programming in Microsoft Visual Basic for Windows

To program in Microsoft Visual Basic for Windows, you need the following files; these files are provided in the ASO-4300 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library
DASSUPRT.DLL	Dynamic Link Library
DAS4300.DLL	Dynamic Link Library
DASDECL.BAS	Include file; must be added to the Project List

To create an executable file from the Microsoft Visual Basic for Windows environment, choose Make EXE File from the File menu.

4

Function Reference

The FCD functions are organized into the following groups:

- Initialization functions
- Operation functions
- Frame management functions
- Memory management functions
- Buffer address functions
- Channel and gain functions
- Clock functions
- Trigger functions
- Miscellaneous functions

The particular functions associated with each function group are presented in Table 4-1. The remainder of the chapter presents detailed descriptions of all the FCD functions, arranged in alphabetical order.

Table 4-1. Functions

Function Type	Function Name	Page Number
Initialization	K_OpenDriver	page 4-31
	K_CloseDriver	page 4-6
	K_GetDevHandle	page 4-15
	K_FreeDevHandle	page 4-9
	K_DASDevInit	page 4-8
Operation	K_IntStart	page 4-24
	K_IntStatus	page 4-25
	K_IntStop	page 4-28
Frame Management	K_GetADFrame	page 4-11
	K_FreeFrame	page 4-10
	K_ClearFrame	page 4-5
Memory Management	K_IntAlloc	page 4-21
	K_IntFree	page 4-23
	K_MoveBufToArray	page 4-30
Buffer Address	K_SetBuf	page 4-36
	K_SetBufI	page 4-38
Channel and Gain	K_SetChn	page 4-40
	K_SetG	page 4-46
Clock	K_SetClk	page 4-41
	K_SetClkRate	page 4-42
	K_GetClkRate	page 4-13

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Trigger	K_SetTrig	page 4-50
	K_SetADTrig	page 4-34
	K_SetDITrig	page 4-44
	K_SetAboutTrig	page 4-33
	K_ClrAboutTrig	page 4-7
	K_SetPostTrigDelay	page 4-48
Miscellaneous	K_GetErrMsg	page 4-17
	K_GetVer	page 4-19
	K_GetShellVer	page 4-18

Keep the following conventions in mind throughout this chapter:

- The data types **DWORD**, **WORD**, and **BYTE** are defined in the language-specific include files.
- Variable names are shown in italics.
- The return value for all DAS-4300 Series FCD functions is the error/status code. A value of 0 indicates that the function executed successfully. A non-zero value indicates that an error occurred. Refer to Appendix A for more information.
- The description shows the prototype for the function.
- In the Usage section, the variables are not defined. It is assumed that the variables are defined as shown in the prototype.

The name of each function argument in the Prototype and Usage sections includes a prefix that indicates the associated data type. These prefixes are described in Table 4-2.

Table 4-2. Data Type Prefixes

Prefix	Data Type	Comments
sz	Pointer to string terminated by zero	This data type is typically used for variables that specify the driver's configuration file name.
h	Handle to device, frame, and memory block	This data type is used for handle-type variables. You declare handle-type variables in your program as long or DWORD, depending on the language you are using. The actual variable is passed to the driver by value.
ph	Pointer to a handle-type variable	This data type is used when calling the FCD functions to get a driver handle, a frame handle, or a memory handle. The actual variable is passed to the driver by reference.
p	Pointer to a variable	This data type is used for pointers to all types of variables, except handles (h). It is typically used when passing a parameter of any type to the driver by reference.
n	Number value	This data type is used when passing a number, typically a byte, to the driver by value.
w	16-bit word	This data type is typically used when passing an unsigned integer to the driver by value.
a	Array	This data type is typically used in conjunction with other prefixes listed here; for example, <i>anVar</i> denotes an array of numbers.
f	Float	This data type denotes a single-precision floating-point number.
d	Double	This data type denotes a double-precision floating-point number.
dw	32-bit double word	This data type is typically used when passing an unsigned long to the driver by value.

K_ClearFrame

Purpose	Sets the elements of a frame to their default values.
Prototype	C/C++ DASErr far pascal K_ClearFrame (DWORD <i>hFrame</i>); Visual Basic for Windows Declare Function K_ClearFrame Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long) As Integer
Parameters	<i>hFrame</i> Handle to the frame that defines the operation.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function sets the elements of the frame specified by <i>hFrame</i> to their default values. Refer to Table 3-1 on page 3-4 for the default values of the elements of an A/D frame.
See Also	K_GetADFrame
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_ClearFrame (hAD);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) <pre>... wDasErr = K_ClearFrame (hAD)</pre>

K_CloseDriver

Purpose	Closes a previously initialized Keithley DAS Function Call Driver.
Prototype	C/C++ DASErr far pascal K_CloseDriver (DWORD <i>hDrv</i>); Visual Basic for Windows Declare Function K_CloseDriver Lib "DASSHELL.DLL" (ByVal <i>hDrv</i> As Long) As Integer
Parameters	<i>hDrv</i> Driver handle you want to free.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function frees the driver handle specified by <i>hDrv</i> and closes the associated use of the Function Call Driver. This function also frees all device handles and frame handles associated with <i>hDrv</i> . If <i>hDrv</i> is the last driver handle specified for the Function Call Driver, the driver is shut down (for all languages) and unloaded (for Windows-based languages only).
See Also	K_FreeDevHandle
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_CloseDriver (hDrv);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) <pre>... wDasErr = K_CloseDriver (hDrv)</pre>

K_ClrAboutTrig

Purpose	Disables the about trigger for an analog input operation.
Prototype	C/C++ DASErr far pascal K_ClrAboutTrig (DWORD <i>hFrame</i>); Visual Basic for Windows Declare Function K_ClrAboutTrig Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long) As Integer
Parameters	<i>hFrame</i> Handle to the frame that defines the operation.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function disables the about trigger for the operation defined by <i>hFrame</i> . K_GetADFrame and K_ClearFrame also disable the about trigger.
See Also	K_ClearFrame, K_GetADFrame, K_SetAboutTrig
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_ClrAboutTrig (hAD);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) ... wDasErr = K_ClrAboutTrig (hAD)

K_DASDevInit

Purpose	Reinitializes a board.
Prototype	C/C++ DASErr far pascal K_DASDevInit (DWORD <i>hDev</i>); Visual Basic for Windows Declare Function K_DASDevInit Lib "DASSHELL.DLL" (ByVal <i>hDev</i> As Long) As Integer
Parameters	<i>hDev</i> Handle associated with the board.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function stops all current operations on the board specified by <i>hDev</i> and verifies that the board identified by the device handle is the board specified in the configuration file associated with the board.
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_DASDevInit (hDev);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) <pre>... wDasErr = K_DASDevInit (hDev)</pre>

K_FreeDevHandle

Purpose	Frees a previously specified device handle.
Prototype	C/C++ DASErr far pascal K_FreeDevHandle (DWORD <i>hDev</i>); Visual Basic for Windows Declare Function K_FreeDevHandle Lib "DASSHELL.DLL" (ByVal <i>hDev</i> As Long) As Integer
Parameters	<i>hDev</i> Device handle you want to free.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function frees the device handle specified by <i>hDev</i> as well as all frame handles associated with <i>hDev</i> .
See Also	K_GetDevHandle
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_FreeDevHandle (hDev);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) ... wDasErr = K_FreeDevHandle (hDev)

K_FreeFrame

Purpose	Frees a frame.
Prototype	C/C++ DASErr far pascal K_FreeFrame (DWORD <i>hFrame</i>); Visual Basic for Windows Declare Function K_FreeFrame Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long) As Integer
Parameters	<i>hFrame</i> Handle to frame you want to free.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function frees the frame specified by <i>hFrame</i> , making the frame available for another operation.
See Also	K_GetADFrame
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_FreeFrame (hAD);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) ... wDasErr = K_FreeFrame (hAD)

K_GetADFrame

Purpose	Accesses an A/D frame for an analog input operation.
Prototype	C/C++ DASErr far pascal K_GetADFrame (DWORD <i>hDev</i> , DWORD far * <i>phFrame</i>); Visual Basic for Windows Declare Function K_GetADFrame Lib "DASHELL.DLL" (ByVal <i>hDev</i> As Long, <i>phFrame</i> As Long) As Integer
Parameters	<i>hDev</i> Handle associated with the board. <i>phFrame</i> Handle to the frame that defines the operation.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function specifies that you want to perform an interrupt-mode analog input operation on the board specified by <i>hDev</i> , and accesses an available A/D frame with the handle <i>phFrame</i> . The frame is initialized to its default settings; the default settings are given in Table 3-1 on page 3-4. The value stored in <i>phFrame</i> is intended to be used exclusively as an argument to functions that require a frame handle. Your program should not modify the value stored in <i>phFrame</i> .
See Also	K_ClearFrame, K_FreeFrame
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... DWORD hAD; ... wDasErr = K_GetADFrame (hDev, &hAD);</pre>

K_GetADFrame (cont.)

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
Global hAD As Long  
...  
wDasErr = K_GetADFrame (hDev, hAD)
```

K_GetClkRate

Purpose Gets a count value that determines the time between conversions for the internal pacer clock.

Prototype **C/C++**
DASErr far pascal K_GetClkRate (DWORD *hFrame*,
DWORD far **pRate*);

Visual Basic for Windows

Declare Function K_GetClkRate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pRate* As Long) As Integer

Parameters *hFrame* Handle to the frame that defines the operation.
pRate Count value.
Value stored: **20** to **25600**, described as follows:

Conversion Rate	Sample Period	Count Value
1 Gsamples/s	1 ns	20
500 Msamples/s	2 ns	40
250 Msamples/s	4 ns	80
100 Msamples/s	10 ns	200
50 Msamples/s	20 ns	400
25 Msamples/s	40 ns	800
12.5 Msamples/s	80 ns	1600
6.25 Msamples/s	160 ns	3200
3.125 Msamples/s	320 ns	6400
1.5625 Msamples/s	640 ns	12800
0.78250 Msamples/s	1280 ns	25600

K_GetClkRate (cont.)

Return Value	Error/status code. Refer to Appendix A.
Remarks	<p>For the operation defined by <i>hFrame</i>, this function stores the number of clock ticks between conversions in <i>pRate</i>.</p> <p>The <i>pRate</i> variable contains the value of the Pacer Clock Rate element.</p>
See Also	K_SetClkRate
Usage	<p>C/C++</p> <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... DWORD dwRate; ... wDasErr = K_GetClkRate (hAD, &dwRate);</pre> <p>Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project)</p> <pre>... Global dwRate As Long ... wDasErr = K_GetClkRate (hAD, dwRate)</pre>

K_GetDevHandle

Purpose	Initializes any Keithley DAS board.	
Prototype	C/C++ DASErr far pascal K_GetDevHandle (DWORD <i>hDrv</i> , WORD <i>nBoardNum</i> , DWORD far * <i>pDev</i>); Visual Basic for Windows Declare Function K_GetDevHandle Lib "DASHELL.DLL" (ByVal <i>hDrv</i> As Long, ByVal <i>nBoardNum</i> As Integer, <i>pDev</i> As Long) As Integer	
Parameters	<i>hDrv</i>	Driver handle of the associated Function Call Driver.
	<i>nBoardNum</i>	Board number. Valid values: 0 to 1
	<i>pDev</i>	Handle associated with the board.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	This function initializes the board associated with <i>hDrv</i> and specified by <i>nBoardNum</i> , and stores the device handle of the specified board in <i>pDev</i> . The value stored in <i>pDev</i> is intended to be used exclusively as an argument to functions that require a device handle. Your program should not modify the value stored in <i>pDev</i> .	
See Also	K_FreeDevHandle	
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... DWORD hDev; ... wDasErr = K_GetDevHandle (hDrv, 0, &hDev);</pre>	

K_GetDevHandle (cont.)

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
Global hDev As Long  
...  
wDasErr = K_GetDevHandle (hDrv, 0, hDev)
```

K_GetErrMsg

Purpose	Gets the address of an error message string.	
Prototype	C/C++ DASErr far pascal K_GetErrMsg (DWORD <i>hDev</i> , short <i>nDASErr</i> , char far * far * <i>pErrMsg</i>); Visual Basic for Windows Not supported	
Parameters	<i>hDev</i>	Handle associated with the board.
	<i>nDASErr</i>	Error message number.
	<i>pErrMsg</i>	Address of error message string.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the board specified by <i>hDev</i> , this function stores the address of the string corresponding to error message number <i>nDASErr</i> in <i>pErrMsg</i> . Refer to page 2-3 for more information about error handling. Refer to Appendix A for a list of error codes and their meanings.	
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... char far *pErrMsg; ... wDasErr = K_GetErrMsg (hDev, wDASErr, &pErrMsg);</pre>	

K_GetShellVer

Purpose	Gets the current DAS shell version.	
Prototype	C/C++ DASErr far pascal K_GetShellVer (WORD far * <i>pVersion</i>);	
	Visual Basic for Windows Declare Function K_GetShellVer Lib "DASSHELL.DLL" (<i>pVersion</i> As Integer) As Integer	
Parameters	<i>pVersion</i>	A word value containing the major and minor version numbers of the DAS shell.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	This function stores the current DAS shell version in <i>pVersion</i> . To obtain the major version number of the DAS shell, divide <i>pVersion</i> by 256. To obtain the minor version number of the DAS shell, perform a Boolean AND operation with <i>pVersion</i> and 255 (0FF hex).	

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
WORD wShellVer;
wDasErr = K_GetShellVer (&wShellVer);
printf ("Shell Ver %d.%d", wShellVer >> 8, wShellVer & 0xff);
```

Visual Basic for Windows

(Add *DASDECL.BAS* to your project)

```
...
Global wShellVer As Integer
...
wDasErr = K_GetShellVer (wShellVer)
ShellVer$ = LTRIM$(STR$(INT(wShellVer / 256))) + "." + :
    LTRIM$(STR$(wShellVer AND &HFF))
PRINT "Driver Ver: " + ShellVer$
```

Purpose	Gets revision numbers.	
Prototype	C/C++ DASErr far pascal K_GetVer (DWORD <i>hDev</i> , short far * <i>pSpecVer</i> , short far * <i>pDrvVer</i>);	
	Visual Basic for Windows Declare Function K_GetVer Lib "DASSHELL.DLL" (ByVal <i>hDev</i> As Long, <i>pSpecVer</i> As Integer, <i>pDrvVer</i> As Integer) As Integer	
Parameters	<i>hDev</i>	Handle associated with the board.
	<i>pSpecVer</i>	Revision number of the Keithley DAS Driver Specification to which the driver conforms.
	<i>pDrvVer</i>	Driver version number.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the board specified by <i>hDev</i> , this function stores the revision number of the DAS-4300 Series Function Call Driver in <i>pDrvVer</i> and the revision number of the driver specification in <i>pSpecVer</i> . The values stored in <i>pSpecVer</i> and <i>pDrvVer</i> are two-byte (16-bit) integers; the high byte of each contains the major revision level and the low byte of each contains the minor revision level. For example, if the driver version number is 2.1, the major revision level is 2 and the minor revision level is 10; therefore, the high byte of <i>pDrvVer</i> contains the value of 2 (512) and the low byte of <i>pDrvVer</i> contains the value of 10 ; the value of both bytes is 522. To extract the major and minor revision levels from the value stored in <i>pDrvVer</i> or <i>pSpecVer</i> , use the following equations:	

$$\text{major revision level} = \text{Integer portion of} \left(\frac{\text{returned value}}{256} \right)$$

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

K_GetVer (cont.)

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
short nSpecVer, nDrvVer;
...
wDasErr = K_GetVer (hDev, &nSpecVer, &nDrvVer);
printf ("Driver Ver %d.%d", nDrvVer >> 8, nDrvVer & 0xff);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
Global nSpecVer As Integer
Global nDrvVer As Integer
...
wDasErr = K_GetVer (hDev, nSpecVer, nDrvVer)
DrvVer$ = LTRIM$(STR$(INT(nDrvVer / 256))) + "." + :
    LTRIM$(STR$(nDrvVer AND &HFF))
PRINT "Driver Ver: " + DrvVer$
```

Purpose	Dynamically allocates a buffer for an interrupt-mode operation.	
Prototype	C/C++ DASErr far pascal K_IntAlloc (DWORD <i>hFrame</i> , DWORD <i>dwSamples</i> , void far * far * <i>pBuf</i> , WORD far * <i>pMem</i>);	
	Visual Basic for Windows Declare Function K_IntAlloc Lib "DASHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>dwSamples</i> As Long, <i>pBuf</i> As Long, <i>pMem</i> As Integer) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>dwSamples</i>	Number of samples. Valid values: 1 to 8192
	<i>pBuf</i>	Starting address of the allocated buffer.
	<i>pMem</i>	Handle associated with the allocated buffer.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the operation defined by <i>hFrame</i> , this function dynamically allocates a buffer of the size specified by <i>dwSamples</i> , and stores the starting address of the buffer in <i>pBuf</i> and the handle of the buffer in <i>pMem</i> . The data in the allocated buffer is stored as counts. Refer to Appendix B for information on converting a count value to voltage.	
See Also	K_IntFree, K_SetBuf	
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... void far *pBuf; // Pointer to allocated buffer WORD hMem; // Memory Handle to buffer ... wDasErr = K_IntAlloc (hAD, 8192, &pBuf, &hMem);</pre>	

K_IntAlloc (cont.)

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
Global pBuf As Long  
Global hMem As Integer  
...  
wDasErr = K_IntAlloc (hAD, 8192, pBuf, hMem)
```


Purpose Frees a buffer dynamically allocated for an interrupt-mode operation.

Prototype **C/C++**
DASErr far pascal K_IntFree (WORD *hMem*);

Visual Basic for Windows
Declare Function K_IntFree Lib "DASSHELL.DLL"
(ByVal *hMem* As Integer) As Integer

Parameters *hMem* Handle to interrupt buffer.

Return Value Error/status code. Refer to Appendix A.

Remarks This function frees the buffer specified by *hMem*; the buffer was previously allocated dynamically using **K_IntAlloc**.

See Also K_IntAlloc

Usage **C/C++**

```
#include "DASDECL.H" // Use "DASDECL.HPP for C++
...
wDasErr = K_IntFree (hMem);
```

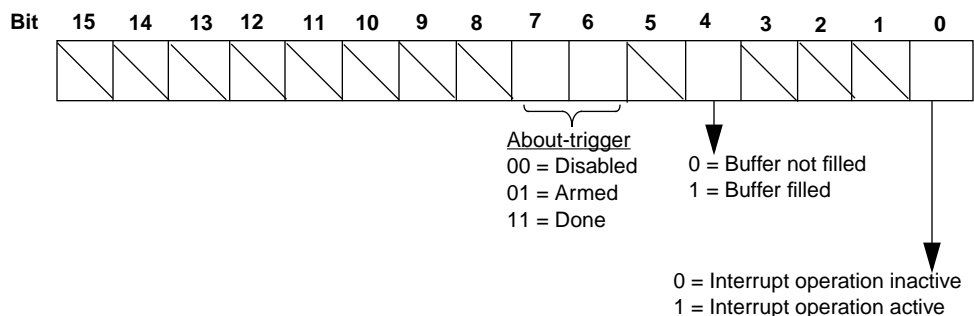
Visual Basic for Windows
(Add *DASDECL.BAS* to your project)
...
wDasErr = K_IntFree (hMem)

K_IntStart

Purpose	Starts an interrupt-mode operation.
Prototype	C/C++ DASErr far pascal K_IntStart (DWORD <i>hFrame</i>); Visual Basic for Windows Declare Function K_IntStart Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long) As Integer
Parameters	<i>hFrame</i> Handle to the frame that defines the operation.
Return Value	Error/status code. Refer to Appendix A.
Remarks	This function starts the interrupt-mode operation defined by <i>hFrame</i> . Refer to page 3-6 for a discussion of the programming tasks associated with interrupt-mode operations.
See Also	K_IntStatus, K_IntStop
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_IntStart (hAD);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) <pre>... wDasErr = K_IntStart (hAD)</pre>

K_IntStatus

Purpose	Gets the status of an interrupt-mode operation.	
Prototype	C/C++ DASErr far pascal K_IntStatus (DWORD <i>hFrame</i> , short far * <i>pStatus</i> , DWORD far * <i>pCount</i>);	
	Visual Basic for Windows Declare Function K_IntStatus Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long, <i>pStatus</i> As Integer, <i>pCount</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>pStatus</i>	Status of interrupt operation; see Remarks below for value stored.
	<i>pCount</i>	Current number of samples transferred into the array or buffer.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the interrupt-mode operation defined by <i>hFrame</i> , this function stores the status in <i>pStatus</i> and the current number of samples transferred into the array or buffer in <i>pCount</i> . The value stored in <i>pStatus</i> depends on the settings in the Status word, as shown below:	



K_IntStatus (cont.)

Figure 4-1. Status Word Settings

The bits are described as follows:

- Bit 0: This bit indicates whether an interrupt-mode operation is in progress.
- Bits 1 to 3: Not used.
- Bit 4: This bit is set when the array or buffer that is assigned to the active operation has been filled with data.
- Bits 6 and 7: These bits indicate the state of the about trigger.
- Bits 8 to 15: Not used.

See Also K_IntStart, K_IntStop

K_IntStatus (cont.)

Usage

C/C++

```
#include "DASDECL.H" // Use "DASDECL.HPP for C++
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_IntStatus (hAD, &wStatus, &dwCount);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_IntStatus (hAD, wStatus, dwCount)
```

K_IntStop

Purpose	Stops an interrupt-mode operation.	
Prototype	C/C++ DASErr far pascal K_IntStop (DWORD <i>hFrame</i> , short far <i>*pStatus</i> , DWORD far <i>*pCount</i>); Visual Basic for Windows Declare Function K_IntStop Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long, <i>pStatus</i> As Integer, <i>pCount</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>pStatus</i>	Status of interrupt operation. Refer to page 4-26 for more information on the status word returned.
	<i>pCount</i>	Current number of samples transferred into the array or buffer.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	This function stops the board from acquiring data, disables the interrupt-mode operation, and returns the status of the operation when your program called this function. No data is transferred into the array or buffer in computer memory. If you are using an external start or about trigger, call this function if the trigger event does not occur.	
See Also	K_IntStart, K_IntStatus	
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... WORD wStatus; DWORD dwCount; ... wDasErr = K_IntStop (hAD, &wStatus, &dwCount);</pre>	

K_IntStop (cont.)

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
Global wStatus As Integer  
Global dwCount As Long  
...  
wDasErr = K_IntStop (hAD, wStatus, dwCount)
```

K_MoveBufToArray

Purpose Transfers data from a buffer allocated through **K_IntAlloc** to a locally dimensioned array.

Prototype **C/C++**
Not supported

Visual Basic for Windows

Declare Function K_MoveBufToArray Lib "DASSHELL.DLL" Alias "K_MoveDataBuf" (*pDest* As Integer, ByVal *pSource* As Long, ByVal *nCount* As Integer) As Integer

Parameters

<i>pDest</i>	Address of destination array.
<i>pSource</i>	Address of source buffer.
<i>nCount</i>	Number of samples to transfer. Value values: 1 to 8192

Return Value Error/status code. Refer to Appendix A.

Remarks This function transfers the number of samples specified by *nCount* from the buffer at address *pSource* to the array at address *pDest*.
In Visual Basic for Windows, the buffer allocated through **K_IntAlloc** is not accessible to your program; you must use **K_MoveBufToArray** to move the data from the allocated buffer to the program's local array.

See Also K_IntAlloc

Usage **Visual Basic for Windows**
(Add *DASDECL.BAS* to your project)

```
...  
wDasErr = K_IntAlloc (hAD, 8192, pBuf, hMem)  
...  
wDasErr = K_MoveBufToArray (ADArray(0), pBuf, 8192)
```


K_OpenDriver

Purpose	Initializes any Keithley DAS Function Call Driver.	
Prototype	C/C++ DASErr far pascal K_OpenDriver (char far * <i>szDrvName</i> , char far * <i>szCfgName</i> , DWORD far * <i>pDrv</i>);	
	Visual Basic for Windows Declare Function K_OpenDriver Lib "DASSHELL.DLL" (ByVal <i>szDrvName</i> As String, ByVal <i>szCfgName</i> As String, <i>pDrv</i> As Long) As Integer	
Parameters	<i>szDrvName</i>	Board name. Valid value: "DAS4300" (for DAS-4301/8K boards)
	<i>szCfgName</i>	Driver configuration file. Valid values: The name of a configuration file; 0 if driver has already been opened
	<i>pDrv</i>	Handle associated with the driver.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	<p>This function initializes the DAS-4300 Series Function Call Driver according to the information in the configuration file specified by <i>szCfgName</i>, and stores the driver handle in <i>pDrv</i>.</p> <p>You can use this function to initialize the Function Call Driver associated with any Keithley MetraByte DAS board. For DAS-4301/8K boards, the string stored in <i>szDrvName</i> must be DAS4300. Refer to other Function Call Driver user's guides for the appropriate string to store in <i>szDrvName</i> for other Keithley MetraByte DAS boards.</p> <p>The value stored in <i>pDrv</i> is intended to be used exclusively as an argument to functions that require a driver handle. Your program should not modify the value stored in <i>pDrv</i>.</p> <p>You create a configuration file using the CFG4300.EXE utility. Refer to your <i>DAS-4300 Series User's Guide</i> for more information.</p>	

K_OpenDriver (cont.)

If *szCfgName* = 0, **K_OpenDriver** checks whether the driver has already been opened and linked to a configuration file and if it has, uses the current configuration; this is useful in the Windows environment.

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
DWORD hDrv;
...
wDasErr = K_OpenDriver ("DAS4300", "DAS4300.CFG", &hDrv);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
DIM hDrv As Long
...
wDasErr = K_OpenDriver("DAS4300", "DAS4300.CFG", hDrv)
```

K_SetAboutTrig

Purpose	Enables the about trigger and specifies the number of post-trigger samples.	
Prototype	C/C++ DASErr far pascal K_SetAboutTrig (DWORD <i>hFrame</i> , DWORD <i>dwSamples</i>);	
	Visual Basic for Windows Declare Function K_SetAboutTrig Lib "DASHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>dwSamples</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>dwSamples</i>	Number of post-trigger samples. Valid values: 0 to 8191
Return Value	Error/status code. Refer to Appendix A.	
Remarks	This function enables the about trigger and specifies the number of post-trigger samples in <i>dwSamples</i> . At least one pre-trigger sample must be in the buffer. Note that you cannot use an about trigger with an external start trigger.	
See Also	K_ClrAboutTrig	
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_SetAboutTrig (hAD, 100);</pre>	
	Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) ... wDasErr = K_SetAboutTrig (hAD, 100)	

K_SetADTrig

Purpose	Sets up an external analog start or about trigger.	
Prototype	C/C++ DASErr far pascal K_SetADTrig (DWORD <i>hFrame</i> , short <i>nOpt</i> , short <i>nChan</i> , DWORD <i>dwLevel</i>); Visual Basic for Windows Declare Function K_SetADTrig Lib "DASHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>nOpt</i> As Integer, ByVal <i>nChan</i> As Integer, ByVal <i>dwLevel</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>nOpt</i>	Analog trigger polarity and sensitivity. Valid values: 0 for Positive edge 2 for Negative edge
	<i>nChan</i>	Trigger channel. Valid values: 0 for ± 1 V signal from Ch A connector 1 for ± 1 V signal from Trg/Ch B connector 2 for ± 10 V signal from Trg/Ch B connector
	<i>dwLevel</i>	Level at which the trigger event occurs, specified in counts. Valid values: -128 to 127
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the operation defined by <i>hFrame</i> , this function specifies the channel used for an analog trigger in <i>nChan</i> , the level used for the analog trigger in <i>dwLevel</i> , and the trigger polarity and trigger sensitivity in <i>nOpt</i> . If you specified a value of 0 for <i>nChan</i> , you must specify channel 0 in K_SetChn . If you specified a value of 1 for <i>nChan</i> , you must specify channel 1 in K_SetChn . If you specified a value of 2 for <i>nChan</i> , you can specify either channel 0 or channel 1 in K_SetChn .	

K_SetADTrig (cont.)

You specify the value for *dwLevel* in counts. Refer to Appendix B for information on converting the actual voltage to a count value.

The values you specify set the following elements in the frame identified by *hFrame*:

- *nOpt* sets the value of the Trigger Polarity and Trigger Sensitivity elements.
- *nChan* sets the value of the Trigger Channel element.
- *dwLevel* sets the value of the Trigger Level element.

K_SetADTrig does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart**.

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetADTrig (hAD, 0, 1, 127);
```

Visual Basic for Windows

(Add *DASDECL.BAS* to your project)

```
...
wDasErr = K_SetADTrig (hAD, 0, 1, 127)
```

K_SetBuf

Purpose Specifies the starting address of a previously allocated array or buffer and the number of samples in the array or buffer.

Prototype **C/C++**
DASErr far pascal K_SetBuf (DWORD *hFrame*, void far **pBuf*,
DWORD *dwSamples*);

Visual Basic for Windows

Declare Function K_SetBuf Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *pBuf* As Long,
ByVal *dwSamples* As Long) As Integer

Parameters

<i>hFrame</i>	Handle to the frame that defines the operation.
<i>pBuf</i>	Starting address of array or buffer.
<i>dwSamples</i>	Number of samples. Valid values: 1 to 8192

Return Value Error/status code. Refer to Appendix A.

Remarks For the operation defined by *hFrame*, this function specifies the starting address of a previously allocated array or buffer in *pBuf* and the number of samples (the size of the array or buffer) in *dwSamples*.

For Visual Basic for Windows, use this function only for dynamically allocated buffers. For locally dimensioned arrays, use **K_SetBufI**.

For C application programs, make sure that you use proper typecasting to prevent C/C++ type-mismatch warnings.

The values you specify set the following elements in the frame identified by *hFrame*:

- *pBuf* sets the value of the Buffer element.
- *dwSamples* sets the value of the Number of Samples element.

See Also K_IntAlloc, K_SetBufI

K_SetBuf (cont.)

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
void far *pBuf;    // Pointer to allocated buffer
...
wDasErr = K_IntAlloc (hAD, 8192, &pBuf, &hMem);
wDasErr = K_SetBuf (hAD, pBuf, 8192);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
Global pBuf As Long
...
wDasErr = K_IntAlloc (hAD, 8191, pBuf, hMem)
wDasErr = K_SetBuf (hAD, pBuf, 8192)
```

K_SetBufI

Purpose	Specifies the starting address of a locally dimensioned integer array and the number of samples in the array.	
Prototype	C/C++ Not supported	
	Visual Basic for Windows Declare Function K_SetBufI Lib "DASSHELL.DLL" Alias "K_SetBuf" (ByVal <i>hFrame</i> As Long, <i>pBuf</i> As Integer, ByVal <i>dwSize</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>pBuf</i>	Starting address of the locally dimensioned integer array.
	<i>dwSize</i>	Number of samples. Valid values: 1 to 8192
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the operation defined by <i>hFrame</i> , this function specifies the starting address of a locally dimensioned integer array in <i>pBuf</i> and the number of samples stored in the array in <i>dwSize</i> . Do not use this function for C; instead, use K_SetBuf . For Visual Basic for Windows, use this function only for locally dimensioned arrays. For buffers allocated dynamically using K_IntAlloc , use K_SetBuf . The <i>pBuf</i> variable sets the value of the Buffer element; the <i>dwSize</i> variable sets the value of the Number of Samples element.	
See Also	K_IntAlloc, K_SetBuf	

K_SetBufI (cont.)

Usage

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
Global ADData(8191) As Integer  
...  
wDasErr = K_SetBufI (hAD, ADData(0), 8192)
```

K_SetChn

Purpose	Specifies a single channel.
Prototype	C/C++ DASErr far pascal K_SetChn (DWORD <i>hFrame</i> , short <i>nChan</i>); Visual Basic for Windows Declare Function K_SetChn Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>nChan</i> As Integer) As Integer
Parameters	<i>hFrame</i> Handle to the frame that defines the operation. <i>nChan</i> Channel on which to perform the operation. Valid values: 0 for signal from Ch A connector 1 for signal from Trg/Ch B connector
Return Value	Error/status code. Refer to Appendix A.
Remarks	For the operation defined by <i>hFrame</i> , this function specifies the single channel used in <i>nChan</i> . Software channel 0 corresponds to the analog input signal from the Ch A connector on the board; software channel 1 corresponds to the analog input signal from the Trg/Ch B connector on the board. The value you specify in <i>nChan</i> sets the Channel element in the frame identified by <i>hFrame</i> .
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_SetChn (hAD, 0);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) <pre>... wDasErr = K_SetChn (hAD, 0)</pre>

Purpose	Specifies the pacer clock source.
Prototype	C/C++ DASErr far pascal K_SetClk (DWORD <i>hFrame</i> , short <i>nMode</i>); Visual Basic for Windows Declare Function K_SetClk Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>nMode</i> As Integer) As Integer
Parameters	<i>hFrame</i> Handle to the frame that defines the operation. <i>nMode</i> Pacer clock source. Valid values: 0 for Internal 1 for External
Return Value	Error/status code. Refer to Appendix A.
Remarks	For the operation defined by <i>hFrame</i> , this function specifies the pacer clock source in <i>nMode</i> . The value you specify in <i>nMode</i> sets the Clock Source element in the frame identified by <i>hFrame</i> . K_GetADFrame and K_ClearFrame specify internal as the default clock source.
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_SetClk (hAD, 1);</pre> Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project) ... wDasErr = K_SetClk (hAD, 1)

K_SetClkRate

Purpose Specifies a count value that determines the time between conversions for the internal pacer clock.

Prototype **C/C++**
DASErr far pascal K_SetClkRate (DWORD *hFrame*,
DWORD *dwDivisor*);

Visual Basic for Windows

Declare Function K_SetClkRate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwDivisor* As Long) As Integer

Parameters

<i>hFrame</i>	Handle to the frame that defines the operation.
<i>dwDivisor</i>	Count value. Valid values: 20 to 25 600 , described as follows::

Conversion Rate	Sample Period	Count Value
1 Gsamples/s	1 ns	20
500 Msamples/s	2 ns	40
250 Msamples/s	4 ns	80
100 Msamples/s	10 ns	200
50 Msamples/s	20 ns	400
25 Msamples/s	40 ns	800
12.5 Msamples/s	80 ns	1600
6.25 Msamples/s	160 ns	3200
3.125 Msamples/s	320 ns	6400
1.5625 Msamples/s	640 ns	12800
0.78250 Msamples/s	1280 ns	25600

K_SetClkRate (cont.)

Return Value	Error/status code. Refer to Appendix A.
Remarks	<p>For the operation defined by <i>hFrame</i>, this function specifies the count value, which is divided into 20 Gsamples/s, for the internal pacer clock in <i>dwDivisor</i>.</p> <p>The value you specify in <i>dwDivisor</i> sets the Pacer Clock Rate element in the frame identified by <i>hFrame</i>.</p> <p>Refer to page 2-8 for more information on the internal pacer clock.</p>
See Also	K_GetClkRate
Usage	<p>C/C++</p> <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... wDasErr = K_SetClkRate (hAD, 20);</pre> <p>Visual Basic for Windows (Add <i>DASDECL.BAS</i> to your project)</p> <pre>... wDasErr = K_SetClkRate (hAD, 20)</pre>

K_SetDITrig

Purpose	Sets up an external digital start or about trigger.	
Prototype	C/C++ DASErr far pascal K_SetDITrig (DWORD <i>hFrame</i> , short <i>nOpt</i> , short <i>nChan</i> , DWORD <i>nPattern</i>); Visual Basic for Windows Declare Function K_SetDITrig Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>nOpt</i> As Integer, ByVal <i>nChan</i> As Integer, ByVal <i>nPattern</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>nOpt</i>	Trigger polarity and sensitivity. Valid values: 0 for Positive edge 2 for Negative edge
	<i>nChan</i>	Digital input channel. Valid value: 0
	<i>nPattern</i>	Trigger pattern.
Return Value	Error/status code. Refer to Appendix A.	
Remarks	This function specifies the use of a digital trigger for the operation defined by <i>hFrame</i> . Since an external digital trigger is always connected to the Trg IO connector on the board, the value of <i>nChan</i> is meaningless. In addition, the DAS-4300 Series Function Call Driver does not support digital pattern triggering; therefore, the value of <i>nPattern</i> is meaningless. The <i>nChan</i> and <i>nPattern</i> parameters are provided for future compatibility.	

K_SetDITrig (cont.)

The values you specify set the following elements in the frame identified by *hFrame*:

- *nOpt* sets the value of the Trigger Polarity element.
- *nChan* sets the value of the Trigger Channel element.
- *nPattern* sets the value of the Trigger Pattern element.

K_SetDITrig does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart**.

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++  
...  
wDasErr = K_SetDITrig (hAD, 0, 0, 0);
```

Visual Basic for Windows

(Add *DASDECL.BAS* to your project)

```
...  
wDasErr = K_SetDITrig (hAD, 0, 0, 0)
```

K_SetG

Purpose Sets the input range.

Prototype **C/C++**
DASErr far pascal K_SetG (DWORD *hFrame*, short *nGain*);

Visual Basic for Windows

Declare Function K_SetG Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nGain* As Integer) As Integer

Parameters *hFrame* Handle to the frame that defines the operation.
nGain Gain code.
Valid values: **0** to **15**, described as follows:

Analog Input Range	Gain Code	Analog Input Range	Gain Code
±25 mV	12	±200 mV	0
±31.25 mV	13	±250 mV	1, 10
±62.5 mV	14	±312.5 mV	6
±100 mV	8	±0.5 V	2, 11
±125 mV	4, 9, 15	±0.625 V	7
±156.25 mV	5	±1.0 V	3

Return Value Error/status code. Refer to Appendix A.

Remarks For the operation defined by *hFrame*, this function specifies the gain code, which represents the input voltage range, for a single channel in *nGain*. Refer to Appendix C to understand the effect of input voltage ranges on the bandwidth of the DAS-4301/8K board.

The value you specify in *nGain* sets the Gain element in the frame identified by *hFrame*.

K_GetADFrame and **K_ClearFrame** specify 0 as the default gain code.

K_SetG (cont.)

Usage

C/C++

```
#include "DASDECL.H" // Use "DASDECL.HPP for C++  
...  
wDasErr = K_SetG (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
wDasErr = K_SetG (hAD, 1)
```

K_SetPostTrigDelay

Purpose	Sets the number of post-trigger delay samples.	
Prototype	C/C++ DASErr far pascal K_SetPostTrigDelay (WORD <i>hFrame</i> , DWORD <i>nDelay</i>); Visual Basic for Windows Declare Function K_SetPostTrigDelay Lib "DAS4300.DLL" (ByVal <i>hFrame</i> As Long, <i>nDelay</i> As Long) As Integer	
Parameters	<i>hFrame</i>	Handle to the frame that defines the operation.
	<i>nDelay</i>	Post-trigger delay samples. Valid values: 0 to 65536 (at conversion rate of 100 Msamples/s or slower) 0 to 655360 (at conversion rate of 250 Msamples/s or faster)
Return Value	Error/status code. Refer to Appendix A.	
Remarks	For the operation defined by <i>hFrame</i> , this function specifies the number of post-trigger delay samples in <i>nDelay</i> . At a conversion rate of 100 Msamples/s or slower, the post-trigger delay can range from 0 to 65,536 samples (in increments of 1); at a conversion rate of 250 Msamples/s or faster, the post-trigger delay can range from 0 to 655,360 samples (in increments of 10). You cannot specify a post-trigger delay if you are using an about trigger.	
Usage	C/C++ <pre>#include "DASDECL.H" // Use "DASDECL.HPP for C++ ... WORD nDelay; ... wDasErr = K_SetPostTrigDelay (hFrame, hDelay);</pre>	

K_SetPostTrigDelay (cont.)

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
Global Delay As Long  
...  
wDasErr = K_SetPostTrigDelay (hFrame, nDelay)
```

K_SetTrig

Purpose	Specifies the trigger source.
Prototype	C/C++ DASErr far pascal K_SetTrig (DWORD <i>hFrame</i> , short <i>nMode</i>); Visual Basic for Windows Declare Function K_SetTrig Lib "DASSHELL.DLL" (ByVal <i>hFrame</i> As Long, ByVal <i>nMode</i> As Integer) As Integer
Parameters	<i>hFrame</i> Handle to the frame that defines the operation. <i>nMode</i> Trigger source. Valid values: 0 for Internal start trigger 1 for External start or about trigger
Return Value	Error/status code. Refer to Appendix A.
Remarks	<p>For the operation defined by <i>hFrame</i>, this function specifies the trigger source in <i>nMode</i>.</p> <p>An internal trigger is a software trigger; conversions begin when your application program calls K_IntStart. An external trigger is either an analog trigger or a digital trigger. Refer to page 2-10 for more information on trigger sources.</p> <p>You cannot use an external start trigger (as required for post-trigger acquisition) when the about trigger is enabled. If you want to use an external start trigger, ensure that the about trigger is disabled by using the K_ClrAboutTrig function. Refer to page 2-13 for more information on trigger acquisitions.</p> <p>If <i>nMode</i> = 1, an external digital trigger is assumed. Use K_SetDITrig to change the conditions of the digital trigger. Use K_SetADTrig to specify the conditions for an external analog trigger.</p> <p>K_GetADFrame and K_ClearFrame set the trigger source to internal.</p>

K_SetTrig (cont.)

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++  
...  
wDasErr = K_SetTrig (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...  
wDasErr = K_SetTrig (hAD, 1)
```

A

Error/Status Codes

Table A-1 lists the error/status codes that are returned by the DAS-4300 Series Function Call Driver, possible causes for error conditions, and possible solutions for resolving error conditions.

If you cannot resolve an error condition, contact the Keithley MetraByte Applications Engineering Department.

Table A-1. Error/Status Codes

Error Code		Cause	Solution
Hex	Decimal		
0	0	No error has been detected.	Status only; no action is necessary.
6000	24576	Error in configuration file: The configuration file you specified in the driver initialization function is corrupt, does not exist, or contains one or more undefined keywords.	Check that the file exists at the specified path. Check for illegal keywords in file; you can avoid illegal keywords by using the configuration utility to create and modify configuration files.
6001	24577	Illegal base address in configuration file: The board's base I/O address in the configuration file is illegal and/or does not match the base address switches on the board.	Use the configuration utility to change the base I/O address to one that matches the base address switches on the board.
6002	24578	Illegal IRQ level in configuration file: The interrupt level in the configuration file is illegal.	Use the configuration utility to change the interrupt level to a legal one for your board. Refer to the user's guide for legal interrupt levels.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6003	24579	Illegal DMA channel in configuration file: The DMA channel in the configuration file is illegal.	Use the configuration utility to change the DMA channel to a legal one for your board. Refer to the user's guide for legal DMA channels.
6005	24581	Illegal channel number: The specified channel number is illegal for the board and/or for the range type (unipolar or bipolar).	Specify a legal channel number. Refer to the user's guide or to the description of K_SetChn in Chapter 4 for legal channel numbers.
6006	24582	Illegal gain code: The specified channel gain code is illegal for this board.	Specify a legal gain code. Refer to the user's guide or to the description of K_SetG in Chapter 4 for a list of legal gain codes.
6007	24583	Illegal DMA address: An FCD function specified a buffer address that is not suitable for a DMA operation for the number of samples required.	Use the K_DMAAlloc function to allocate dynamic buffers for DMA operations. In Windows, make sure that the Keithley Memory Manager is installed; refer to the user's guide for information.
6008	24584	Illegal number in configuration file: The configuration file contains one or more numeric values that are illegal.	Use the configuration utility to check and then change the configuration file.
600A	24586	Configuration file not found: The driver cannot find the configuration file specified as an argument to the driver initialization function.	Check that the file exists at the specified path. Check that the file name is spelled correctly in the driver initialization function parameter list.
600B	24587	Error returning DMA buffer: DOS returned an error in INT 21H function 49H during the execution of K_DMAFree .	Check that the memory handle passed as an argument to K_DMAFree was previously obtained using K_DMAAlloc .

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
600C	24588	Error returning interrupt buffer: The memory handle specified in K_IntFree is invalid.	Check the memory handle stored by K_IntAlloc and make sure that it was not modified.
600D	24589	Illegal frame handle: The specified frame handle is not valid for this operation.	Check that the frame handle exists. Check that you are using the appropriate frame handle.
600E	24590	No more frame handles: No frames are left in the pool of available frames.	Use K_FreeFrame to free a frame that the application is no longer using.
600F	24591	Requested buffer size too large: The requested buffer cannot be dynamically allocated because of its size.	Specify a smaller buffer size; refer to the description of K_IntAlloc in Chapter 4 for the legal range. If in Windows Enhanced mode with the Keithley Memory Manager (VDMAD.386) installed, use KMMSETUP.EXE to increase the reserved buffer heap size.
6010	24592	Cannot allocate interrupt buffer: (Windows-based languages only) K_IntAlloc failed because there was not enough available DOS memory.	Remove some Terminate and Stay Resident programs (TSRs) that are no longer needed.
6012	24594	Interrupt buffer deallocation error: (Windows-based languages only) An error occurred when K_IntFree attempted to free a memory handle.	Make sure that the memory handle passed as an argument to K_IntFree was previously obtained using K_IntAlloc .
6015	24597	DMA Buffer too large: The number of samples specified in K_DMAAlloc is too large.	Refer to the description of K_DMAAlloc in Chapter 4 for the buffer size range.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6016	24598	VDS - Region not contiguous: An error occurred while using Windows Virtual DMA Services. You tried to use K_DMAAlloc in Windows Enhanced mode and the Keithley Memory Manager (VDMAD.386) was not installed	Refer to the user's guide for information on how to install and set up the Keithley Memory Manager (VDMAD.386).
6017	24599	VDS - DMA wraparound: See error 6016.	See error 6016.
6018	24600	VDS - Unable to lock region: See error 6016.	See error 6016.
6019	24601	VDS - No buffer available: See error 6016.	See error 6016.
601A	24602	VDS - Region too large: See error 6016.	See error 6016.
601B	24603	VDS - Buffer in use: See error 6016.	See error 6016.
601C	24604	VDS - Illegal region: See error 6016.	See error 6016.
601D	24605	VDS - Region not locked: See error 6016.	See error 6016.
601E	24606	VDS - Illegal page: See error 6016.	See error 6016.
601F	24607	VDS - Illegal buffer: See error 6016.	See error 6016.
6020	24608	VDS - Copy out of range: See error 6016.	See error 6016.
6021	24609	VDS - Illegal DMA channel: See error 6016.	See error 6016.
6022	24610	VDS - Count overflow: See error 6016.	See error 6016.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6023	24611	VDS - Count underflow: See error 6016.	See error 6016.
6024	24612	VDS - Function not supported: See error 6016.	See error 6016.
6025	24613	Illegal OBM mode: The mode number specified in K_SetOBMMode is illegal.	Refer to the description of K_SetOBMMode in Chapter 4 for legal mode values.
6026	24614	Illegal DMA structure: An error occurred during the execution of K_DMAFree .	Try using K_DMAFree again. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
6027	24615	DMA allocation error: See error 6026.	See error 6026.
6028	24616	NULL DMA handle: See error 6026.	See error 6026.
6029	24617	DMA unlock error: See error 6026.	See error 6026.
602A	24618	DMA free error: See error 6026.	See error 6026.
602B	24619	Not enough memory to accommodate request: The number of samples you requested in the Keithley Memory Manager is greater than the largest contiguous block available in the reserved heap.	Specify a smaller number of samples. Free a previously allocated buffer. Use the KMMSETUP utility to expand the reserved heap.
602C	24620	Requested buffer size exceeds maximum: The number of samples you requested from the Keithley Memory Manager is greater than the allowed maximum.	Specify a value within the legal range when calling K_DMAAlloc or K_IntAlloc in Windows Enhanced mode. Refer to the description of K_DMAAlloc or K_IntAlloc in Chapter 4 for legal values.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
602D	24621	Illegal device handle: A bad device handle was passed to a function such as K_GetADFrame . The handle used was not initialized using K_GetDevHandle or it was corrupted by your program.	Check the device handle value.
602E	24622	Illegal Setup option: An illegal option was specified to a function that accepts a user option, such as K_SetDITrig .	Check the option value passed to the function where the error occurred.
6030	24624	DMA word-page wrap: During K_DMAAlloc , a DMA word-page wrap condition occurred and the allocation attempt failed since there is not enough free memory to accommodate the allocation request.	Reduce the number of samples and retry. If in Windows Enhanced mode, install and configure VDMAD.386.
6031	24625	Illegal memory handle: A bad memory handle was passed to K_IntFree or K_DMAFree . The handle used was not initialized through a call to K_IntAlloc or K_DMAAlloc , or it was corrupted by you program.	Restart your program and monitor the memory handle value(s).
6032	24626	Out of memory handles: An attempt to allocate a memory block using K_IntAlloc or K_DMAAlloc failed because the maximum number of handles has already been assigned.	Use K_IntFree or K_DMAFree to free previously allocated memory blocks before allocating again.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6034	24628	Memory corrupted: Int 21H function 48H, used to allocate a memory block from the DOS far heap, returned the DOS error 7; this means that memory is corrupted. It is likely that you stored data (through a DMA-mode or interrupt-mode operation) into an illegal area of DOS memory.	Recheck the parameters set by K_DMAAlloc and K_SetDMABuf . If a fatal system error, restart your computer.
6035	24629	Driver in use: The driver attempted to configure a device that had already been configured by a call to K_OpenDriver . (This can occur since, under Windows, it is possible to open the same driver from multiple programs that are running simultaneously.)	To continue using the driver with the same configuration, pass a null string as the second argument to K_OpenDriver . To use the driver with a different configuration, close any application programs currently accessing the driver, and then open the driver again (using K_OpenDriver).
6036	24630	Illegal driver handle: The specified driver handle is not valid.	Someone may have closed the driver; if so, use K_OpenDriver to reopen the driver with the desired driver handle. Try again using another driver handle.
6037	24631	Driver not found: The specified driver cannot be found.	Check your link statement to make sure the specified driver is included. Make sure that the device name string is entered correctly in K_OpenDriver .

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6038	24632	Invalid source pointer: (Windows-based languages only) The pointer to the source buffer that you passed as an argument to K_MoveBufToArray is invalid for the specified count. (The source pointer, when added to the number of samples, exceeds the programmed addressing range of that pointer.)	Check the pointer to the source buffer and the number of samples to transfer that you specified in K_MoveBufToArray .
6039	24633	Invalid destination pointer: (Windows-based languages only) The pointer to the destination buffer (local array) that you passed as an argument to K_MoveBufToArray is invalid for the specified count. (The destination pointer, when added to the number of samples, exceeds the dimension of the local array.)	Check the dimension of the local array and the number of samples to transfer that you specified in K_MoveBufToArray .
603A	24634	Illegal setup value: An illegal value was passed to the function in which the error occurred.	Check the legal ranges of all parameters passed to this function.
603B	24635	Error freeing buffer selector: K_DMAFree or K_IntFree failed because one or more of the selectors that reference the memory buffer could not be freed.	Check that the memory buffer being freed was previously obtained through K_DMAAlloc or K_IntAlloc .
603C	24636	Error allocating buffer selector: K_DMAAlloc or K_IntAlloc failed because a selector could not be allocated from Window's Local Descriptor Table.	Close all applications and restart Windows. If the error continues, contact the Keithley MetraByte Applications Engineering Department.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
603D	24637	Error allocating memory buffer: K_DMAAlloc or K_IntAlloc failed because a necessary internal buffer could not be allocated to complete the operation.	Close all applications and restart Windows. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
7000	28672	No board number: The board number field was missing or out of place in the specified configuration file.	Specify the board number in the configuration file.
7002	28674	Illegal board number: The driver initialization function found an illegal board number in the specified configuration file.	Specify a legal board number: 0 to 1
7003	28675	Illegal base address: The driver initialization function found an illegal base I/O address in the specified configuration file.	Specify a base I/O address in the inclusive range &H240 (576) to &H2F8 (760) in increments of 8H (8). Make sure that &H precedes hexadecimal numbers.
7004	28676	Illegal memory address: The driver initialization function found an illegal memory address in the specified configuration file.	Specify a memory address in the inclusive range &HA000 to &HDC00 in increments of 400H. Make sure that &H precedes hexadecimal numbers.
7005	28677	Illegal interrupt level: The driver initialization function found an illegal interrupt level in the specified configuration file.	Specify a legal interrupt level: 5, 7, 9, 10, 11, 12, or 15
7007	28679	Illegal zero wait state: The driver initialization function found an illegal input in the specified configuration file.	Specify enabled or disabled.
7009	28681	Illegal A/D channel: The analog input channel specified in K_SetChn is illegal.	Make sure that the analog input channel in K_SetChn is 0 (Ch A connector), or (Trg/Ch B connector).

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
700A	28682	Illegal start- and about-trigger combination: The start trigger must be internal when the about trigger is enabled.	Either set the start trigger to internal using K_SetTrig , or disable the about trigger using K_ClrAboutTrig .
700B	28683	Illegal analog trigger channel: The analog input channel specified in K_SetADTrig does not match the analog input channel being sampled (specified in K_SetChn).	When using an external software analog trigger, make sure that the analog trigger channel is the same as the analog input channel that is sampled.
700C	28684	Illegal about- and post-trigger delay combination: The about trigger cannot be enabled while the post-trigger delay is specified.	Either disable the about trigger using K_ClrAboutTrig or do not specify a delay with K_SetPostTrigDelay .
700E	28686	Resource busy: The application program attempted to start an operation while a similar operation was in progress.	Use K_IntStop to stop the in-progress operation before initiating the second operation.
700F	28687	Start and stop channel are not equal: Only one analog input channel can be sampled during an operation.	Do not try to acquire data from more than one channel at a time.
7010	28688	Illegal number of post-trigger delay samples: At a conversion rate of 100 Msamples/s or slower, the number of post-trigger delay samples can range from 0 to 65,536. At a conversion rate of 250 Msamples/s or faster, the number of post-trigger delay samples can range from 0 to 655,360.	Use K_SetPostTrigDelay to specify a valid number of post-trigger delay samples.
7012	28690	Missing clock pulse or rate too slow: The external clock is using a conversion rate less than 100 Msamples/s.	Ensure that the external clock uses a conversion rate above 100 Msamples/s.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
7015	28693	Illegal number of about-trigger samples: The number of about-trigger samples exceeds the buffer size or is greater than 8192 samples.	Ensure that the number of about-trigger samples specified in K_SetAboutTrig is less or equal to 8191 or is less than or equal to the size of your array or buffer as specified in K_SetBuf or K_SetBufL .
7017	28695	Illegal calibration on board: The EEPROM settings are not giving a consistent value.	Contact the Keithley MetraByte Applications Engineering Department.
7018	28696	Memory map conflict or inconsistency: The board could not map into the memory address configured.	Configure the jumpers on the board to choose another memory address.
7019	28697	Windows could not supply memory map: Windows did not return the selector for the memory location of the board requested.	Check to make sure that your memory manager excludes the memory your board is using. For example, if you are using EMM386, your CONFIG.SYS file should contain a line similar to the following: <code>DEVICE=C:\DOS\EMM386.EXE</code> <code>X=CC00-CFFF</code> (Note this should be typed on one line.)
8001	32769	Function not supported: You have attempted to use a function not supported by the Function Call Driver.	Contact the Keithley MetraByte Applications Engineering Department.
8003	32771	Illegal board number: An illegal board number was specified in the board initialization function.	Refer to the description of K_GetDevHandle in Chapter 4 for legal board numbers.
8004	32772	Illegal error number: The error message number specified in K_GetErrMsg is invalid.	The error number must be one the error numbers listed in this appendix.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
8005	32773	Board not found at configured address: The board initialization function does not detect the presence of a board.	Make sure that the base address setting of the switches on the board matches the base address setting in the configuration file.
8006	32774	A/D not initialized: You attempted to start a frame-based analog input operation without the A/D frame being properly initialized.	Always call K_ClearFrame before setting up a new frame-based operation.
8007	32775	D/A not initialized: You attempted to start a frame-based analog output operation without the D/A frame being properly initialized.	Always call K_ClearFrame before setting up a new frame-based operation.
8008	32776	Digital input not initialized: You attempted to start a frame-based digital input operation without the DI frame being properly initialized.	Always call K_ClearFrame before setting up a new frame-based operation.
8009	32777	Digital output not initialized: You attempted to start a frame-based digital output operation without the DO frame being properly initialized.	Always call K_ClearFrame before setting up a new frame-based operation.
800B	32779	Conversion overrun: Data was overwritten before it was transferred to the computer's memory.	Adjust the clock source to slow down the rate at which the board acquires data. Remove other application programs that are running and using computer resources.
8016	32790	Interrupt overrun: The board communicated a hardware event to the software by generating a hardware interrupt, but the software was still servicing a previous interrupt. This is usually caused by a pacer clock rate that is too fast.	Check the maximum throughput rate for your computer's programming environment and use K_SetClkRate to specify an appropriate rate.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
801A	32794	Interrupts already active: You have attempted to start an operation whose interrupt level is being used by another system resource.	Use K_IntStop to stop the first operation before starting the second operation.
801B	32795	DMA already active: You attempted to start a DMA-mode operation using a DMA channel that is currently used by another active operation.	Use K_DMAStop to stop the first operation before starting the second operation.
801C	32796	Timer channel already active: This error appears when you try to perform an operation and the timer channel is already in use by another system resource.	Stop the first operation before starting the next operation, or wait until the first operation stops before starting the next operation.
8020	32800	FIFO Overflow event detected: During data acquisition, the temporary on-board data storage (FIFO) overflowed.	The conversion rate is too fast for your computer's programming environment; use K_SetClkRate to reduce the conversion rate. If you are using DMA-mode and your board supports dual-DMA, use the configuration utility to reconfigure your board to use dual-DMA.
8021	32801	Illegal clock sync mode: The two operations you are trying to synchronize cannot be synchronized on your board.	Check the synchronizing clock source that you specified in K_SetSync . Make sure that your board supports clock synchronization.
FFFF	65535	User aborted operation: You pressed Ctrl+Break during a synchronous-mode operation or while waiting for an analog trigger event to occur.	Start the operation again, if desired.

B

Data Formats

The DAS-4300 Series Function Call Driver can read and write counts only. When reading a value, you may want to convert the count to a more meaningful voltage value; when writing a value (as in **K_SetTrig**), you must convert the voltage value to a count value.

The remainder of this appendix contains instructions for converting counts to voltage and for converting voltage to counts.

Converting Counts to Voltage

You may want to convert counts to voltage when reading an analog input value.

To convert an analog input value to a voltage, use the following equation, where *count* is the count value, and *span* is the appropriate value from Table B-1 on page B-2:

$$\text{Voltage} = \frac{\text{count} \times \text{span}}{256}$$

Table B-1. Some Span Values For Analog Input Data Conversion Equations

Gain Code	Input Range	Span (V)
0	±200 mV	0.4
1	±250 mV	.5
2	±0.5 V	1
3	±1 V	2
4	±125 mV	0.25
5	±156.25 mV	0.3125
6	±312.5 mV	0.625
7	±0.625 V	1.25
8	±100 mV	0.2
9	±125 mV	0.25
10	±250 mV	0.5
11	±0.5 V	1
12	±25 mV	0.5
13	±31.25 mV	0.625
14	±62.5 mV	0.125
15	±125 mV	0.25

For example, assume that you want to read analog input data from a channel on a DAS-4301/8K board with an input range of ±1 V (span of 2 V); the count value is 72. The voltage is determined as follows:

$$\frac{72 \times 2 \text{ V}}{256} = 0.5625 \text{ V}$$

Converting Voltage to Counts

You must convert voltage to a count value when specifying an analog trigger level.

To convert a voltage to a count value when specifying an analog trigger level, use one of the following equations, where V_{trig} is the desired voltage, and $span$ is the appropriate value from Table B-1 on page B-2:

For ± 1 V Trigger Signal from Ch A or Trg/Ch B Connector

$$\text{Count} = \frac{V_{trig} \times 256}{span}$$

For ± 10 V Trigger Signal from Trg/Ch B Connector

$$\text{Count} = \frac{V_{trig} \times 256}{20}$$

For example, assume that you want to specify an analog trigger level of 125 mV for the input signal coming into the Ch A connector on a DAS-4301/8K board configured for an input range type of ± 200 mV (span of 0.4 V). The count value is determined as follows:

$$\frac{0.125 \text{ V} \times 256}{0.4 \text{ V}} = 80$$

Alternatively, assume that you want to specify an analog trigger level of 5 V for the trigger input signal coming into the Trg/Ch B connector on a DAS-4301/8K board. The count value is determined as follows:

$$\frac{5 \text{ V} \times 256}{20 \text{ V}} = 64$$

C

Bandwidth Charts for Input Voltage Ranges

The following figures show the effect of input voltage ranges on the bandwidth of the DAS-4301/8K board. These figures are useful in determining the best input voltage range for a particular application. Note that the number in parentheses indicates the gain code used.

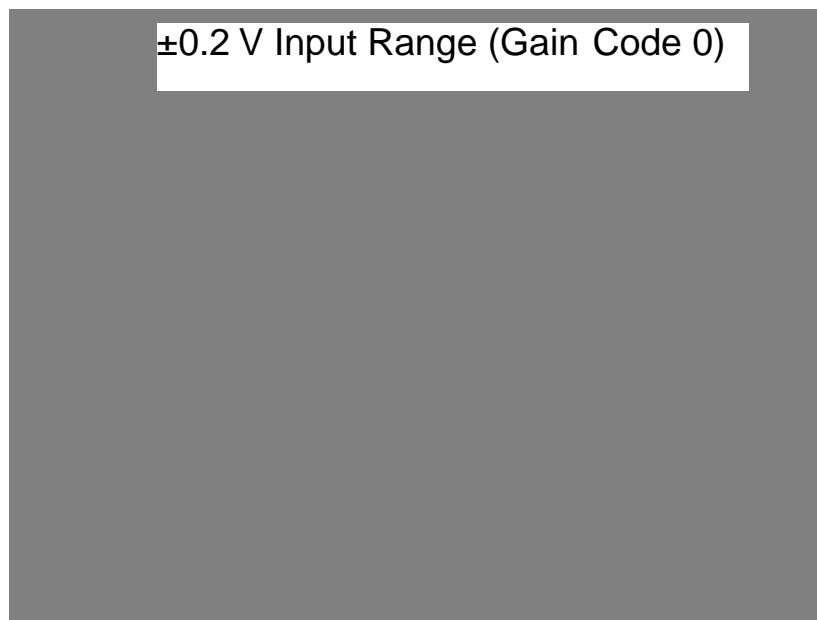


Figure C-1. ±0.2 V Input Range (Gain Code 0)

± 0.25 V Input Range (Gain Code 1)

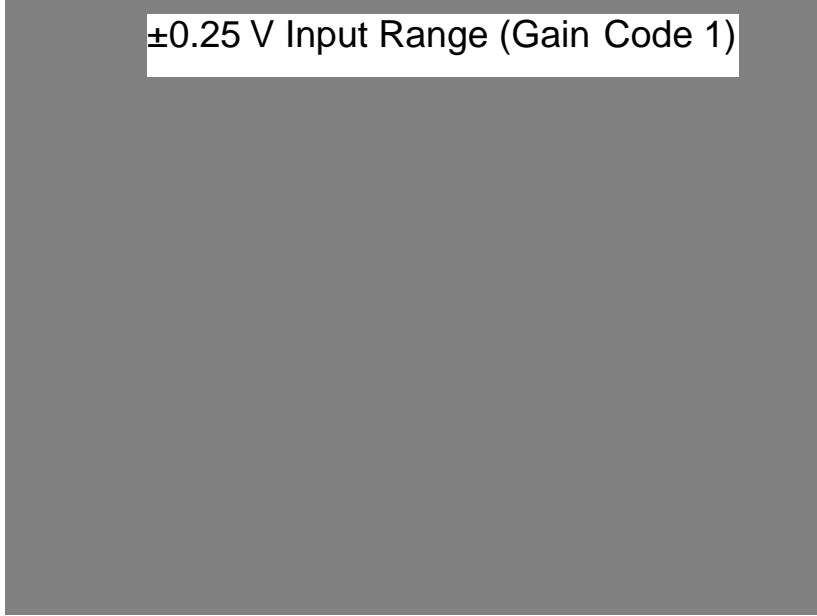


Figure C-2. ± 0.25 V Input Range (Gain Code 1)

± 0.5 V Input Range (Gain Code 2)

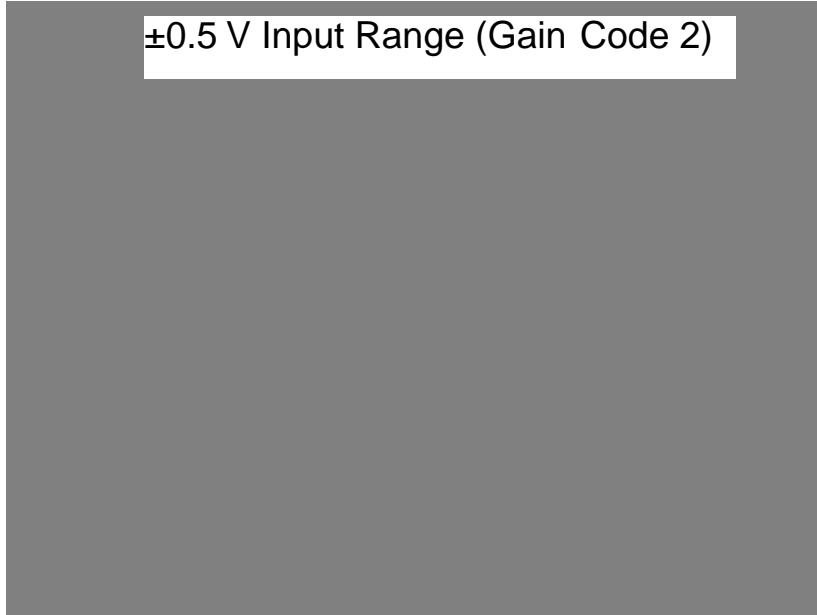


Figure C-3. ± 0.5 V Input Range (Gain Code 2)

± 1 V Input Range (Gain Code 3)



Figure C-4. ± 1 V Input Range (Gain Code 3)

± 0.125 V Input Range (Gain Code 4)



Figure C-5. ± 0.125 V Input Range (Gain Code 4)

± 0.15625 V Input Range (Gain Code 5)

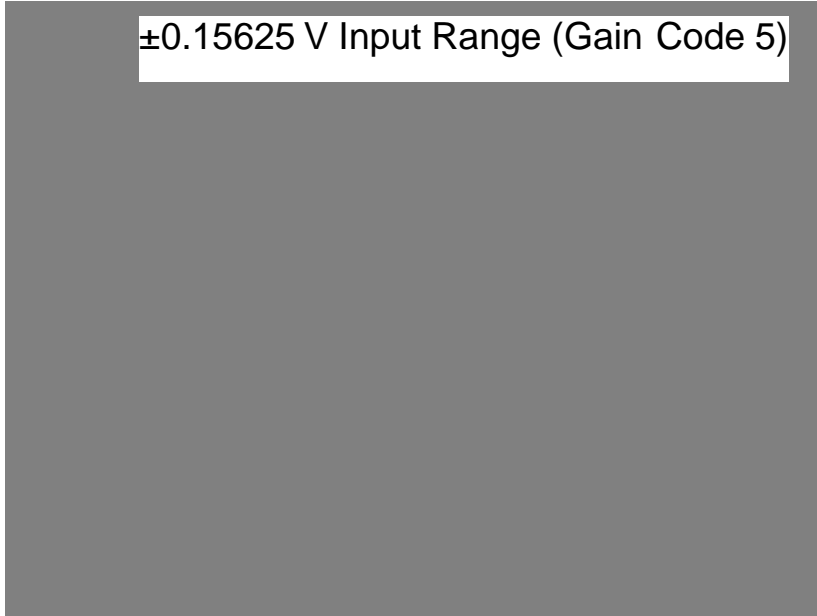


Figure C-6. ± 0.15625 V Input Range (Gain Code 5)

± 0.3125 V Input Range (Gain Code 6)

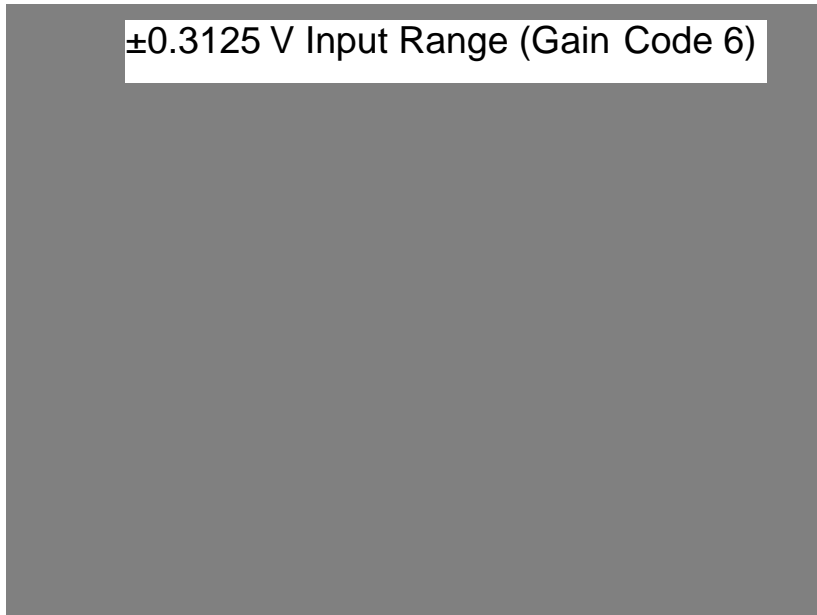


Figure C-7. ± 0.3125 V Input Range (Gain Code 6)

± 0.625 V Input Range (Gain Code 7)



Figure C-8. ± 0.625 V Input Range (Gain Code 7)

± 0.1 V Input Range (Gain Code 8)



Figure C-9. ± 0.1 V Input Range (Gain Code 8)

± 0.125 V Input Range (Gain Code 9)

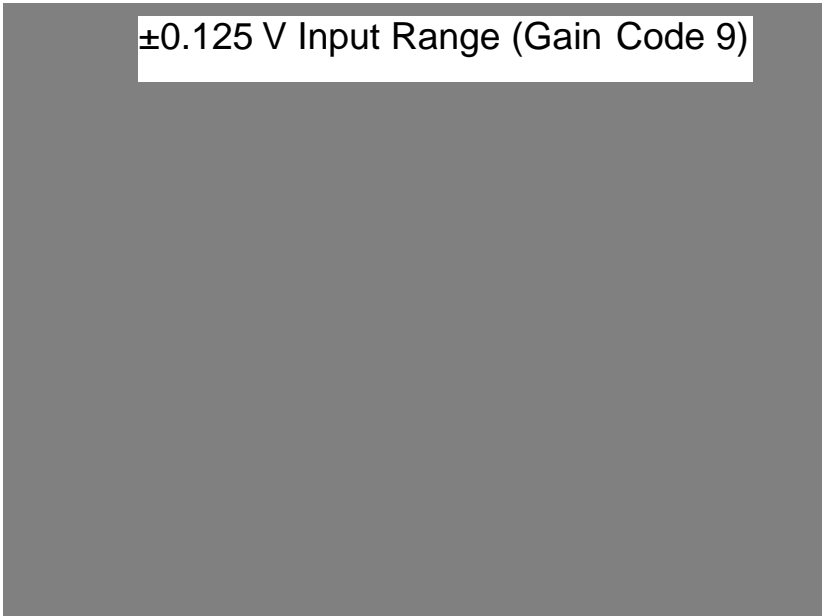


Figure C-10. ± 0.125 V Input Range (Gain Code 9)

± 0.25 V Input Range (Gain Code 10)

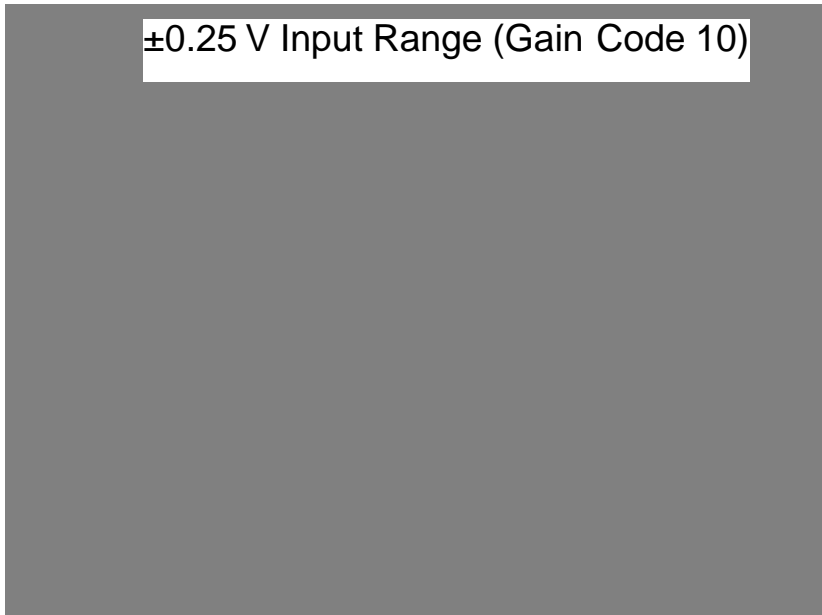


Figure C-11. ± 0.25 V Input Range (Gain Code 10)

± 0.5 V Input Range (Gain Code 11)



Figure C-12. ± 0.5 V Input Range (Gain Code 11)

± 0.025 V Input Range (Gain Code 12)



Figure C-13. ± 0.025 V Input Range (Gain Code 12)

± 0.03125 V Input Range (Gain Code 13)



Figure C-14. ± 0.03125 V Input Range (Gain Code 13)

± 0.0625 V Input Range (Gain Code 14)



Figure C-15. ± 0.0625 V Input Range (Gain Code 14)

± 0.125 V Input Range (Gain Code 15)



Figure C-16. ± 0.125 V Input Range (Gain Code 15)

Index

A

- about-trigger acquisition 2-15
- allocating memory
 - dynamically in C/C++ 3-9
 - dynamically in Visual Basic for Windows 3-16
 - locally in C/C++ 3-8
 - locally in Visual Basic for Windows 3-15
- analog input operations
 - channels 2-7
 - converting analog input values to voltages B-1
 - input ranges 2-7
 - memory allocation 2-5
 - operation modes 2-4
 - pacer clocks 2-8
 - programming tasks 3-6
 - triggers 2-10
- analog trigger B-3

B

- bandwidth charts C-1
- board initialization 2-2
- Borland C/C++
 - compile and link statements for DOS 3-13
 - compile and link statements for Windows 3-14
 - creating an executable file for DOS 3-13
 - creating an executable file for Windows 3-14
 - dimensioning a local array 3-8
 - dynamically allocating a memory buffer 3-9

- files required for DOS 3-13
- files required for Windows 3-14
- handling errors 3-10
- buffer address 2-7
- buffer address function 4-2

C

- C/C++: *see* Borland C/C++, Microsoft C/C++
- channel and gain functions 4-2
- channels 2-7
- clock functions 4-2
- clock sources: *see* pacer clocks
- compile and link statements
 - Borland C/C++ (for DOS) 3-13
 - Borland C/C++ (for Windows) 3-14
 - Microsoft C/C++ (for DOS) 3-11
 - Microsoft C/C++ (for Windows) 3-12
- conventions 4-3
- conversion rate 2-9
- converting
 - counts to voltages B-1
 - voltages to counts B-3
- creating an executable file
 - Borland C/C++ (for DOS) 3-13
 - Borland C/C++ (for Windows) 3-14
 - Microsoft C/C++ (for DOS) 3-11
 - Microsoft C/C++ (for Windows) 3-12
 - Visual Basic for Windows 3-18

D

- data conversions
 - converting counts to voltages B-1
 - converting voltages to counts B-3
- data types 4-4
- default values of A/D frame elements 3-4
- device handle 2-2, 3-1

- digital trigger 2-12
- dimensioning a local array
 - C/C++ 3-8
 - Visual Basic for Windows 3-15
- driver handle 2-2
- driver initialization 2-2
- dynamically allocating a memory buffer
 - C/C++ 3-9
 - Visual Basic for Windows 3-16

E

- elements of frame 3-4
- error codes A-1
- error handling 2-3
 - Borland C/C++ 3-10
 - Microsoft C/C++ 3-10
 - Visual Basic for Windows 3-17
- external pacer clock 2-9

F

- files required
 - Borland C/C++ (for DOS) 3-13
 - Borland C/C++ (for Windows) 3-14
 - Microsoft C/C++ (for DOS) 3-11
 - Microsoft C/C++ (for Windows) 3-12
 - Visual Basic for Windows 3-18
- frame elements 3-4
- frame handle 3-2
- frame management functions 4-2
- frame types 3-2
- functions
 - buffer address 4-2
 - channel and gain 4-2
 - clock 4-2
 - frame management 4-2
 - initialization 4-2
 - K_ClearFrame 3-3, 4-5

- K_CloseDriver 2-2, 4-6
- K_ClrAboutTrig 4-7
- K_DASDevInit 2-3, 4-8
- K_FreeDevHandle 2-2, 4-9
- K_FreeFrame 3-3, 4-10
- K_GetADFrame 3-2, 4-11
- K_GetClkRate 2-9, 4-13
- K_GetDevHandle 4-15
- K_GetErrMsg 2-4, 4-17
- K_GetShellVer 2-3, 4-18
- K_GetVer 2-3, 4-19
- K_IntAlloc 2-6, 4-21
- K_IntFree 2-6, 4-23
- K_IntStart 2-4, 4-24
- K_IntStatus 2-4, 4-25
- K_IntStop 2-4, 4-28
- K_MoveBufToArray 2-6, 4-30
- K_OpenDriver 2-2, 4-31
- K_SetAboutTrig 2-15, 4-33
- K_SetADTrig 2-12, 4-34
- K_SetBuf 2-7, 4-36
- K_SetBufI 2-7, 4-38
- K_SetChn 2-7, 4-40
- K_SetClk 2-8, 4-41
- K_SetClkRate 4-42
- K_SetDITrig 2-12, 4-44
- K_SetG 2-7, 4-46
- K_SetPostTrigDelay 2-13, 4-48
- K_SetTrig 2-10, 4-50
- memory management 4-2
- miscellaneous 4-3
- operation 4-2
- trigger 4-3

H

handle

- device 2-2, 3-1
- driver 2-2
- frame 3-2
- memory 2-6

I

- initialization functions 4-2
- initializing a board 2-2
- initializing the driver 2-2
- input ranges 2-7
 - bandwidth charts C-1
- internal pacer clock 2-8
- internal trigger 2-10
- interrupt-mode operations 2-4

K

- K_ClearFrame 3-3, 4-5
- K_CloseDriver 2-2, 4-6
- K_ClrAboutTrig 4-7
- K_DASDevInit 2-3, 4-8
- K_FreeDevHandle 2-2, 4-9
- K_FreeFrame 3-3, 4-10
- K_GetADFrame 3-2, 4-11
- K_GetClkRate 2-9, 4-13
- K_GetDevHandle 4-15
- K_GetErrMsg 2-4, 4-17
- K_GetShellVer 2-3, 4-18
- K_GetVer 2-3, 4-19
- K_IntAlloc 2-6, 4-21
- K_IntFree 2-6, 4-23
- K_IntStart 2-4, 4-24
- K_IntStatus 2-4, 4-25
- K_IntStop 2-4, 4-28
- K_MoveBufToArray 2-6, 4-30

- K_OpenDriver 2-2, 4-31
- K_SetAboutTrig 2-15, 4-33
- K_SetADTrig 2-12, 4-34
- K_SetBuf 2-7, 4-36
- K_SetBufI 2-7, 4-38
- K_SetChn 2-7, 4-40
- K_SetClk 2-8, 4-41
- K_SetClkRate 4-42
- K_SetDITrig 2-12, 4-44
- K_SetG 2-7, 4-46
- K_SetPostTrigDelay 2-13, 4-48
- K_SetTrig 2-10, 4-50

M

- maintenance operations: *see* system operations
- memory allocation 2-5
 - Visual Basic for Windows 3-15
- memory handle 2-6
- memory management functions 4-2
- Microsoft C/C++
 - compile and link statements for DOS 3-11
 - compile and link statements for Windows 3-12
 - creating an executable file for DOS 3-11
 - creating an executable file for Windows 3-12
 - dimensioning a local array 3-8
 - dynamically allocating a memory buffer 3-9
 - files required for DOS 3-11
 - files required for Windows 3-12
 - handling errors 3-10
- Microsoft Visual Basic for Windows
 - see* Visual Basic for Windows
- miscellaneous functions 4-3
- miscellaneous operations: *see* system operations

O

operation functions 4-2
operations
 analog input 2-4
 system 2-1

P

pacemaker clocks 2-8
post-trigger acquisition 2-13
pre-trigger acquisition 2-14
programming information
 C/C++ 3-8
 Visual Basic for Windows 3-15
programming overview 3-5
programming tasks
 analog input operations 3-6
 preliminary 3-6

R

ranges 2-7
 bandwidth charts C-1
return values 2-3
revision levels 2-3
routines: *see* functions

S

software trigger: *see* internal trigger
specifying an analog trigger level B-3
starting an operation 2-4
status 2-4
status codes 2-3
stopping an operation 2-3, 2-4
system operations 2-1

T

tasks: *see* programming tasks
trigger functions 4-3
trigger level, specifying an analog trigger
 B-3
triggers 2-10
 about-trigger acquisition 2-15
 external analog trigger source 2-11
 external digital trigger source 2-12
 internal trigger source 2-10
 post-trigger acquisition 2-13
 pre-trigger acquisition 2-14

V

Visual Basic for Windows
 creating an executable file 3-18
 dimensioning a local array 3-15
 dynamically allocating memory buffers
 3-16
 files required 3-18
 handling errors 3-17
voltage input ranges 2-7
 bandwidth charts C-1