DAS-4200 Series
Function Call Driver

USER'S GUIDE

DAS-4200 Series
Function Call Driver
User's Guide

Revision B - May 1996
Part Number: 94510

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, 1996.

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

The *DAS-4200 Series Function Call Driver User's Guide* describes how to write programs for DAS-4200 Series boards using the DAS-4200 Series Function Call Driver. The DAS-4200 Series Function Call Driver supports the following DOS-based languages:

- Microsoft® C/C++ (Version 6.0 and higher)
- Borland® C/C++ (Version 1.0 and higher)

The DAS-4200 Series Function Call Driver supports the following Windows-based languages:

- Microsoft C/C++ (Version 7.0 and higher)
- Microsoft Visual C++ (Version 1.0 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 programmers using a DAS-4200 Series board in an IBM® PC AT® or compatible computer. It is assumed that users have read the *DAS-4200 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-4200 Series Function Call Driver User's Guide* is organized as follows:

- Chapter 1 contains installation information, a brief description of available functions, and an illustration of the procedures to follow when programming a DAS-4200 Series board using the DAS-4200 Series Function Call Driver. The last section of this chapter explains how to get help.
- Chapter 2 contains the background information needed to use the functions included in the DAS-4200 Series Function Call Driver.
- Chapter 3 contains a programming overview and language-specific information related to using the DAS-4200 Series Function Call Driver.
- Chapter 4 contains detailed descriptions of the DAS-4200 Series Function Call Driver functions, arranged in alphabetical order.
- Appendix A contains a list of the error codes returned by DAS-4200 Series Function Call Driver functions.
- Appendix B contains instructions for converting counts to voltage and for converting voltage to counts.

An index completes this manual.

Keep the following conventions in mind as you use this manual:

- References to DAS-4200 Series boards apply to both the DAS-4201/128K and the DAS-4201/512K boards. When a feature applies to a particular board, that board's name is used.
- Unless otherwise noted, references to Windows include Windows 3.1, Windows 3.11 for Workgroups, and Windows 95.
- Keyboard keys and function names are represented in bold typeface.

Table of Contents

Preface

1 Overview

Features 1-1
Supporting Software. 1-2
Accessories. 1-3

2 Technical Reference

Analog-to-Digital Converter (ADC) 2-3
Reference Voltage Range and Vernier Gain 2-3
Static Conversion Errors 2-3
Noise Gain. 2-6
Dynamic Conversion Errors 2-6
Channels. 2-7
Input Ranges 2-8
Memory 2-8
Onboard Memory 2-8
Buffer Memory. 2-9
Nonvolatile Memory (EEPROM). 2-9
Host Computer Memory 2-10
I/O Address Space 2-10
Memory Address Space 2-10
Bus Interface 2-12
Counters 2-12
Pacer Clocks. 2-13
Internal Pacer Clock 2-13
External Pacer Clock. 2-14
Adjusting the Duty Cycle 2-14
Triggers 2-15
Trigger Sources 2-15
Software Trigger. 2-15
Analog Trigger 2-16
Digital Trigger 2-18

Trigger Acquisition 2-18
Post-Trigger Acquisition 2-18
About-Trigger Acquisition 2-21
Trigger Synchronization 2-24
Equivalent Time Sampling (ETS) 2-25
Peak Detector 2-28

3 Setup and Installation
Unpacking the Board 3-1
Installing the Software 3-2
Installing the DAS-4200 Series Standard Software Package . 3-2
Installing the ASO-4200 Software Package 3-3
DOS Installation 3-3
Windows Installation 3-4
Configuring the Board 3-5
Creating a Configuration File 3-7
Setting Jumpers on the Board 3-9
Setting the Base I/O Address 3-11
Setting the Memory Address 3-12
Excluding the Memory Area 3-13
Setting the Interrupt Level 3-14
Setting Trigger Synchronization. 3-15
Setting the Input Impedance for Analog Input Channels 3-15
Setting the Input Impedance for the Clock I/O Connector 3-17
Setting the Input Impedance for the Trigger I/O Connector 3-19
Adding a Ground Connection 3-20
Installing the Board 3-21
Attaching Applications. 3-22

4 Scope and Test Program
Control Keys for D4200.EXE 4-1
Scope and Test Program Menus 4-4
A/D Menu 4-4
Display Menu 4-7
Gates Menu 4-9
Options Menu 4-11
Configuration Menu 4-11
Saving Waveforms 4-12

Recalling Waveforms 4-13
Calibrating the DAS-4200 Series Board 4-14
Using Parameter Files 4-15

5 Troubleshooting
Identifying Symptoms and Possible Causes 5-1
Testing Board and Host Computer. 5-3
Testing Accessory Slot and I/O Connections. 5-4
Technical Support. 5-4

A Specifications

B Keithley Memory Manager
Installing and Setting Up the KMM in Windows 3.x. B-2
Using KMMSETUP.EXE B-2
Using a Text Editor B-3
Installing and Setting Up the KMM in Windows 95 . . . . . B-4
Removing the KMM B-4

Index

List of Figures

Figure 2-1. DAS-4200 Series Functional Block Diagram 2-1
Figure 2-2. Ideal Transfer Function of a 3-Bit ADC 2-4
Figure 2-3. Non-Ideal Transfer Function of a 3-Bit ADC 2-5
Figure 2-4. Host Computer Memory Address Space 2-11
Figure 2-5. Analog Trigger Modes. 2-17
Figure 2-6. Post-Trigger Acquisition 2-19
Figure 2-7. Memory Usage in Post-Trigger Acquisition 2-20
Figure 2-8. About-Trigger Acquisition 2-22
Figure 2-9. Memory Usage in About-Trigger Acquisition. . .2-23
Figure 2-10. Possible Trigger Position. 2-24
Figure 2-11. Trigger Jitter with Synchronized Divider 2-24
Figure 2-12. Equivalent Time Sampling (ETS) 2-25
Figure 2-13. ETS Delay 2-27
Figure 2-14. Peak Detection. 2-29
Figure 3-1. Jumper Locations. 3-10
Figure 3-2. Analog Input Circuitry. 3-16
Figure 3-3. Clock I/O Connector Circuitry 3-18
Figure 3-4. Trigger I/O Connector Circuitry 3-19

List of Tables

Table 2-1. Analog Input Ranges 2-8
Table 2-2. Available Conversion Rates Using Internal Clock 2-13
Table 3-1. Configuring DAS-4200 Series Boards 3-6
Table 3-2. Base I/O Address 3-11
Table 3-3. Memory Address 3-13
Table 3-4. Interrupt Level Selection 3-14
Table 3-5. Changing the Input Impedance 3-17
Table 4-1. Control Keys 4-2
Table 4-2. Suffixes 4-4
Table 4-3. A/D Menu 4-5
Table 4-4. Display Menu. 4-7
Table 4-5. Gates Menu 4-10
Table 4-6. Options Menu 4-11
Table 5-1. Troubleshooting Information. 5-1
Table A-1. DAS-4200 Series Specifications A-1

Table 1-1. Summary of Functions. 1-3
Table 2-1. A/D Frame Elements 2-6
Table 2-2. Analog Input Ranges and Gains 2-10
Table 2-3. Conversion Rates and Sample Periods
for the Internal Pacer Clock2-11
Table 3-1. Protected-Mode Memory Architecture 3-12
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. Interrupt-Mode Operation 2-5
Figure 2-2. Analog Trigger Conditions 2-14
Figure 2-3. Digital Trigger Conditions. 2-15
Figure 4-1. Status Word Settings 4-26

1

Getting Started

This chapter contains the following sections:

- **Overview** - a description of the DAS-4200 Series Function Call Driver.
- **Summary of Functions** - a brief description of the DAS-4200 Series Function Call Driver functions.
- **Programming Flow Diagrams** - an illustration of the procedures to follow when programming a DAS-4200 Series board using the DAS-4200 Series Function Call Driver.
- **Getting Help** - information on how to get help when installing or using the DAS-4200 Series Function Call Driver.

Overview

The DAS-4200 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 **ASO-4200** software package, which includes the following:

- Libraries of FCD functions for Microsoft C/C++ 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-4200 Series 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-4200 Series User's Guide*.

Summary of Functions

Table 1-1 provides a brief description of the functions in the DAS-4200 Series Function Call Driver. For more detailed information about the functions, refer to Chapter 4.

Table 1-1. Summary of Functions

Function Type	Function Name	Description
Initialization	K_OpenDriver	Initializes any Function Call Driver.
	K_CloseDriver	Closes a Function Call Driver.
	K_GetDevHandle	Initializes any Keithley DAS board.
	K_FreeDevHandle	Frees a device handle.
	K_DASDevInit	Reinitializes a board.
Operation	K_IntStart	Starts an interrupt-mode operation.
	K_IntStatus	Gets the status of an interrupt-mode operation.
	K_IntStop	Stops an interrupt-mode operation.
Frame Management	K_GetADFrame	Accesses a frame for an analog input operation.
	K_FreeFrame	Frees a frame.
	K_ClearFrame	Sets all frame elements to their default values.
Memory Management	K_IntAlloc	Dynamically allocates a memory buffer for an interrupt-mode operation.
	K_IntFree	Frees a memory buffer that was dynamically allocated for an interrupt-mode operation.
	K_MoveBufToArray	Transfers data from a dynamically allocated memory buffer to a local integer array.
Buffer Address	K_SetBuf	Specifies the starting address of a local array (C/C++) or a dynamically allocated memory buffer (C/C++, Visual Basic for Windows) for an interrupt-mode operation.
	K_SetBufI	Specifies the starting address of a local array (Visual Basic for Windows) for an interrupt-mode operation.

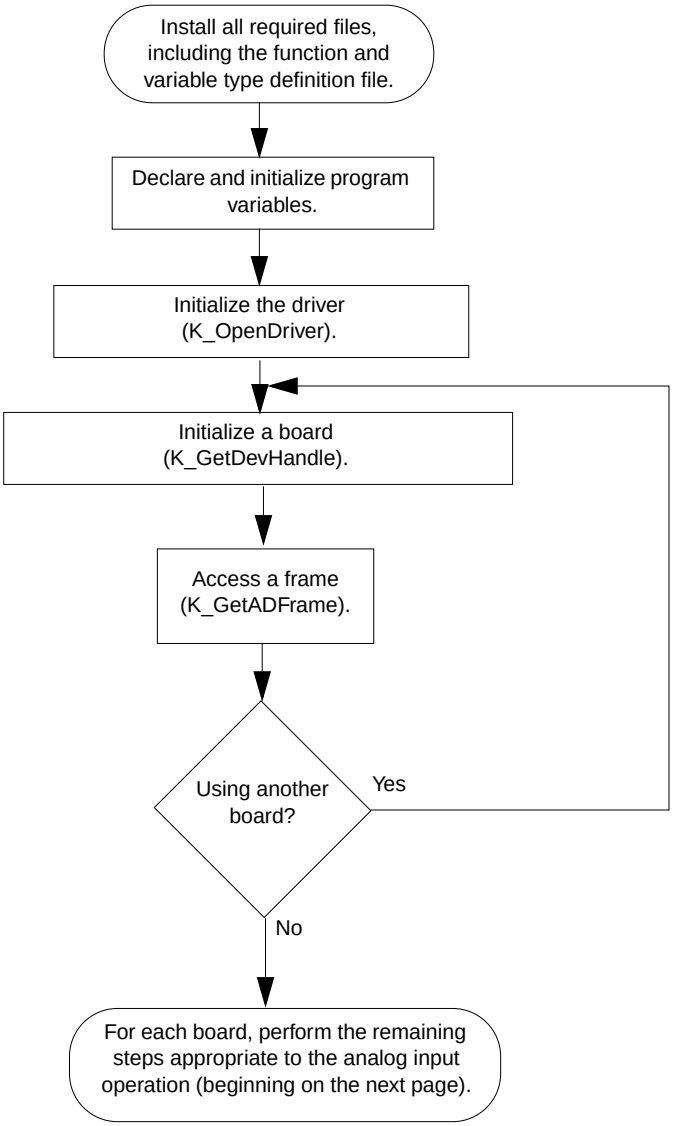
Table 1-1. Summary of Functions (cont.)

Function Type	Function Name	Description
Channel and Gain	K_SetChn	Specifies the channel to use for the operation.
	K_SetG	Specifies the gain for the specified channel.
Clock	K_SetClk	Specifies the pacer clock source.
	K_SetClkRate	Specifies the clock rate for the internal pacer clock.
	K_GetClkRate	Gets the clock rate for the internal pacer clock.
Trigger	K_SetTrig	Specifies the trigger source.
	K_SetADTrig	Sets up an external analog trigger.
	K_SetDITrig	Sets up an external digital trigger.
	K_SetAboutTrig	Enables the about trigger and specifies the number of post-trigger samples.
	K_ClrAboutTrig	Disables the about trigger.
Miscellaneous	K_GetErrMsg	Gets the address of an error message string.
	K_GetVer	Gets revision numbers.
	K_GetShellVer	Gets the current DAS shell version.

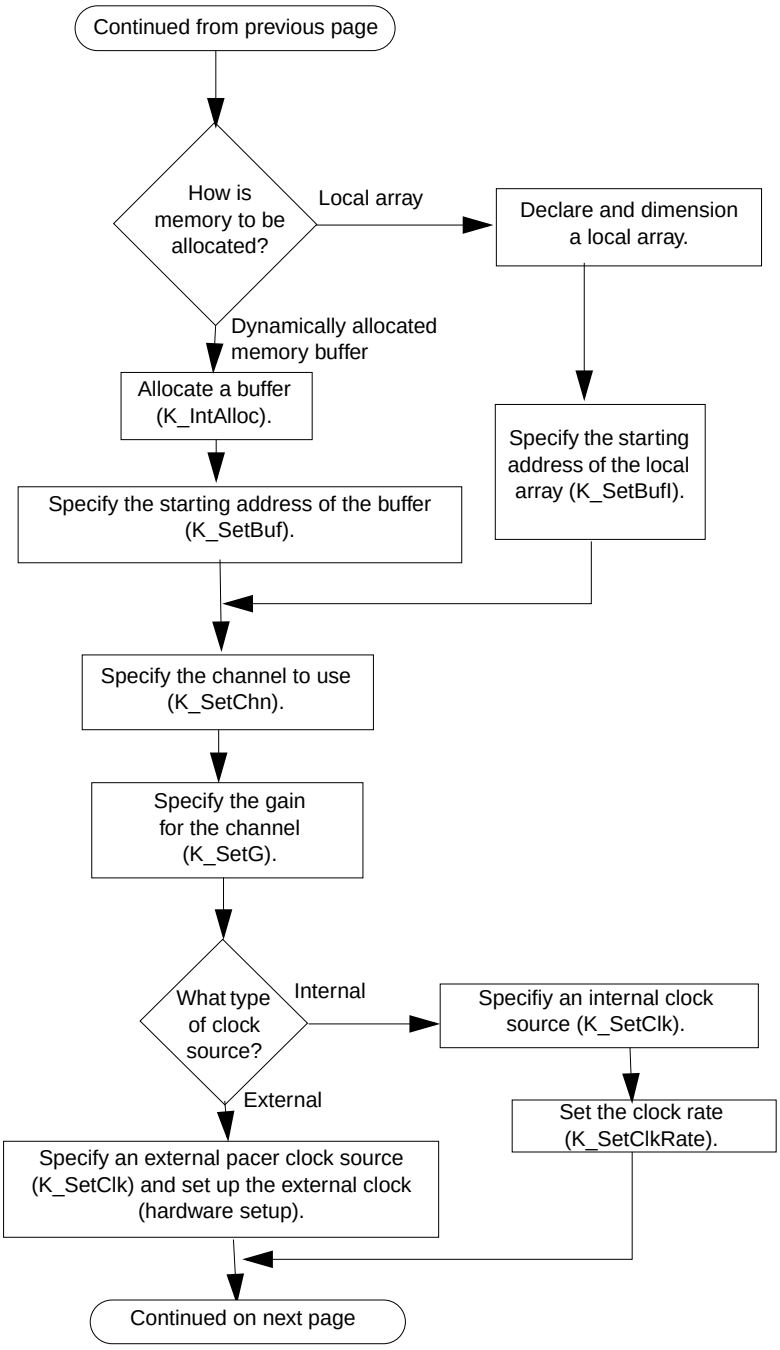
Programming Flow Diagrams

This section contains a series of programming flow diagrams illustrating the procedures used when programming each of the operations supported by the DAS-4200 Series Function Call Driver. Although error checking is not shown in the flow diagrams, it is recommended that you check the error/status code returned by each function used in your program.

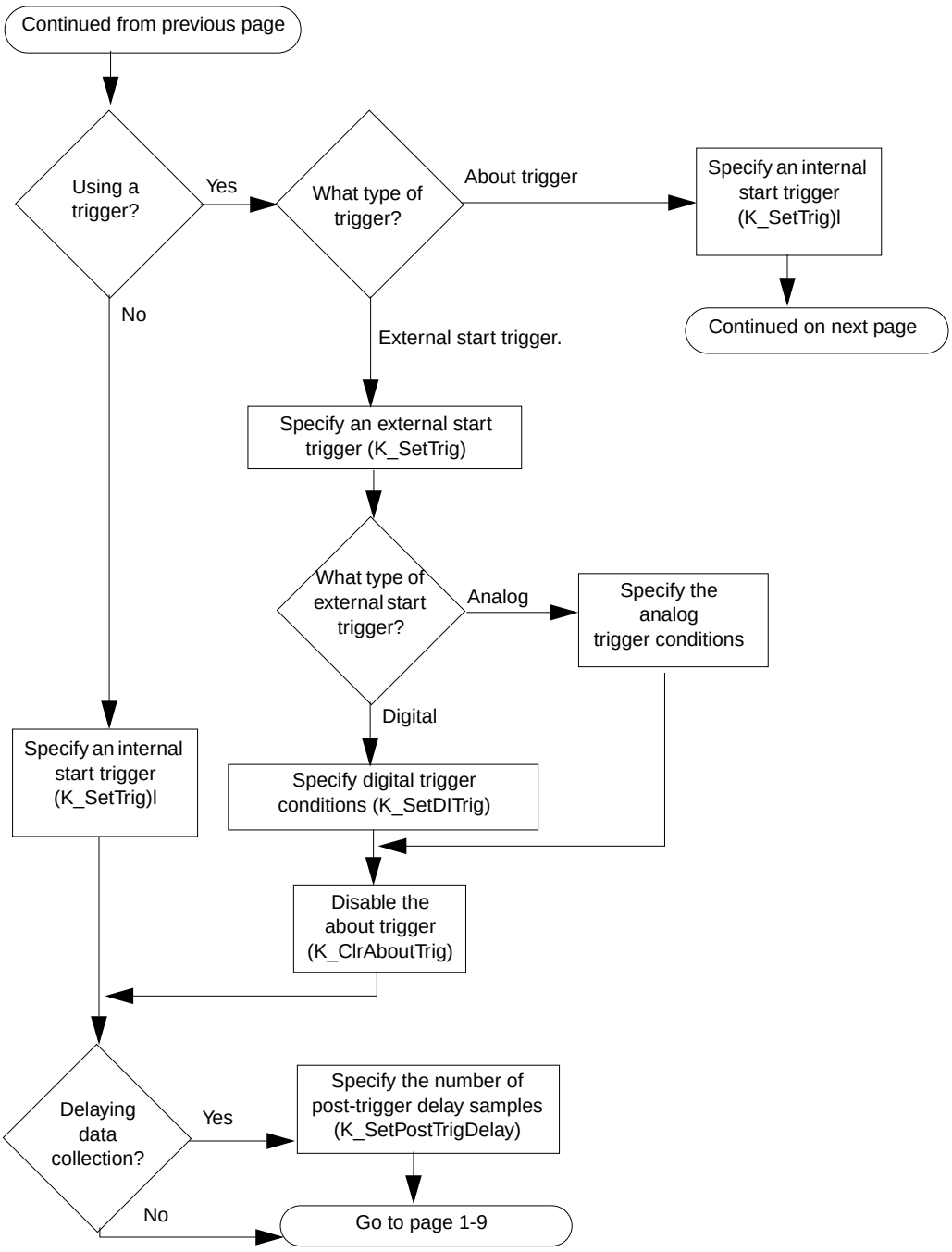
Preliminary Steps



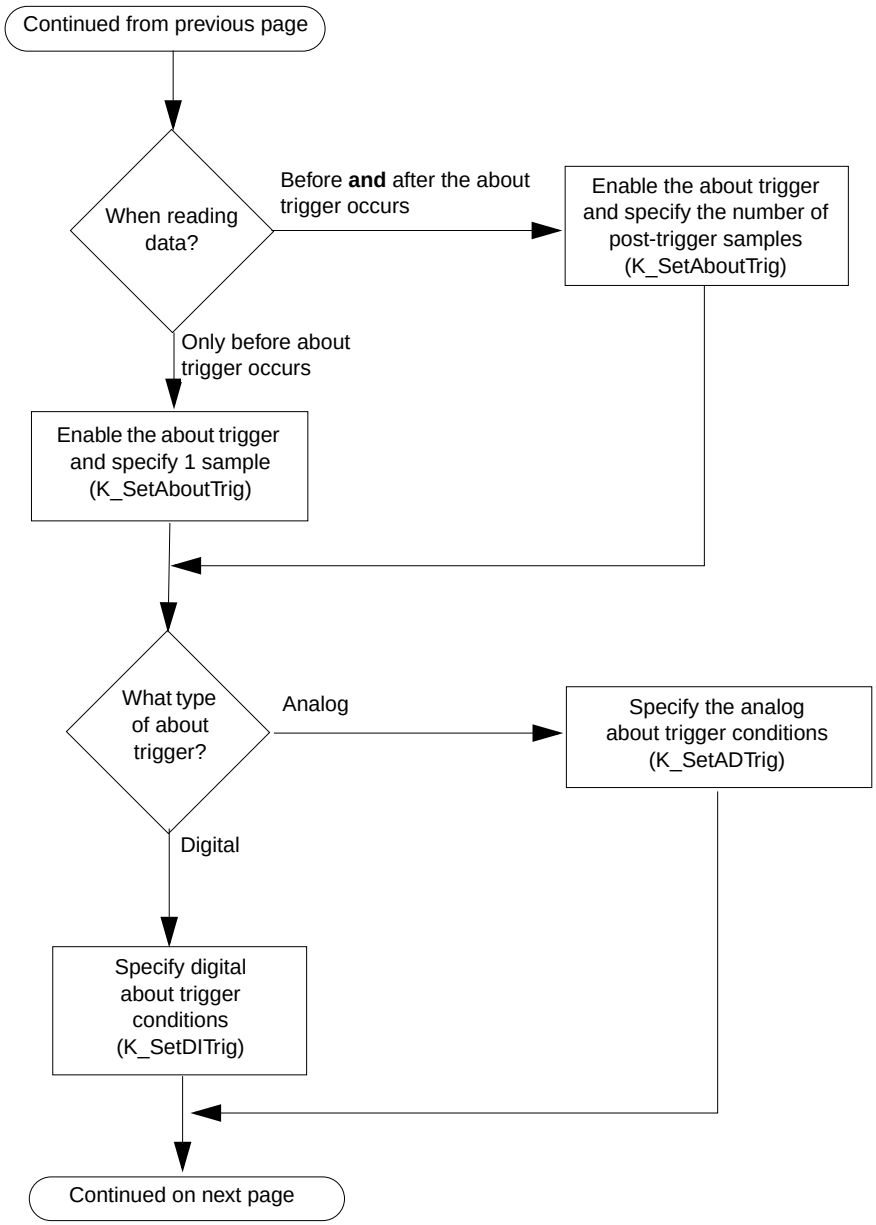
Steps for an Analog Input Operation



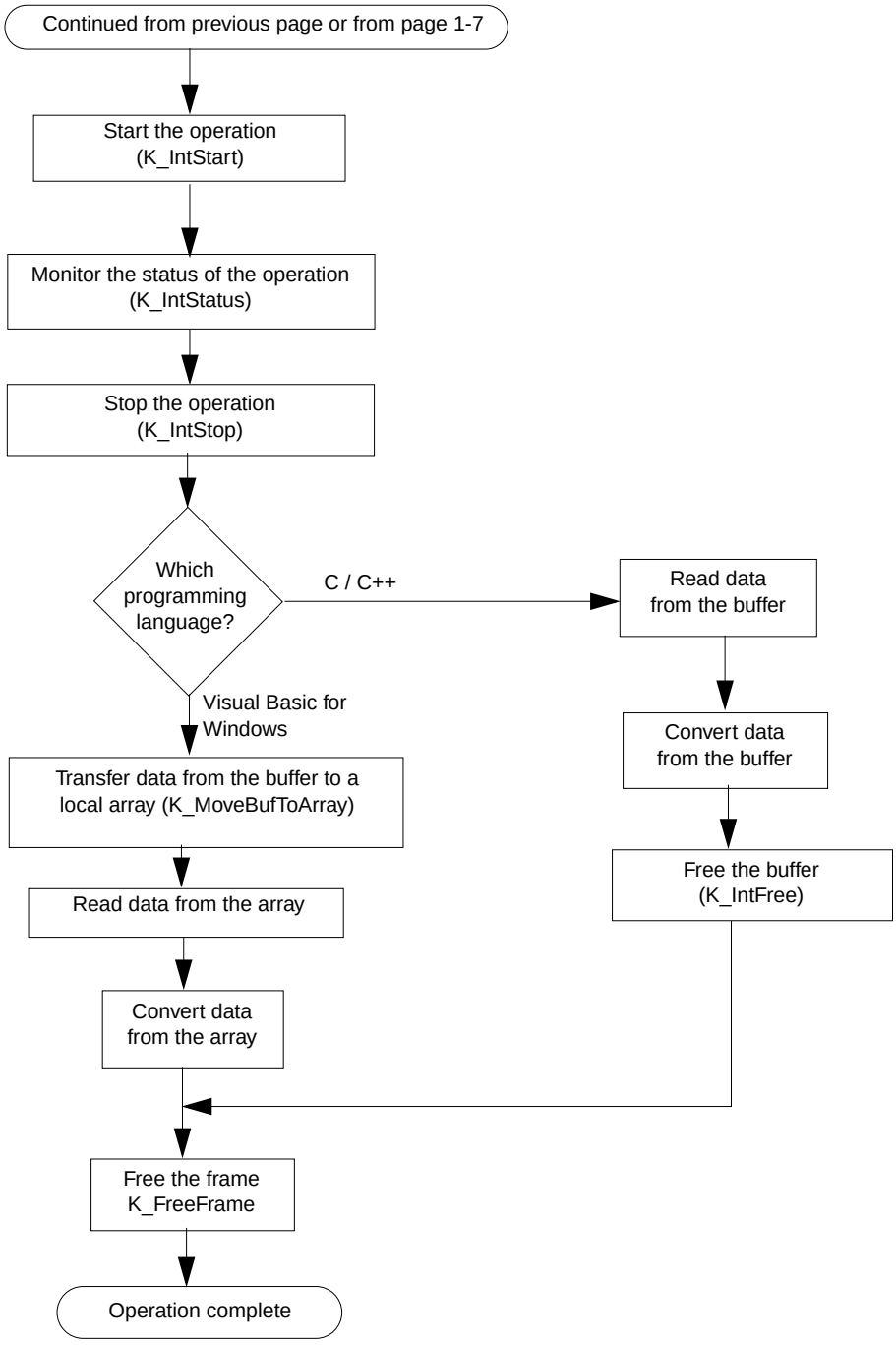
Steps for an Analog Input Operation (cont.)



Steps for an Analog Input Operation (cont.)



Steps for an Analog Input Operation (cont.)



Getting Help

If you need help installing or using the DAS-4200 Series Function Call Driver, call your local sales office or call the following number for technical support:

(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 information on the following page available before you call.

DAS-4200Series board configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Base address setting	_____
	Interrupt level setting	_____
	Input configuration	single-ended, differential
	Input range type	unipolar, bipolar
Computer	Manufacturer	_____
	CPU type	_____
	Clock speed (MHz)	_____
	Amount of RAM	_____
	Video system	_____
	BIOS type	_____
Operating system	DOS version	_____
	Windows version	_____
Software package	Name	_____
	Serial #	_____
	Version	_____
	Invoice/Order #	_____
Compiler (if applicable)	Language	_____
	Manufacturer	_____
	Version	_____
Accessories	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____

2

Available Operations

This chapter contains the following sections:

- **System Operations** - descriptions of the miscellaneous operations and general maintenance operations that apply to DAS-4200 Series boards and to the DAS-4200 Series Function Call Driver.
- **Analog Input Operations** - description of the operation mode available for analog input operations and instructions for allocating memory and setting parameters for an analog input operation.

System Operations

This section describes the miscellaneous operations and general maintenance operations that apply to DAS-4200 Series boards and to the DAS-4200 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-4200 Series Function Call Driver and any other Keithley DAS Function Call Drivers you are using in your 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 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-4200 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 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.

Use **K_GetDevHandle** the first time you initialize a board only. Once you have a device handle, you can reinitialize a board as needed by using the **K_DASDevInit** function.

Retrieving Revision Levels

If you are using functions from different Keithley DAS Function Call Drivers in the same program or if you are having problems with your 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-4200 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 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 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 3-5
Visual Basic for Windows	page 3-14

For C-language programs only, the DAS-4200 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-4200 Series 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 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 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 the operation. Use the **K_IntStatus** function to determine the current status of the operation.

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

Frames

The DAS-4200 Series Function Call Driver uses frames to perform interrupt-mode analog input operations. A frame is a data structure whose elements define the attributes of the operation. For each board you are using in a program, use the **K_GetADFrame** function to access an analog input frame, called an A/D (analog-to-digital) frame. The driver returns a unique identifier for the frame; this identifier is called the frame handle.

Specify the attributes of the operation by using a separate setup function to define each element of the A/D frame. Use the frame handle returned by the driver in each setup function to ensure that you always define the same operation. For example, assume that you access an A/D frame with the frame handle ADFrame. To specify the channel on which to perform the operation, use the **K_SetChn** setup function, referencing the frame handle ADFrame. To specify the gain at which to read the channel, use the **K_SetG** setup function, also referencing the frame handle ADFrame.

When you are ready to perform the operation you have set up, use the **K_IntStart** function to start the operation, again referencing the appropriate frame handle. Figure 2-1 illustrates the use of an A/D frame for an interrupt-mode operation, where the frame handle is ADFrame.

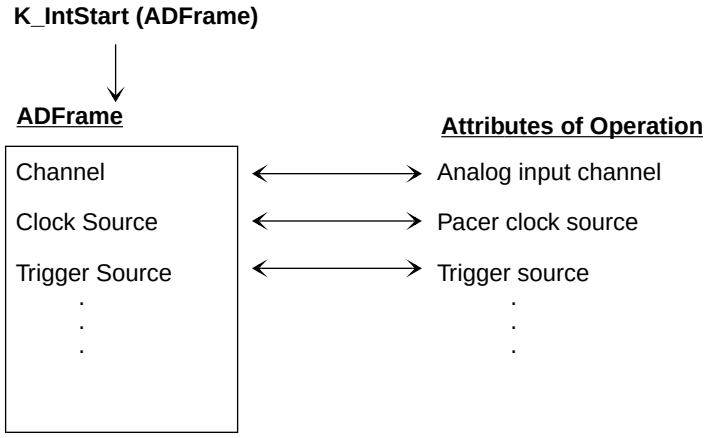


Figure 2-1. Interrupt-Mode Operation

Frames help you create structured programs. They 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.

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 2-1 lists the elements of an A/D frame, the default value of each element, the setup function used to define each element, and the pages in this guide on which to find information specific to the function.

Table 2-1. A/D Frame Elements

Element	Default Value	Setup Function	See Also
Buffer[1]	0 (NULL)	K_SetBuf	page 4-37
		K_SetBufI	page 4-39
Number of Samples	0	K_SetBuf	page 4-37
		K_SetBufI	page 4-39
Channel	0	K_SetChn	page 4-41
Gain	0 (gain of 1)	K_SetG	page 4-47
Clock Source	Internal A/D pacer clock	K_SetClk	page 4-42
Pacer Clock Rate[1]	0	K_SetClkRate	page 4-43
Trigger Source	Internal	K_SetTrig	page 4-49
Trigger Type	Digital	K_SetADTrig	page 4-35
		K_SetDITrig	page 4-45
Trigger Channel	0 (for analog trigger)	K_SetADTrig	page 4-35
	0 (for digital trigger)	Not applicable[2]	page 4-45

Table 2-1. A/D Frame Elements (cont.)

Element	Default Value	Setup Function	See Also
Trigger Polarity	Positive	K_SetADTrig	page 4-35
		K_SetDITrig	page 4-45
Trigger Sensitivity	Edge (for analog)	K_SetADTrig	page 4-35
	Edge (for digital)	Not applicable[2]	page 4-45
Trigger Level	0	K_SetADTrig	page 4-35
About-Trigger Acquisition	Disabled	K_SetAboutTrig	page 4-33
		K_ClrAboutTrig[3]	page 4-7

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.

Memory Allocation and Management

Interrupt-mode analog input operations require a memory buffer in which to store acquired data. DAS-4201/128K boards can acquire 131,072 samples (128K); DAS-4201/512K boards can acquire 524,288 samples (512K).

Note: Even though you can allocate a memory buffer greater than the board requires, it is recommended that you allocate a maximum buffer size of 128K for DAS-4201/128K boards and 512K for DAS-4201/512K boards.

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

Dynamically Allocating a Memory Buffer

You can allocate a memory buffer dynamically outside of your program's memory area. This way is recommended for the DAS-4200 Series boards. 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.

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. 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.

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 4-30 for more information.

Notes: If you are writing Windows 95, 32-bit programs, you must install the Keithley Memory Manager. See your board user's guide for details.

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 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.

Dimensioning a Local Array

A simpler way to reserve a memory location is to dimension an array within your program's memory area. The advantage of this method is that the array is directly accessible to your program. The limitations of this method are as follows:

- Certain programming languages limit the size of local arrays.
- Local arrays occupy permanent memory areas; these memory areas cannot be freed to make them available to other programs or processes.
- You cannot use local arrays with Windows 95, 32-bit programs.

Because of these limitations and because the DAS-4200 Series boards can store up to 524,288 samples onboard, dimensioning a local array is not recommended.

Since the DAS-4200 Series Function Call Driver stores data in 16-bit integers, you must dimension all local arrays as integers.

Assigning the Starting Address

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

Language	Memory Location	Function	Refer to
C/C++	Array or Buffer	K_SetBuf	page 3-4
Visual Basic for Windows	Array	K_SetBufI	page 3-11
	Buffer	K_SetBuf	page 3-10

Channels

DAS-4200 Series boards provide two analog input channels; the software refers to Channel A as channel 0 and Channel B 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.

Gains and Ranges

Each channel on the DAS-4200 Series board can measure signals in one of eight, software-selectable bipolar analog input ranges.

Table 2-2 lists the analog input ranges supported by DAS-4200 Series boards and the gain and gain code associated with each range. Use the **K_SetG** function to specify the gain code (the gain code is used by **K_SetG** to represent the gain).

Table 2-2. Analog Input Ranges and Gains

Analog Input Range	Gain	Gain Code
±2 V	1	0
±1 V	2	1
±500 mV	4	2
±250 mV	8	3
±125 mV	16	4
±62.5 mV	32	5
±31.25 mV	64	6
±15.625 mV	128	7

Pacer Clocks

The pacer clock determines the period between A/D conversions. You can specify an internal or an external pacer clock, as described in the following subsections. Refer to the *DAS-4200 Series User's Guide* for more information.

Internal Pacer Clock

The internal pacer clock is the 100-MHz oscillator on the DAS-4200 Series board. The default clock source is internal; to reset the clock source to the internal clock, use the **K_SetClk** function.

When you start an analog input operation (using **K_IntStart**), conversions are performed at a rate of 3.2 Gsamples/s divided by a clock divider value. Use the **K_SetClkRate** function to specify the clock divider value. Table 2-3 lists the supported clock divider values and the corresponding conversion rates and sample periods for the internal pacer clock.

Table 2-3. Conversion Rates and Sample Periods for the Internal Pacer Clock

Clock Divider	Conversion Rate	Sample Period
32	100 Msamples/s	10 ns
64	50 Msamples/s	20 ns
128	25 Msamples/s	40 ns
256	12.5 Msamples/s	80 ns
512	6.25 Msamples/s	160 ns
1024	3.13 Msamples/s	320 ns
2048	1.56 Msamples/s	640 ns
4096	0.78 Msamples/s	1280 ns

Note: If you enter a clock divider 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 clock divider value of 63, the driver uses a clock divider value of 32 to perform the faster conversion rate. To determine the actual clock divider value used, use the **K_GetClkRate** function.

External Pacer Clock

You connect an external clock to the Clock I/O connector of the DAS-4200 Series board. To specify an external clock source, use the **K_SetClk** function.

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.

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). 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.

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

Trigger Sources

You can specify an internal or an external trigger source. An internal trigger is a software trigger. 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. To reset the trigger source to internal, use the **K_SetTrig** function.

External Analog Trigger

You can use the signal on the analog input channel being sampled as the trigger signal for an analog trigger. The trigger conditions for analog triggers are illustrated in Figure 2-2 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.
- **Positive-Level Trigger** - A trigger event occurs the first time the trigger signal is a voltage that is greater than the trigger level.
- **Negative-Level Trigger** - If the trigger polarity is negative and the sensitivity is level, a trigger event occurs the first time the trigger signal is a voltage that is less than the trigger level.

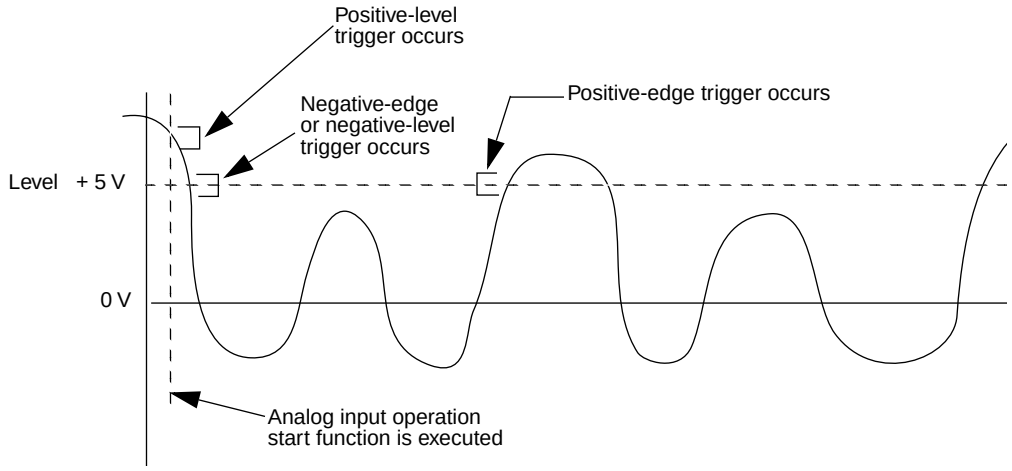


Figure 2-2. Analog Trigger Conditions

To specify an external analog trigger, first use the **K_SetTrig** function to specify an external trigger. Then, use the **K_SetADTrig** function to specify the analog input channel to use as the trigger channel, the trigger level, and the trigger polarity and sensitivity. You specify the trigger level as a count value.

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 Trigger I/O connector of the DAS-4200 Series boards. To specify an external digital trigger, first use the **K_SetTrig** function to specify an external trigger. Then, use the **K_SetDITrig** function to specify 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 events are illustrated in Figure 2-3.

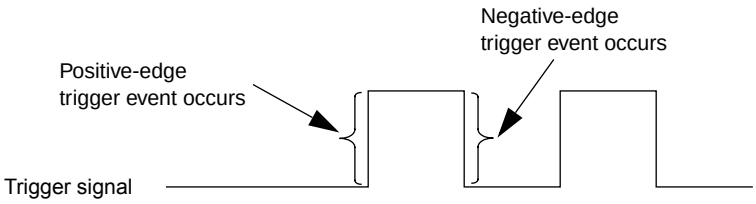


Figure 2-3. Digital Trigger Conditions

Trigger Acquisition

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.

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.

Pre-Trigger Acquisition

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

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

1. Use **K_SetTrig** to specify an internal start-trigger source.
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 start trigger is always an internal trigger; the operation starts when your program calls the **K_IntStart** function. The operation stops after a specified number of samples has been acquired after the about-trigger event occurs. The about trigger can be either an external analog or external digital trigger.

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

1. Use **K_SetTrig** to specify an internal start-trigger source.
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 buffer.

3

Programming with the Function Call Driver

This chapter contains a programming overview and language-specific information related to using the DAS-4200 Series Function Call Driver. It includes the following sections:

- **Programming Overview -** an overview of the tasks required to write a program using the DAS-4200 Series Function Call Driver.
- **C/C++ Programming Information -** language-specific information for programming in Microsoft C/C++ (including Visual C++) and Borland C/C++.
- **Visual Basic for Windows Programming Information -** language-specific information for programming in Microsoft Visual Basic for Windows.

Programming Overview

To write a program using the DAS-4200 Series Function Call Driver, perform the following steps:

1. Define the program'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 program. Refer to the following for additional information:
   - Programming flow diagrams for the preliminary tasks, on page 1-5, which illustrate the programming tasks common to all programs.
   - Programming flow diagrams for an analog input operation, on page 1-6.
   - Chapter 4, which contains detailed descriptions of the FCD functions.
   - The example programs in the ASO-4200 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 3-2 to page 3-9 for C/C++ or page 3-15 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.

C/C++ Programming Information

The following sections contain information you need to allocate and assign a memory buffer when programming in C or C++, as well as language-specific information for Microsoft C/C++ (including Visual 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. Make sure that linker options are set so that case-sensitivity is disabled.

Dynamically Allocating and Assigning a Memory Buffer

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

Note: To ensure that you can allocate a large enough buffer or buffers, it is recommended that you install the Keithley Memory Manager before you begin programming. Refer to the *DAS-4200 Series User's Guide* for information on the Keithley Memory Manager.

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 Data from a Dynamically Allocated Memory Buffer

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.

Dimensioning a Local Array

Although it is not generally recommended for this driver, you can use a single, local array for an interrupt-mode analog input operation. A local array is useful when you are acquiring small amounts of data (less than 32,767 samples). 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.

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);
    }
. . .
```

The following code fragment illustrates how to use the **K_GetErrMsg** function to access the string corresponding to an error code.

```
  . . .
  if ((wDasErr = K_SetChn (hAD, 0) ! = 0)
    {
    Error = K_GetErrMsg (hDev, wDasErr, &pMessage);
    printf ("%s", pMessage);
    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-4200 software package.

File	Description
DAS4200.LIB	Linkable driver
DASRFACE.LIB	Linkable driver
DASDECL.H	Include file when compiling in C
DASDECL.HPP	Include file when compiling in C++
USE4200.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 program.

Type of Compile	Compile and Link Statements[1]
C	CL /c *filename*.c LINK *filename*+use4200.obj,,,das4200+dasrface;
C++	CL /c *filename*.cpp LINK *filename*+use4200.obj,,,das4200+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)

The files you need to program in Microsoft C/C++ (for Windows), including Microsoft Visual C++, depend on whether you are writing 16-bit or 32-bit programs. The following files are provided either in the ASO-4200 software package or on the ASO-Win95/32-Bit disk, which is shipped with the ASO-4200 software package:

Program	File	Description
16 bits	DASSHELL.DLL	Dynamic Link Library of Shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DAS4200.DLL	Dynamic Link Library of board-specific functions
	DASDECL.H	Include file of Shell function definitions (used when compiling in C or C++)
	DASIMP.LIB	Import library of Shell functions
32-bits	DASSHL32.DLL	Dynamic Link Library of Shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASSHL16.DLL	Dynamic Link Library of support functions
	DAS4200.DLL	Dynamic Link Library of board-specific functions
	DASDECL.H	Include file of Shell function definitions (used when compiling in C or C++)
	DASSHL32.LIB	Import library of Shell functions

To create an executable file in the Microsoft C/C++ for Windows environment, perform the following steps. Refer to the documentation supplied with your compiler for information.

1. Create a project file.
2. Add all necessary files to the project make file. Make sure that you include *filename*.c (or *filename*.cpp), *filename*.rc, *filename*.def, DASIMP.LIB (or DASSHL32.LIB), and D4200IMP.LIB (16-bit programs only), where *filename* indicates the name of your program.
3. Create a stand-alone executable file (.EXE) that you can run 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-4200 software package.

File	Description
DAS4200.LIB	Linkable driver
DASRFACE.LIB	Linkable driver
DASDECL.H	Include file when compiling in C
DASDECL.HPP	Include file when compiling in C++
USE4200.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 program.

Type of Compile	Compile and Link Statements[1]
C	BCC *filename*.c use4200.obj das4200.lib dasrface.lib
C++	BCC *filename*.cpp use4200.obj das4200.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)

The files you need to program in Borland C/C++ (for Windows) depend on whether you are writing 16-bit or 32-bit programs. The following files are provided either in the ASO-4200 software package or on the ASO-Win95/32-Bit disk, which is shipped with the ASO-4200 software package:

Program	File	Description
16 bits	DASSHELL.DLL	Dynamic Link Library of Shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DAS4200.DLL	Dynamic Link Library of board-specific functions
	DASDECL.H	Include file of Shell function definitions (used when compiling in C or C++)
	DASIMP.LIB	Import library of Shell functions
32-bits	DASSHL32.DLL	Dynamic Link Library of Shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASSHL16.DLL	Dynamic Link Library of support functions
	DAS4200.DLL	Dynamic Link Library of board-specific functions
	DASDECL.H	Include file of Shell function definitions (used when compiling in C or C++)
	DASSHL32.LIB	Import library of Shell functions

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

1. Create a project file.
2. Add all necessary files to the project make file. Make sure that you include *filename*.c (or *filename*.cpp), *filename*.rc, *filename*.def, DASIMP.LIB (or DASSHL32.LIB), and D4200IMP.LIB (16-bit programs only), where *filename* indicates the name of your program.

3. Make sure that you turn OFF the following options for the project:
   - Case sensitive link
   - Case sensitive exports and imports
   - Ignore default libraries
4. Create a stand-alone executable file (.EXE) that you can run 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.

Dynamically Allocating and Assigning a Memory Buffer

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

Note: To ensure that you can allocate a large enough buffer, it is recommended that you use the Keithley Memory Manager before you begin programming. Refer to your *DAS-4200 Series User's Guide* for more information on the Keithley Memory Manager.

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 Data from a Dynamically Allocated Memory Buffer

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.

When Windows is running, the CPU operates in 16-bit protected mode. Memory is addressed using a 32-bit selector:offset pair. The selector is the CPU's handle to a 64K byte memory page; it is a code whose value is significant only to the CPU. No mathematical relationship exists between a selector and the memory location it is associated with. In general, even consecutively allocated selectors have no relationship to each other.

When a memory buffer of more than 64K bytes (32K values) is used, multiple selectors are required. Under Windows, **K_IntAlloc** uses a "tiled" method to allocate memory whereby a mathematical relationship does exist among the selectors. Specifically, when you allocate a buffer of more than 64K bytes, each selector that is allocated has an arithmetic value that is eight greater than the previous one. The format of the address is a 32-bit value whose high word is the 16-bit selector value and low word is the 16-bit offset value. When the offset reaches 64K bytes, the

next consecutive memory address location can be accessed by adding eight to the selector and resetting the offset to zero; to do this, add &h80000 to the buffer starting address.

Table 3-1 illustrates the mapping of consecutive memory locations in protected-mode “tiled” memory, where *xxxxxxxx* indicates the address calculated by the CPU memory mapping mechanism.

Table 3-1. Protected-Mode Memory Architecture

Selector:Offset	32-Bit Linear Address
....:....	.....
32E6:FFFE	*xxxxxxxx*
32E6:FFFF	*xxxxxxxx* + 1
32EE:0000	*xxxxxxxx* + 2
32EE:0001	*xxxxxxxx* + 3
....:....	.....

The following code fragment illustrates moving 1,000 values from a memory buffer (AcqBuf) allocated with 50,000 values to the program’s local array (Array), starting at the sample at buffer index 40,000. First, start with the buffer address passed in **K_SetBuf**. Then, determine how deep (in 64K byte pages) into the buffer the desired starting sample is located and add &h80000 to the buffer address for each 64K byte page. Finally, add any additional offset after the 64K byte pages to the buffer address.

```
Dim AcqBuf As Long
Dim NumSamps As Long

Dim Array(1000) As Integer

NumSamps = 50000
wDasErr = K_IntAlloc (hFrame, NumSamps, AcqBuf, hMem)
.
. ’Acquisition routine
.
```

```
DesiredSamp = 40000
DesiredByte = DesiredSamp * 2       'Number of bytes into buffer
AddSelector = DesiredByte / &h10000 'Number of 64K pages into buffer
RemainingOffset = DesiredByte Mod &h10000    'Additional offset

DesiredBuffLoc = AcqBuf + (AddSelector * &h80000) + RemainingOffset

wDasErr = K_MoveBufToArray (Array(0), DesiredBuffLoc, 1000)
```

To move more than 32,767 values from the memory buffer to the program's local array, the program must call **K_MoveBufToArray** more than once. For example, assume that pBuf is a pointer to a dynamically allocated buffer that contains 65,536 values. The following code fragment illustrates how to move 65,536 values from the dynamically allocated buffer to a local array within the program:

```
...
Dim Data [3, 16384] As Integer
...
wDasErr = K_MoveBufToArray (Data(0,0), pBuf, 16384)

'Same selector, add 32,768 bytes to offset: add &h8000
wDasErr = K_MoveBufToArray (Data(1,0), pBuf + &h8000, 16384)
'Add 8 to selector, offset = 0: add &h80000
wDasErr = K_MoveBufToArray (Data(2,0), pBuf + &h80000, 16384)
'Add 8 to selector, add 32,768 bytes to offset: add &h88000
wDasErr = K_MoveBufToArray (Data(3,0), pBuf + &h88000, 16384)
```

Dimensioning and Assigning a Local Array

If you request fewer than 32,767 samples for an interrupt-mode analog input operation, you can use a single, local array. The following code fragment illustrates how to dimension an array of 8192 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.

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-4200 SERIES ERROR"
   End
End If
. . .
```

Programming in Microsoft Visual Basic for Windows

The files you need to program in Microsoft Visual Basic for Windows depend on whether you are writing a 16-bit or 32-bit program. The following files are provided either in the ASO-4200 software package or on the ASO-Win95/32-Bit disk, which is shipped with the ASO-4200 software package.

Program	File	Description
16 bits	DASSHELL.DLL	Dynamic Link Library
	DASSUPRT.DLL	Dynamic Link Library
	DAS4200.DLL	Dynamic Link Library
	DASDECL.BAS	Include file
	DAS4200.BAS	Include file of board-specific function definitions
32 bits	DASSHL32.DLL	Dynamic Link Library of Shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASSHL16.DLL	Dynamic Link Library of support functions
	DASDEC32.BAS	Include file of Shell function definitions

To create an executable file from the Microsoft Visual Basic for Windows environment, perform the following steps:

1. Start Visual Basic for Windows, and open your project.
2. Add the appropriate include file(s) to your project:
   - 16-bit programs - DASDECL.BAS and DAS4200.BAS files
   - 32-bit programs - DASDEC32.BAS file
3. Create an executable (EXE) file.

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 4-37
	K_SetBufI	page 4-39
Channel and Gain	K_SetChn	page 4-41
	K_SetG	page 4-47
Clock	K_SetClk	page 4-42
	K_SetClkRate	page 4-43
	K_GetClkRate	page 4-13
Trigger	K_SetTrig	page 4-49
	K_SetADTrig	page 4-35
	K_SetDITrig	page 4-45
	K_SetAboutTrig	page 4-33
	K_ClrAboutTrig	page 4-7

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
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-4200 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, frame handle, device handle, or 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, *anVar* 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.

Purpose

Sets the elements of a frame to their default values.

Prototype

C/C++

DASErr far pascal K_ClearFrame (DWORD *hFrame*);

Visual Basic for Windows

Declare Function K_ClearFrame Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame 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 *hFrame* to their default values.

Refer to Table 2-1 on page 2-6 for the default values of the elements of an A/D frame.

See Also

K_GetADFrame

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_ClearFrame (hAD)
```

K_CloseDriver

Purpose

Closes a previously initialized Keithley DAS Function Call Driver.

Prototype

C/C++

DASErr far pascal K_CloseDriver (DWORD *hDrv*);

Visual Basic for Windows

Declare Function K_CloseDriver Lib "DASSHELL.DLL"
(ByVal *hDrv* As Long) As Integer

Parameters

hDrv	Driver handle you want to free.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the driver handle specified by *hDrv* and closes the associated use of the Function Call Driver. This function also frees all device handles and frame handles associated with *hDrv*.

If *hDrv* 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++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_CloseDriver (hDrv)
```

K_ClrAboutTrig

Purpose

Disables the about trigger for an analog input operation.

Prototype

C/C++

DASErr far pascal K_ClrAboutTrig (DWORD *hFrame*);

Visual Basic for Windows

Declare Function K_ClrAboutTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame 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 *hFrame*.

If you disable the about trigger, the trigger source specified in **K_SetTrig** is always the start trigger.

K_GetADFrame and **K_ClearFrame** also disable the about trigger.

See Also

K_ClearFrame, K_GetADFrame, K_SetAboutTrig

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_ClrAboutTrig (hAD)
```

K_DASDevInit

Purpose

Reinitializes a board.

Prototype

C/C++

DASErr far pascal K_DASDevInit (DWORD *hDev*);

Visual Basic for Windows

Declare Function K_DASDevInit Lib "DASSHELL.DLL"
(ByVal *hDev* As Long) As Integer

Parameters

hDev Handle associated with the board.

Return Value

Error/status code. Refer to Appendix A.

Remarks

Use **K_GetDevHandle** the first time you initialize a board only. Once you have a device handle, use this function to reinitialize the board.

See Also

K_GetDevHandle

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_DASDevInit (hDev)
```

Purpose

Frees a previously specified device handle.

Prototype

C/C++

```
DASErr far pascal K_FreeDevHandle (DWORD hDev);
```

Visual Basic for Windows

```
Declare Function K_FreeDevHandle Lib "DASSHELL.DLL"
(ByVal hDev As Long) As Integer
```

Parameters

hDev	Device handle you want to free.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the device handle specified by *hDev* as well as all frame handles associated with *hDev.*

See Also

K_GetDevHandle

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_FreeDevHandle (hDev)
```

K_FreeFrame

Purpose

Frees a frame.

Prototype

C/C++

DASErr far pascal K_FreeFrame (DWORD *hFrame*);

Visual Basic for Windows

Declare Function K_FreeFrame Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to frame you want to free.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the frame specified by *hFrame*, making the frame available for another operation.

See Also

K_GetADFrame

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_FreeFrame (hAD)
```

Purpose

Accesses an A/D frame for an analog input operation.

Prototype

C/C++

DASErr far pascal K_GetADFrame (DWORD *hDev*,
DWORD far * *phFrame*);

Visual Basic for Windows

Declare Function K_GetADFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *phFrame* As Long) As Integer

Parameters

hDev	Handle associated with the board.
phFrame	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 analog input operation on the board specified by *hDev*, and accesses an available A/D frame with the handle *phFrame*. The frame is initialized to its default settings; the default settings are given in Table 2-1 on page 2-6.

The value stored in *phFrame* 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 *phFrame*.

See Also

K_ClearFrame, K_FreeFrame

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD hAD;
...
wDasErr = K_GetADFrame (hDev, &hAD);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

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

Purpose

Gets the clock divider 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 — Clock divider.
Value stored: See the table below.

The following table shows the clock divider values stored and the corresponding conversion rates and sample periods for the internal pacer clock

Clock Divider	Conversion Rate	Sample Period
32	100 Msamples/s	10 ns
64	50 Msamples/s	20 ns
128	25 Msamples/s	40 ns
256	12.5 Msamples/s	80 ns
512	6.25 Msamples/s	160 ns
1024	3.13 Msamples/s	320 ns
2048	1.56 Msamples/s	640 ns
4096	0.78 Msamples/s	1280 ns

Return Value

Error/status code. Refer to Appendix A.

K_GetClkRate (cont.)

Remarks

For the operation defined by *hFrame*, this function stores the clock divider for the internal pacer clock in *pRate*.

The *pRate* variable contains the value of the Pacer Clock Rate element.

See Also

K_SetClkRate

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD dwRate;
...
wDasErr = K_GetClkRate (hAD, &dwRate);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global dwRate As Long
...
wDasErr = K_GetClkRate (hAD, dwRate)
```

K_GetDevHandle

Purpose

Initializes any Keithley DAS board.

Prototype

C/C++

DASErr far pascal K_GetDevHandle (DWORD *hDrv*,
WORD *nBoardNum*, DWORD far * *phDev*);

Visual Basic for Windows

Declare Function K_GetDevHandle Lib "DASSHELL.DLL"
(ByVal *hDrv* As Long, ByVal *nBoardNum* As Integer, *phDev* As Long)
As Integer

Parameters

hDrv	Driver handle of the associated Function Call Driver.
nBoardNum	Board number. Valid values: **0** to **1**
phDev	Handle associated with the board.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function initializes the board associated with *hDrv* and specified by *nBoardNum*, and stores the device handle of the specified board in *phDev*.

The value stored in *phDev* 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 *phDev*.

Use this function the first time you initialize a board only. Once you have a device handle (*phDev*), use the **K_DASDevInit** function to reinitialize the board.

See Also

K_DASDevInit, K_FreeDevHandle

K_GetDevHandle (cont.)

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
DWORD phDev;
...
wDasErr = K_GetDevHandle (hDrv, 0, &phDev);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

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

Purpose

Gets the address of an error message string.

Prototype

C/C++

DASErr far pascal K_GetErrMsg (DWORD *hDev*, short *nDASErr*, char far * far * *pErrMsg*);

Visual Basic for Windows

Not supported

Parameters

hDev	Handle associated with the board.
nDASErr	Error message number.
pErrMsg	Address of error message string.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the address of the string corresponding to error message number *nDASErr* in *pErrMsg*.

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++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
char far *pErrMsg;
...
wDasErr = K_GetErrMsg (hDev, wDASErr, &pErrMsg);
```

K_GetShellVer

Purpose

Gets the current DAS shell version.

Prototype

C/C++

DASErr far pascal K_GetShellVer (WORD far **pVersion*);

Visual Basic for Windows

Declare Function K_GetShellVer Lib "DASSHELL.DLL"
(*pVersion* As Integer) As Integer

Parameters

pVersion	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 *pVersion*.

To obtain the major version number of the DAS shell, divide *pVersion* by 256. To obtain the minor version number of the DAS shell, perform a Boolean AND operation with *pVersion* and 255 (0FFh).

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 or DASDEC32.BAS to your project)

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

Purpose

Gets revision numbers.

Prototype

C/C++

DASErr far pascal K_GetVer (DWORD *hDev*, short far * *pSpecVer*, short far * *pDrvVer*);

Visual Basic for Windows

Declare Function K_GetVer Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pSpecVer* As Integer, *pDrvVer* As Integer)
As Integer

Parameters

hDev	Handle associated with the board.
pSpecVer	Revision number of the Keithley DAS Driver Specification to which the driver conforms.
pDrvVer	Driver version number.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the revision number of the DAS-4200 Series Function Call Driver in *pDrvVer* and the revision number of the driver specification in *pSpecVer*.

The values stored in *pSpecVer* and *pDrvVer* 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 2.10, the major revision level is 2 and the minor revision level is 10; therefore, the high byte of *pDrvVer* contains the value of **2** (512) and the low byte of *pDrvVer* contains the value of **10**; the value of both bytes is 522.

To obtain the major version number of the Function Call Driver, divide *pDrvVer* by 256; to obtain the minor version number of the Function Call Driver, perform a Boolean AND operation with *pDrvVer* and 255 (0FFh).

To obtain the major version number of the driver specification, divide *pSpecVer* by 256; to obtain the minor version number of the driver specification, perform a Boolean AND operation with *pSpecVer* and 255 (0FFh).

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 or DASDEC32.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))
Msgbox "Driver Version: " + DrvVer$
```

Purpose

Allocates a buffer for an analog input operation.

Prototype

C/C++

DASErr far pascal K_IntAlloc (DWORD *hFrame*, DWORD *dwSamples*, void far * far **pBuf*, WORD far **phMem*);

Visual Basic for Windows

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

Parameters

hFrame	Handle to the frame that defines the operation.
dwSamples	Number of samples. Valid values: **1** to **131072** for the DAS-4201/128K board **1** to **524288** for the DAS-4201/512K board
pBuf	Starting address of the allocated buffer.
phMem	Handle associated with the allocated buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function allocates a buffer of the size specified by *dwSamples*, and stores the starting address of the buffer in *pBuf* and the handle of the buffer in *phMem*.

The data in the allocated buffer is stored as counts. Refer to Appendix B for information on converting a count value to voltage.

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

See Also

K_IntFree, K_SetBuf

K_IntAlloc (cont.)

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
void far *pBuf;   // Pointer to allocated buffer
WORD phMem;   // Memory Handle to buffer
...
wDasErr = K_IntAlloc (hAD, 131072, &pBuf, &phMem);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf As Long
Global phMem As Integer
...
wDasErr = K_IntAlloc (hAD, 131072, pBuf, phMem)
```

Purpose

Frees a buffer allocated for an analog input operation.

Prototype

C/C++

DASErr far pascal K_IntFree (WORD *phMem*);

Visual Basic for Windows

Declare Function K_IntFree Lib "DASSHELL.DLL"
(ByVal *phMem* As Integer) As Integer

Parameters

phMem Handle to buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the buffer specified by *phMem*; 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 (phMem);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_IntFree (phMem)
```

K_IntStart

Purpose

Starts an analog input operation.

Prototype

C/C++

DASErr far pascal K_IntStart (DWORD *hFrame*);

Visual Basic for Windows

Declare Function K_IntStart Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function starts the operation defined by *hFrame*. Refer to page 1-4 for a summary of the programming tasks associated with analog input operations.

See Also

K_IntStatus, K_IntStop

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_IntStart (hAD)
```

Purpose

Gets the status of an analog input operation.

Prototype

C/C++

DASErr far pascal K_IntStatus (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Visual Basic for Windows

Declare Function K_IntStatus Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long)
As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pStatus	Status of operation; see **Remarks** below for value stored.
pCount	Current number of samples transferred into the buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function stores the status in *pStatus* and the current number of samples transferred into the buffer in *pCount*.

The value stored in *pStatus* depends on the settings in the Status word, as shown below:

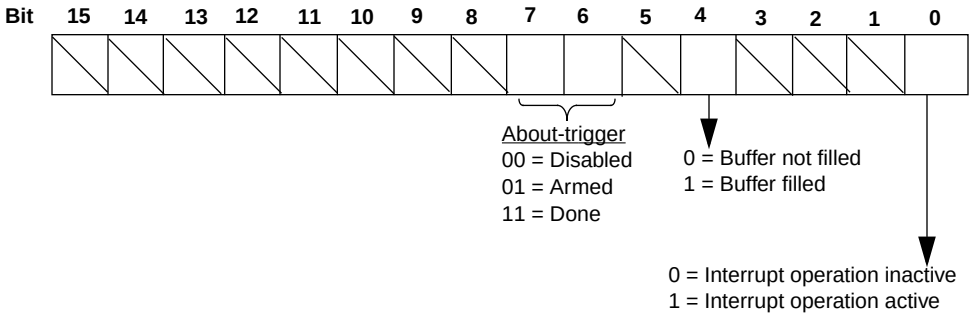


Figure 4-1. Status Word Settings

The bits are described as follows:

- Bit 0: This bit indicates whether an analog input operation is in progress.
- Bits 1 to 3: Not used.
- Bit 4: This bit is set when the 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

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 or DASDEC32.BAS to your project)

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

K_IntStop

Purpose

Stops an analog input operation.

Prototype

C/C++

DASErr far pascal K_IntStop (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Visual Basic for Windows

Declare Function K_IntStop Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pStatus	Status of operation.
pCount	Current number of samples transferred into the buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function stops the board from acquiring data, disables the operation, and returns the status of the operation at the point when your program called this function. No data is transferred into the buffer in computer memory.

Refer to page 4-25 for more information on the status word returned.

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++

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

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.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

pDest	Address of destination array.
pSource	Address of source buffer.
nCount	Number of samples to transfer. Value values: **1** to **32767**

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 or DASDEC32.BAS to your project)

```
...
Dim ADArray(2000) As Integer
...
wDasErr = K_IntAlloc (hAD, 131072, pBuf, hMem)
...
wDasErr = K_MoveBufToArray (ADArray(0), pBuf, 1000)
```

Purpose

Initializes any Keithley DAS Function Call Driver.

Prototype

C/C++

DASErr far pascal K_OpenDriver (char far * *szDrvName*, char far * *szCfgName*, DWORD far * *phDrv*);

Visual Basic for Windows

Declare Function K_OpenDriver Lib "DASSHELL.DLL" (ByVal *szDrvName* As String, ByVal *szCfgName* As String, *phDrv* As Long) As Integer

Parameters

szDrvName	Board name. Valid value:	**"DAS4200"** (for DAS-4200 Series boards)
szCfgName	Driver configuration file. Valid values:	The name of a configuration file; **0** if driver has already been opened
phDrv	Handle associated with the driver.	

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function initializes the DAS-4200 Series Function Call Driver according to the information in the configuration file specified by *szCfgName*, and stores the driver handle in *phDrv*.

You can use this function to initialize the Function Call Driver associated with any Keithley MetraByte DAS board. For DAS-4200 Series boards, the string stored in *szDrvName* must be DAS4200. Refer to other Function Call Driver user's guides for the appropriate string to store in *szDrvName* for other Keithley MetraByte DAS boards.

The value stored in *phDrv* 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 *phDrv.*

You create a configuration file using the CFG4200.EXE utility. Refer to your *DAS-4200 Series User's Guide* for more information.

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 phDrv;
...
wDasErr = K_OpenDriver ("DAS4200", "DAS4200.CFG", &phDrv);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
DIM phDrv As Long
...
wDasErr = K_OpenDriver("DAS4200", "DAS4200.CFG", phDrv)
```

Purpose

Enables the about trigger and specifies the number of post-trigger samples.

Prototype

C/C++

DASErr far pascal K_SetAboutTrig (DWORD *hFrame*,
DWORD *dwSamples*);

Visual Basic for Windows

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

Parameters

hFrame	Handle to the frame that defines the operation.
dwSamples	Number of post-trigger samples. Valid values: **1** to **131072** for the DAS-4201/128K board **1** to **524288** for the DAS-4201/512K board

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 *dwSamples*.

Note that you cannot use an about trigger with an external start trigger. If you enable the about trigger and specify an external trigger source in **K_SetTrig**, the software assumes that the external trigger is the about trigger. For pre-trigger and about-trigger acquisition, the start trigger is always an internal trigger.

See Also

K_ClrAboutTrig

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_SetAboutTrig (hAD, 100);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
wDasErr = K_SetAboutTrig (hAD, 100)
```

Purpose

Sets up an external analog start or about trigger.

Prototype

C/C++

DASErr far pascal K_SetADTrig (DWORD *hFrame*, short *nOpt*, short *nChan*, DWORD *dwLevel*);

Visual Basic for Windows

Declare Function K_SetADTrig Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nOpt* As Integer, ByVal *nChan* As Integer, ByVal *dwLevel* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nOpt	Analog trigger polarity and sensitivity. Valid values: **0** for Positive edge **1** for Positive level **2** for Negative edge **3** for Negative level
nChan	Analog input channel. Valid values: **0**, **1**
dwLevel	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 *hFrame*, this function specifies the channel used for an analog trigger in *nChan*, the level used for the analog trigger in *dwLevel*, and the trigger polarity and trigger sensitivity in *nOpt*.

The analog input channel you specify in *nChan* must be the same as the analog input channel that is sampled; otherwise, the driver returns an error.

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 or DASDEC32.BAS to your project)

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

Purpose

Specifies the starting address of a previously allocated buffer and the number of samples in the 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

hFrame	Handle to the frame that defines the operation.
pBuf	Starting address of buffer.
dwSamples	Number of samples. Valid values: **1** to **131072** for the DAS-4201/128K board **1** to **524288** for the DAS-4201/512K board

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 buffer in *pBuf* and the number of samples (the size of the buffer) in *dwSamples*.

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

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

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, 131072, &pBuf, &hMem);
wDasErr = K_SetBuf (hAD, pBuf, 131072);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf As Long
...
wDasErr = K_IntAlloc (hAD, 131072, pBuf, hMem)
wDasErr = K_SetBuf (hAD, pBuf, 131072)
```

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 *hFrame* As Long, *pBuf* As Integer, ByVal *dwSize* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pBuf	Starting address of the locally dimensioned integer array.
dwSize	Number of samples. Valid values: **1** to **32768**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the starting address of a locally dimensioned integer array in *pBuf* and the number of samples stored in the array in *dwSize*.

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 *pBuf* variable sets the value of the Buffer element; the *dwSize* 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 or DASDEC32.BAS to your project)

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

Purpose

Specifies a single channel.

Prototype

C/C++

DASErr far pascal K_SetChn (DWORD *hFrame*, short *nChan*);

Visual Basic for Windows

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

Parameters

hFrame	Handle to the frame that defines the operation.
nChan	Channel on which to perform an analog input operation. Valid values: **0**, **1**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the single channel used in *nChan*.

Software channel 0 corresponds to Channel A on the board; software channel 1 corresponds to Channel B on the board.

The value you specify in *nChan* sets the Channel element in the frame identified by *hFrame*.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

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

K_SetClk

Purpose

Specifies the pacer clock source.

Prototype

C/C++

DASErr far pascal K_SetClk (DWORD *hFrame*, short *nMode*);

Visual Basic for Windows

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

Parameters

hFrame	Handle to the frame that defines the operation.
nMode	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 *hFrame*, this function specifies the pacer clock source in *nMode*. The value you specify in *nMode* sets the Clock Source element in the frame identified by *hFrame*.

K_GetADFrame and **K_ClearFrame** specify internal as the default clock source.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

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

Purpose

Specifies the clock divider 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

hFrame — Handle to the frame that defines the operation.

dwDivisor — Clock divider.
Valid values: See the table below.

The following table shows the valid clock divider values and the corresponding conversion rates and sample periods for the internal pacer clock:

Clock Divider	Conversion Rate	Sample Period
32	100 Msamples/s	10 ns
64	50 Msamples/s	20 ns
128	25 Msamples/s	40 ns
256	12.5 Msamples/s	80 ns
512	6.25 Msamples/s	160 ns
1024	3.13 Msamples/s	320 ns
2048	1.56 Msamples/s	640 ns
4096	0.78 Msamples/s	1280 ns

Return Value

Error/status code. Refer to Appendix A.

K_SetClkRate (cont.)

Remarks

For the operation defined by *hFrame*, this function specifies the clock divider for the internal pacer clock in *dwDivisor.*

The value you specify in *dwDivisor* sets the Pacer Clock Rate element in the frame identified by *hFrame*.

If you enter a clock divider value that is not one of those specified as a valid value above, the driver uses the next fastest rate. For example, if you enter a clock divider value of 63, the driver uses a clock divider value of 32 to perform the faster conversion rate. To determine the actual clock divider used, use **K_GetClkRate**.

Refer to page 2-11 for more information on the internal pacer clock.

See Also

K_GetClkRate

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
wDasErr = K_SetClkRate (hAD, 64);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetClkRate (hAD, 64)
```

Purpose

Sets up an external digital start or about trigger.

Prototype

C/C++

DASErr far pascal K_SetDITrig (DWORD *hFrame*, short *nOpt*, short *nChan*, DWORD *nPattern*);

Visual Basic for Windows

Declare Function K_SetDITrig Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nOpt* As Integer, ByVal *nChan* As Integer, ByVal *nPattern* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nOpt	Trigger polarity and sensitivity. Valid values: **0** for Positive edge **2** for Negative edge
nChan	Digital input channel. Valid value: **0**
nPattern	Trigger pattern. Valid value: **0**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame,* this function specifies the trigger polarity and sensitivity in *nOpt*.

Since an external digital trigger is always connected to the Trigger I/O connector on the board, the value of *nChan* is meaningless. In addition, the DAS-4200 Series Function Call Driver does not support digital pattern triggering; therefore, the value of *nPattern* is meaningless. The *nChan* and *nPattern* parameters are provided for future compatibility.

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 or DASDEC32.BAS to your project)

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

Purpose

Sets the gain.

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 **7**, described as follows:

Analog Input Range	Gain	Gain Code
±2 V	1	0
±1 V	2	1
±500 mV	4	2
±250 mV	8	3
±125 mV	16	4
±62.5 mV	32	5
±31.25 mV	64	6
±15.625 mV	128	7

Return Value

Error/status code. Refer to Appendix A.

K_SetG (cont.)

Remarks

For the operation defined by *hFrame*, this function specifies the gain code for a single channel in *nGain*.

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.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

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

Purpose

Specifies the start trigger source.

Prototype

C/C++

DASErr far pascal K_SetTrig (DWORD *hFrame*, short *nMode*);

Visual Basic for Windows

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

Parameters

hFrame	Handle to the frame that defines the operation.
nMode	Trigger source. Valid values: **0** for Internal start trigger **1** for External start trigger

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the trigger source in *nMode*.

An internal trigger is a software trigger; conversions begin when your program calls **K_IntStart**. An external trigger is either an analog trigger or a digital trigger.

You cannot use an external start trigger when the about trigger is enabled. If you specify an external trigger and the about trigger is enabled (using **K_SetAboutTrig**), the software returns an error.

If *nMode* = **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.

K_GetADFrame and **K_ClearFrame** set the trigger source to internal.

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 or DASDEC32.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-4200 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 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 was not installed	Refer to the user's guide for information on how to install and set up the Keithley Memory Manager.
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 the Keithley Memory Manager.
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 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 the Windows 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. You attempted to specify the starting address of a locally dimensioned array in Windows 95.	Close all programs and restart Windows. In Windows 95, make sure that you use **K_IntAlloc** or **K_DMAAlloc** to dynamically allocate a memory buffer and make sure that you use **K_SetBuf** or **K_BufListAdd** to specify the starting address of the dynamically allocated memory buffer. If the error continues, contact Keithley MetraByte for technical support.
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.
7001	28673	**Bad AD Channel Mode**: The only input range type supported by the board is bipolar.	Specify bipolar in the configuration file.
7002	28674	**Bad board numbe**r: The driver initialization function found an illegal board number in the specified configuration file.	Specify a legal board number: 0 to 1
7003	28675	**Bad 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	**Bad 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	**Bad 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

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

Error Code		Cause	Solution
Hex	Decimal		
7006	28678	**Bad bus transfer width**: The driver initialization function found an illegal bus transfer width in the configuration file.	Specify a legal width type: 16, 8
7007	28679	**Bad zero wait-state**: The driver initialization function found an illegal input in the specified configuration file.	Specify enabled or disabled.
7008	28680	**Bad unipolar filter**: The driver initialization function found an illegal unipolar filter level in the specified configuration file.	Specify a legal unipolar filter value: 0 to 7
7009	28681	**Invalid analog trigger channel**: The analog input channel specified in **K_SetADTrig** does not match the analog input channel being sampled (specified in **K_SetChn**).	Make sure that the analog trigger channel is the same as the analog input channel that is sampled.
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 coupling**: The driver initialization found an illegal coupling value in the configuration file.	Specify either AC or DC in the configuration file.
700C	28684	**Illegal number of samples**: The driver detected a request to acquire more samples than the DAS-4200 Series board could hold.	Specify a number of samples within the size of the buffer (131,072 for 128K buffers, 544,288 for 512K buffers)
700E	28686	**Error - Resource busy**: The 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.

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

Error Code		Cause	Solution
Hex	Decimal		
700F	28687	**Error - Start and stop channels are not equal**: The start channel parameter is not equal to the stop channel parameter.	Ensure that the start channel parameter is equal to the stop channel parameter.
7015	28693	**Error - Illegal number of about-trigger samples**: The number of about-trigger samples is greater than the acquisition buffer size.	Reduce the number of about-trigger samples to the size of the buffer or increase the buffer size.
7801	30721	**No board at base address:** The DAS-4200 Series board was not found at the base I/O address specified.	Run CFG4200 and check the base I/O address settings. Make sure that the settings in the configuration file match the settings of the jumpers on the board.
7802	30722	**Windows cannot find 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: `DEVICE=C:\DOS\EMM386.EXE X=CC00-CFFF` (Note this should be typed on one line.)
7804	30724	**Warning old board revision:** The board you are using is an older revision. Calibrations will not be accurate.	Contact the Keithley MetraByte Applications Engineering Department.
7806	30726	**Warning EEPROM incorrectly set**: The EEPROM settings are not giving a consistent value.	Run D4200 to set up the EEPROM.

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

Error Code		Cause	Solution
Hex	Decimal		
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.
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.

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

Error Code		Cause	Solution
Hex	Decimal		
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 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.
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.

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

Error Code		Cause	Solution
Hex	**Decimal**		
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-4200 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	Input Range	Span (V)
1	–2 to 2 V	4
2	–1 to 1 V	2
4	–500 to 500 mV	1
8	–250 to 250 mV	0.50
16	–125 to 125 mV	0.25
32	–62.5 to 62.5 mV	0.125
64	–31.25 to 31.25 mV	.0625
128	–15.625 to 15.625 mV	0.03125

For example, assume that you want to read analog input data from a channel on the DAS-4200 Series board configured for the ±1 V input range. The channel collects the data at a gain of 2; the count value is 72. The voltage is determined as follows:

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

As another example, assume that you want to read analog input data from a channel on a DAS-4200 Series board configured for the ±500 mV input range. The channel collects the data at a gain of 4; the count value is -112. The voltage is determined as follows:

$$\frac{-112 \times 1\ \text{V}}{256} = (-0.4375)$$

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:

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

For example, assume that you want to specify an analog trigger level of 100 mV for a channel on the DAS-4200 Series board configured for a input range of ±125 mV. The count value is determined as follows:

$$\frac{0.1\ \text{V} \times 256}{0.25\ \text{V}} = 102.4$$

Index

A

about-trigger acquisition 2-17
allocating memory buffers 2-8
  C/C++ 3-3
  Microsoft Visual Basic for Windows 3-10
analog input
  programming flow diagrams 1-6
analog input operations
  channels 2-10
  converting analog input values to voltages B-1
  input ranges 2-10
  memory allocation 2-7
  operation modes 2-4
  pacer clocks 2-11
  triggers 2-12
analog trigger 2-13, B-3
assigning the starting address of a memory location 2-9

B

board initialization 2-2
Borland C/C++
  compile and link statements for DOS 3-8
  creating an executable file for DOS 3-8
  dynamically allocating a memory buffer 3-3
  files required for DOS 3-8
  files required for Windows 3-9
  handling errors 3-5
buffer address 2-9
buffer address function 1-3, 4-2

C

C/C++: *see* Borland C/C++, Microsoft C/C++
channel and gain functions 1-4, 4-2
channels 2-10
clock functions 1-4, 4-2
clock sources: *see* pacer clocks
compile and link statements
  Borland C/C++ (for DOS) 3-8
  Microsoft C/C++ (for DOS) 3-6
conventions 4-3
converting
  counts to voltages B-1
  voltages to counts B-3
creating an executable file
  Borland C/C++ (for DOS) 3-8
  Microsoft C/C++ (for DOS) 3-6
  Visual Basic for Windows 3-15

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 2-6
device handle 2-2
digital trigger 2-14
driver handle 2-2
driver initialization 2-2
dynamically allocating a memory buffer 2-8
  C/C++ 3-3
  Visual Basic for Windows 3-10

E

elements of frame 2-6
error codes A-1

error handling 2-3
Borland C/C++ 3-5
Microsoft C/C++ 3-5
Visual Basic for Windows 3-14
external pacer clock 2-12

F

files required
Borland C/C++ (for DOS) 3-8
Borland C/C++ (for Windows) 3-9
Microsoft C/C++ (for DOS) 3-6
Microsoft C/C++ (for Windows) 3-7
flow diagrams 1-4
frame elements 2-6
frame handle 2-5
frame management functions 1-3, 4-2
frame types 2-5
functions
buffer address 1-3, 4-2
channel and gain 1-4, 4-2
clock 1-4, 4-2
frame management 1-3, 4-2
initialization 1-3, 4-2
K_ClearFrame 2-6, 4-5
K_CloseDriver 2-2, 4-6
K_ClrAboutTrig 4-7
K_DASDevInit 2-3, 4-8
K_FreeDevHandle 2-3, 4-9
K_FreeFrame 2-6, 4-10
K_GetADFrame 2-5, 4-11
K_GetClkRate 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 2-8, 4-21
K_IntFree 2-8, 4-23
K_IntStart 2-4, 4-24
K_IntStatus 2-4, 4-25
K_IntStop 2-4, 4-28
K_MoveBufToArray 4-30
K_OpenDriver 2-2, 4-31
K_SetAboutTrig 4-33
K_SetADTrig 2-14, 4-35
K_SetBuf 2-9, 4-37
K_SetBufI 4-39
K_SetChn 4-41
K_SetClk 2-12, 4-42
K_SetClkRate 4-43
K_SetDITrig 2-14, 4-45
K_SetG 4-47
K_SetTrig 2-13, 2-14, 4-49
memory management 1-3, 4-2
miscellaneous 1-4, 4-3
operation 1-3, 4-2
summary 1-3
trigger 1-4, 4-2

H

handle
device 2-2
driver 2-2
frame 2-5
memory 2-8
help 1-10

I

initialization functions 1-3, 4-2
initializing a board 2-2
initializing the driver 2-2
internal pacer clock 2-11
internal trigger 2-13
interrupt-mode operations 2-4

K

K_ClearFrame 2-6, 4-5
K_CloseDriver 2-2, 4-6
K_ClrAboutTrig 4-7
K_DASDevInit 2-3, 4-8
K_FreeDevHandle 2-3, 4-9
K_FreeFrame 2-6, 4-10
K_GetADFrame 2-5, 4-11
K_GetClkRate 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 2-8, 4-21
K_IntFree 2-8, 4-23
K_IntStart 2-4, 4-24
K_IntStatus 2-4, 4-25
K_IntStop 2-4, 4-28
K_MoveBufToArray 4-30
K_OpenDriver 2-2, 4-31
K_SetAboutTrig 4-33
K_SetADTrig 2-14, 4-35
K_SetBuf 2-9, 4-37
K_SetBufI 4-39
K_SetChn 4-41
K_SetClk 2-12, 4-42
K_SetClkRate 4-43
K_SetDITrig 2-14, 4-45
K_SetG 4-47
K_SetTrig 2-13, 2-14, 4-49

M

maintenance operations: *see* system operations
memory allocation 2-7
  C/C++ 3-3
  Visual Basic for Windows 3-10
memory handle 2-8
memory management functions 1-3, 4-2
Microsoft C/C++
  allocating a memory buffer 3-3
  compile and link statements for DOS 3-6
  creating an executable file for DOS 3-6
  files required for DOS 3-6
  files required for Windows 3-7
  handling errors 3-5
Microsoft Visual Basic for Windows
  see Visual Basic for Windows
miscellaneous functions 1-4, 4-3
miscellaneous operations: *see* system operations

O

operation functions 1-3, 4-2
operations
  analog input 2-4
  system 2-2

P

pacer clocks 2-11
post-trigger acquisition 2-15
preliminary procedures 1-5
pre-trigger acquisition 2-16
procedures 1-4
  analog input 1-6
  preliminary 1-5
programming flow diagrams 1-4
programming information
  C/C++ 3-2
  Visual Basic for Windows 3-10
programming overview 3-2

R

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 address: *see* buffer address
starting an operation 2-4
status 2-4
status codes 2-3
stopping an operation 2-3, 2-4
summary of functions 1-3
system operations 2-2

T

tasks 1-4
    analog input 1-6
    preliminary 1-5
technical support 1-10
trigger functions 1-4, 4-2
trigger level, specifying an analog trigger
        B-3
triggers 2-12
troubleshooting 1-10

V

Visual Basic for Windows
    allocating a memory buffer 3-10
    creating an executable file 3-15
    handling errors 3-14