DAS-4100 Series
Function Call Driver

USER'S GUIDE

DAS-4100 Series
Function Call Driver
User's Guide

Revision A - October 1995
Part Number: 94550

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

Keithley MetraByte Division

Keithley Instruments, Inc.

440 Myles Standish Blvd. Taunton, MA 02780

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

Preface

This manual describes how to write application programs for DAS-4100 Series boards using the DAS-4100 Series Function Call Driver. The DAS-4100 Series Function Call Driver supports the following DOS-based languages:

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

The DAS-4100 Series Function Call Driver supports the following Windows-based languages:

- Microsoft C/C++ (Version 7.0 and higher)
- Borland C/C++ (Version 4.0 and higher)
- Microsoft Visual Basic® for Windows (Version 3.0 and higher)
- Microsoft Visual C++™ (Version 1.0 and higher)

The manual is intended for application programmers using a DAS-4100 Series boards in an IBM® PC AT® or compatible computer. It is assumed that users have read the *DAS-4100 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-4100 Series Function Call Driver User's Guide* is organized as follows:

- Chapter 1 contains the information needed to install the DAS-4100 Series Function Call Driver and to get help.
- Chapter 2 contains the background information needed to use the functions included in the DAS-4100 Series Function Call Driver.
- Chapter 3 contains programming guidelines and language-specific information related to using the DAS-4100 Series Function Call Driver.
- Chapter 4 contains detailed descriptions of the DAS-4100 Series Function Call Driver functions, arranged in alphabetical order.
- Appendix A contains a list of the error codes returned by DAS-4100 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.

Table of Contents

List of Figures

List of Tables

1

Getting Started

The DAS-4100 Series Function Call Driver is a library of data acquisition and control functions (referred to as the Function Call Driver or FCD functions). It is part of the following two software packages:

- **DAS-4100 Series Standard Software package** - This is the software package that is shipped with DAS-4100 Series boards; it includes utility programs, running under DOS, that allow you to configure, calibrate, and test the DAS-4100 Series boards.
- **ASO-4100 software package** - This is the Advanced Software Option for DAS-4100 Series boards. You purchase the ASO-4100 software package separately from the board; it includes the following:
  - Libraries of FCD functions for Microsoft C/C++ (for DOS) and Borland C/C++ (for DOS).
  - Dynamic Link Libraries (DLLs) of FCD functions for Microsoft Visual Basic for Windows, Microsoft C/C++ (for Windows), and Borland C/C++ (for Windows).
  - Support files, containing program elements, such as function prototypes and definitions of variable types, that are required by the FCD functions.
  - Utility programs, running under DOS, that allow you to configure, calibrate, and test the DAS-4100 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-4100 Series User's Guide*.

If you need help installing or using the DAS-4100 Series Function Call Driver, call your local sales office or the Keithley MetraByte Applications Engineering Department at:

(508) 880-3000

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

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

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

DAS-4100 Series board configuration	Serial #	_____
	Revision code	_____
	Base I/O address setting	_____
	Memory address setting	_____
	Interrupt level setting	_____
Computer	Manufacturer	_____
	CPU type	_____
	Clock speed (MHz)	_____
	KB of RAM	_____
	Video system	_____
	BIOS type	_____
	Memory manager	_____
Operating system	DOS version	_____
	Windows version	_____
	Windows mode	_____
Software package	Name	_____
	Serial #	_____
	Version	_____
	Invoice/Order #	_____

Compiler (if applicable)	Language	_____
	Manufacturer	_____
	Version	_____
		___
Accessories	Type	_____

2

Available Operations

This chapter contains the background information you need to use the FCD functions to perform operations on DAS-4100 Series boards. The supported operations are listed in Table 2-1.

Table 2-1. Supported Operations

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

System Operations

This section describes the miscellaneous operations and general maintenance operations that apply to DAS-4100 Series boards and to the DAS-4100 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-4100 Series Function Call Driver and any other Keithley DAS Function Call Drivers you are using in your application program. To initialize the drivers, use the **K_OpenDriver** function. You specify the driver you are using and the configuration file that defines the use of the driver. The driver returns a unique identifier for the driver; this identifier is called the driver handle.

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

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

Initializing a Board

The DAS-4100 Series Function Call Driver supports up to two boards. You must use the **K_GetDevHandle** function to specify the boards you want to use. The driver returns a unique identifier for each board; this identifier is called the device handle.

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

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

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

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

Retrieving Revision Levels

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

The **K_GetVer** function allows you to get both the revision number of the DAS-4100 Series Function Call Driver and the revision number of the Keithley DAS Driver Specification to which the driver conforms.

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

Handling Errors

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

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

C/C++	page 3-11
Visual Basic for Windows	page 3-19

For C-language application programs only, the DAS-4100 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-4100 Series boards support interrupt mode only. In interrupt mode, the board acquires multiple samples from the analog input channels. A hardware clock initiates A/D conversions. Once the analog input operation begins, control returns to your application program. The hardware continues to store the acquired data in its onboard memory until the specified number of samples is acquired, then transfers the data all at once to a user-defined array or buffer in the computer using an interrupt service routine.

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

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

Memory Allocation and Management

Interrupt-mode analog input operations require a memory location in which to store acquired data. Table 2-2 lists ranges for the DAS-4100 Series boards.

Table 2-2. Sample Ranges

Board	Sample Range
DAS-4101/64K	1 to 64K
DAS-4101/256K	1 to 256K
DAS-4101/1M	1 to 1024K
DAS-4101/2M	1 to 2048K
DAS-4102/64K	1 to 128K
DAS-4102/256K	1 to 512K
DAS-4102/1M	1 to 2048K

Note: Even though you can reserve a memory location greater than the board requires, to conserve memory it is recommended that you allocate only the required amount.

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

Dimensioning a Local Array

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

Since the DAS-4100 Series Function Call Driver stores data in 16-bit integers, you must dimension a local array as an integer data type.

Dynamically Allocating a Memory Buffer

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

- The size of the buffer is limited by the amount of free physical memory available in your computer at run time.
- A dynamically allocated memory buffer can be freed to make it available to other programs or processes.
- The limitation of this method is that, for Visual Basic for Windows, data in a dynamically allocated buffer is not directly accessible to your program. You must use the **K_MoveBufToArray** function to move the data from the dynamically allocated buffer to the program's local array; refer to page 4-30 for more information.

Use the **K_IntAlloc** function to dynamically allocate a memory buffer. You specify the operation requiring the buffer and the number of samples to store in the buffer, refer to Table 2-2. 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.

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

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

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

Assigning the Starting Address

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

Language	Memory Location	Function	See...
C/C++	Array or Buffer	K_SetBuf	page 4-38
Visual Basic for Windows	Array	K_SetBufI	page 4-40
	Buffer	K_SetBuf	page 4-38

Channels

The DAS-4100 Series board provides up to two analog input channels; the software refers to the analog input signal from the Channel A connector as channel 0 and the analog input signal from the Channel B connector as channel 1. You can perform an analog input operation on a single channel at a time or both channels.

Acquiring Samples from a Single Channel

You can acquire a single sample or multiple samples from a single analog input channel. For DAS-4102 boards use the **K_SetChn** function to specify the channel. For DAS-4101 boards you can only acquire off channel 0. Use **K_SetG** function to specify the gain code. Refer to the next section for the various input ranges available.

Acquiring Samples from Both Channels

With DAS-4102 boards, you can acquire samples from both channels at once. Use **K_SetStartStopChn** to specify channel 0 as the start channel and channel 1 as the stop channel. The samples are gathered simultaneously from both channels, but channel 0 is placed in the buffer first before channel 1. Use **K_SetG** function to specify the gain code for both channels. Refer to Table 2-3 on page 2-9 for the various input ranges available.

Acquiring Samples Using a Channel-Gain Queue

With DAS-4102 boards, you can acquire samples from both channels with different gain settings, using a channel-gain queue. In the channel-gain queue, you specify the gain for channel 0 and channel 1.

Channel 0 must be the first in the queue and channel 1 must be the other channel in the list. The samples will be copied to the buffer in this order, although they are acquired simultaneously.

The way that you specify the channel and gains in a channel-gain queue depends on the language you are using. Refer to Chapter 4 for information on programming the various drivers.

After you create the channel-gain queue in your program, use the **K_SetChnGAry** function to transfer the contents of the channel-gain queue to the driver/board.

Input Ranges

Each channel on the DAS-4100 Series board can measure signals in one of 16, software-selectable bipolar analog input ranges.

Table 2-3 lists the analog input ranges supported by DAS-4100 Series boards and the gain code associated with each range. Use the **K_SetG** function to specify the gain code. For the ±125 mV, ±250 mV, and ±0.5 V analog input ranges, the choice of gain code affects the bandwidth; refer to Appendix C of the *DAS-4100 Series User's Guide* for more information.

Table 2-3. Analog Input Ranges

Analog Input Range	Gain Code	Analog Input Range	Gain Code
±100 mV	12	±800 mV	15
±125 mV	8	±1 V	5, 11
±250 mV	4	±1.25 V	6
±400 mV	13	±2 V	1, 7
±500 mV	0, 9, 14	±2.5 V	2
±625 mV	10	±4.0 V	3

Pacer Clocks

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

The internal and external pacer clocks are described in the following subsections; refer to the *DAS-4100 Series User's Guide* for more information.

Internal Pacer Clock

When you start an analog input operation (using **K_IntStart**), conversions are performed at a rate of 16.384 Gsamples/s divided by a count value of 256, 512, 1024, 2048, 4096, 8192, 16384 or 32768. Use the **K_SetClkRate** function to specify the count value. Each count represents .0610351 ns between conversions.

Table 2-4 lists the conversion rates, sample periods, and count values for the internal pacer clock.

Table 2-4. Available Conversion Rates Using Internal Clock

Conversion Rate	Sample Period	Count Value
64 Msamples/s	15.625 ns	256
32 Msamples/s	31.25 ns	512
16 Msamples/s	62.5 ns	1024
8 Msamples/s	125 ns	2048
4 Msamples/s	250 ns	4096
2 Msamples/s	500 ns	8192
1 Msamples/s	1 µs	16384
500 Ksamples/s	2 µs	32768

Note: If you enter a count value that is not one of those listed in Table 2-4, the driver uses the next fastest rate. For example, if you enter a count value of 550, the driver uses a count value of 512 to perform the faster conversion rate. To determine the actual count value used, use the **K_GetClkRate** function.

External Pacer Clock

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

Note: Do not use a conversion rate greater than 64 Msamples/s when using an external pacer clock or sample accuracy will not be guaranteed.

The conversion rate divisor, as set by **K_SetClkRate**, still has an effect, with a conversion rate divisor of 256 producing the base rate specified by the input clock frequency. Conversion rate divisors must go by powers of two (for instance, 128, 256, 512, 1024) with lower divisors producing higher conversion rates and higher divisors producing lower conversion rates. Each doubling of the divisor has the net effect of halving the effective digitization rate. Conversely, halving the divisor doubles the effective digitization rate.

Triggers

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

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

When the trigger event occurs, a TTL-level signal is output on the Trigger I/O connector. The signal is edge-sensitive with a positive polarity.

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

Trigger Sources

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

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

Internal Trigger

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

External Analog Trigger

You can select the digitized analog input signal from the Channel A connector (referred to in software as analog trigger channel 0) and/or (with the DAS-4102 board) from the Channel B connector (referred to in software as analog trigger channel 1). You program the trigger level as a count value in 256 steps (–128 to +127) from –4 V to +3.96 V.

You can also select the ±16 V trigger input signal from the Analog Trigger In connector (referred to in software as analog trigger channel 2) as the trigger signal. You program the trigger level as a count value in 256 steps (–128 to +127) from –16 V to +15.875 V. If you use analog trigger channel 2, you can acquire analog input data from either the Channel A connector or from the Channel B connector or both.

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

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

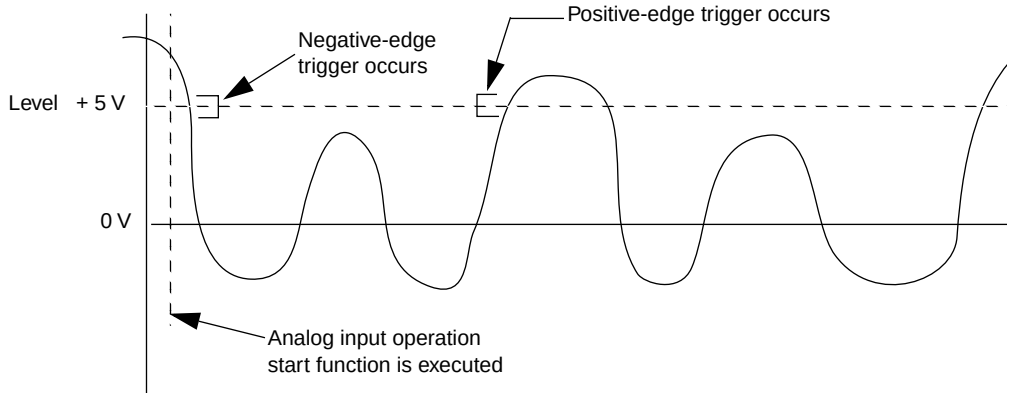


Figure 2-1. Analog Trigger Conditions

Use the **K_SetADTrig** function to specify the analog input channel to use as the trigger channel, the trigger level, and the trigger polarity. The trigger sensitivity is always edge for the DAS-4100 Series board.

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

External Digital Trigger

The digital trigger signal is connected to the Trigger I/O connector of the DAS-4100 Series board. Use **K_SetDITrig** to specify the digital input channel to use as the trigger channel and whether you want the trigger event to occur on the rising edge of the digital trigger signal (positive-edge trigger) or on a falling edge of the digital trigger signal (negative-edge trigger). The trigger sensitivity is always edge for DAS-4100 Series boards. The trigger events are illustrated in Figure 2-2.

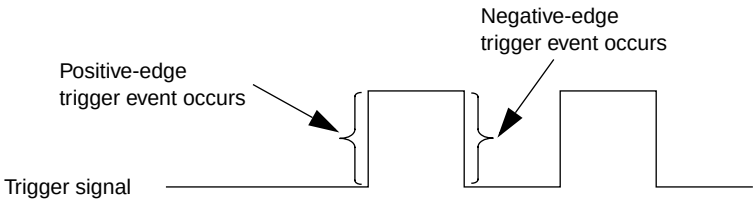


Figure 2-2. Digital Trigger Conditions

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. If desired, you can specify the number of samples to wait between when the trigger event occurs and when the data is collected by using the **K_SetPostTrigDelay** function.

The post-trigger delay can range from 0 to 8,388,608 samples.

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

1. Specify the start trigger.
   - Use **K_SetTrig** to specify an internal or an external trigger source (specify external for an analog or digital trigger).
   - If you specify an external start trigger in **K_SetTrig**, define the start trigger conditions using **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger).

2. If you specified an external analog or digital start trigger, use **K_ClrAboutTrig** to disable the about trigger.
3. To specify the number of post-trigger samples to wait after the trigger event occurs and before data is collected, use **K_SetPostTrigDelay**.

Pre-Trigger Acquisition

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

Note: The implied number of pre-trigger samples must be acquired before the board can accept a trigger event. A trigger that comes before all implied pre-trigger samples are acquired is ignored.

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 application program calls the **K_IntStart** function. The about trigger can be either an external analog or external digital trigger; the operation stops after a specified number of samples has been acquired after the about-trigger event occurs.

Note: The implied number of pre-trigger samples must be acquired before the board can accept a trigger event. A trigger that comes before all implied pre-trigger samples are acquired is ignored.

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.

3

Programming with the Function Call Driver

This chapter contains an overview of the structure of the DAS-4100 Series Function Call Driver, as well as programming guidelines and language-specific information to assist you when writing application programs with the DAS-4100 Series Function Call Driver.

How the Driver Works

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

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

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

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

You indicate that you want to perform an analog input operation by getting an available frame for the driver. The DAS-4100 Series Function Call Driver provides analog input frames, called A/D (analog-to-digital) frames. You use the **K_GetADFrame** function to access an available A/D frame. The driver returns a unique identifier for the frame; this name is called the frame handle.

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

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

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

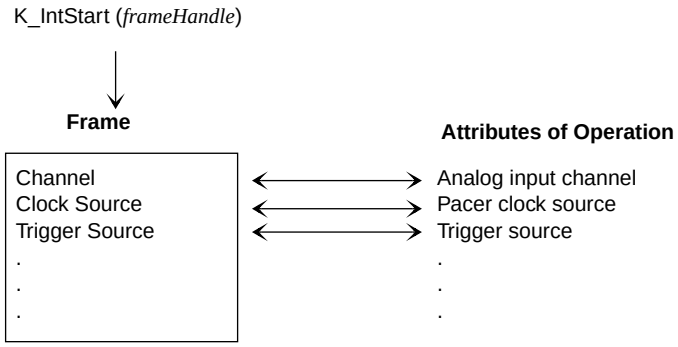


Figure 3-1. Interrupt-Mode Operation

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

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

Table 3-1 lists the elements of a DAS-4100 Series A/D frame. This table also lists the default value of each element and the setup function used to define each element.

Table 3-1. A/D Frame Elements

Element	Default Value	Setup Function
Buffer[1]	0 (NULL)	K_SetBuf
Number of Samples	0	K_SetBuf
Start and Stop Channel	0	K_SetChn K_SetStartStopChn
Channel-Gain Queue	0(NULL)	K_SetChnGAry
Input range	0 (±200 mV)	K_SetG
Clock Source	Internal A/D pacer clock	K_SetClk
Pacer Clock Rate	0	K_SetClkRate
Trigger Source	Internal	K_SetTrig
Trigger Type	Digital	K_SetADTrig K_SetDITrig
Trigger Channel	0 (for analog trigger)	K_SetADTrig
	0 (for digital trigger)	Not applicable[2]
Trigger Polarity	Positive	K_SetADTrig K_SetDITrig

Table 3-1. A/D Frame Elements (cont.)

Element	Default Value	Setup Function
Trigger Sensitivity	Edge (for analog)	Not applicable[2]
	Edge (for digital)	Not applicable[2]
Trigger Level	0	K_SetADTrig
About-Trigger Acquisition	Disabled	K_SetAboutTrig K_ClrAboutTrig[3]
Post-Trigger Delay	0	K_SetPostTrigDelay

Notes
[1] This element must be set.
[2] The default value of this element cannot be changed.
[3] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

Note: The DAS-4100 Series Function Call Driver provides many other functions that are not related to controlling frames or defining the elements of frames. These functions include initialization functions, memory management functions, and miscellaneous functions.

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

Programming Overview

To write an application program using the DAS-4100 Series Function Call Driver, perform the following steps:

1. Define the application's requirements. Refer to Chapter 2 for a description of the board operations supported by the Function Call Driver and the functions that you can use to define each operation.

2. Write your application program. Refer to the following for additional information:
   - Preliminary Tasks, the next section, which describes the programming tasks that are common to all application programs.
   - Analog Input Programming Tasks on page 3-6, which describes operation-specific programming tasks and the sequence in which these tasks must be performed.
   - Chapter 4, which contains detailed descriptions of the FCD functions.
   - The example programs in the ASO-4100 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 3-12 to page 3-16 for C/C++ or page 3-16 for Visual Basic for Windows), or to the EXAMPLES.TXT file in the installation directory for compile and link statements and other language-specific considerations for each supported language.

Preliminary Tasks

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

1. Include the function and variable type definition file for your language. This file is included in the ASO-4100 software package.
2. Declare and initialize program variables.
3. Use **K_OpenDriver** to initialize the driver.
4. Use **K_GetDevHandle** to specify the board you want to use and to initialize the board. If you are using more than one board, use the board initialization function once for each board you are using.

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

Analog Input Programming Tasks

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

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

   Otherwise, use the **K_SetBuf** function to assign the starting address of the array or buffer and to specify the number of samples in the array or buffer.
4. Use the appropriate setup functions to specify the attributes of the operation. The setup functions are listed in Table 3-2.

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

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

Attribute	Setup Function(s)
Channel	K_SetChn K_SetStartStopChn
Input Range	K_SetG
Channel-Gain Queue	K_SetChnGAry
Clock Source	K_SetClk

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

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

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

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

C/C++ Programming Information

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

Note: When programming in C/C++, proper typecasting may be required to avoid C/C++ type-mismatch warnings. Make sure that linker options are set so that case-sensitivity is disabled.

Dimensioning and Assigning a Local Array

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

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

Refer to the example programs on disk for more information.

Dynamically Allocating and Assigning a Memory Buffer

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

Note: If you are programming in Windows Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you install the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer; refer to the *DAS-4100 Series User's Guide* for more information.

Allocating a Memory Buffer

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

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

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

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

Accessing the Data

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

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

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

Creating a Channel-Gain Queue

The DASDECL.H and DASDECL.HPP file define a special data type (GainChanTable) that you can use to declare your channel-gain queue. GainChanTable is defined as follows:

```
typedef struct GainChanTable
{
   WORD num_of_codes;
   struct {
      BYTE Chan;
      chan Gain;
   } GainChanAry[256];
} GainChanTable;
```

The following example illustrates how to create a channel-gain queue called MyChanGainQueue for a DAS-4100 board by declaring and initializing a variable of type GainChanTable.

```
GainChanTable MyChanGainQueue =
{
   2, // number of entries
   0, 7 // Channel 0, gain of 7
1, 3 // Channel 1, gain of 3
};
```

After you create MyChanGainQueue, you must assign the starting address of MyChanGainQueue to the frame defined by hFrame, as follows:

```
wDasErr = K_SetChnGAry (hFrame, &MyChanGainQueue);
```

When you start the next analog input operation (using **K_IntStart**), channel 0 is sampled at a gain of 7, and channel 1is sampled at a gain of 3.

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-4100 software package.

File	Description
DAS4100.LIB	Linkable driver
DASRFACE.LIB	Linkable driver
DASDECL.H	Include file when compiling in C
DASDECL.HPP	Include file when compiling in C++
USE4100.OBJ	Linkable object

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

Type of Compile	Compile and Link Statements[1]
C	CL /c *filename*.c LINK *filename*+use4100.obj,,,das4100+dasrface;
C++	CL /c *filename*.cpp LINK *filename*+use4100.obj,,,das4100+dasrface;

Notes

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

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

To program in Microsoft C/C++ (for Windows), including Microsoft Visual C++, you need the following files; these files are provided in the ASO-4100 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library
DASSUPRT.DLL	Dynamic Link Library
DAS4100.DLL	Dynamic Link Library
DASDECL.H	Include file when compiling in C
DASDECL.HPP	Include file when compiling in C++
DASIMP.LIB	DAS Shell Imports

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

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

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

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

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

To program in Borland C/C++ (for DOS), you need the following files; these files are provided in the ASO-4100 software package.

File	Description
DAS4100.LIB	Linkable driver
DASRFACE.LIB	Linkable driver
DASDECL.H	Include file when compiling in C
DASDECL.HPP	Include file when compiling in C++
USE4100.OBJ	Linkable object

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

Type of Compile	Compile and Link Statements[1]
C	BCC *filename*.c use4100.obj das4100.lib dasrface.lib
C++	BCC *filename*.cpp use4100.obj das4100.lib dasrface.lib

Notes

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

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

To program in Borland C/C++ (for Windows), you need the following files; these files are provided in the ASO-4100 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library
DASSUPRT.DLL	Dynamic Link Library
DAS4100.DLL	Dynamic Link Library
DASDECL.H	Include file when compiling in C
DASDECL.HPP	Include file when compiling in C++
DASIMP.LIB	DAS Shell Imports

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

Type of Compile	Compile and Link Statements
C	BCC -c *filename*.c TLINK *filename*,,,dasimp, *filename*.def; BRC -r *filename*.rc BRC *filename*.res
C++	BCC -c *filename*.cpp TLINK *filename*,,,dasimp, *filename*.def; BRC -r *filename*.rc BRC *filename*.res

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

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

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

Microsoft Visual Basic for Windows Programming Information

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

Dimensioning and Assigning a Local Array

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

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

Refer to the example programs on disk for more information.

Dynamically Allocating and Assigning a Memory Buffer

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

Note: If you are programming in Windows Enhanced mode, you may be limited in the amount of memory you can allocate. It is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate a large enough buffer. Refer to your *DAS-4100 Series User's Guide* for more information.

Allocating a Memory Buffer

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

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

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

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

Accessing the Data

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

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

Creating a Channel-Gain Queue

Before you create your channel-gain queue, you must declare an array of integers to accommodate the required number of entries. Next, you fill the array with the channel-gain information. After you create the channel-gain queue, use **K_FormatChnGAry** to reformat the channel-gain queue so that it can be used by the DAS-4100 Series Function Call Driver.

The following code fragment illustrates how to create a two-entry channel-gain queue called MyChanGainQueue for a DAS-4100 board and how to use **K_SetChnGAry** to assign the starting address of MyChanGainQueue to the frame defined by hFrame.

```
. . .
Global MyChanGainQueue(5) As Integer
. . .
MyChanGainQueue(0) = 2 ' Number of channel-gain pairs
MyChanGainQueue(1) = 0 ' Channel 0
MyChanGainQueue(2) = 7 ' Use gain of 7 on channel 0
MyChanGainQueue(3) = 1 ' Channel 1
MyChanGainQueue(4) = 3 ' Use gain of 3 on channel 1
. . .
wDasErr = K_FormatChnGAry (MyChanGainQueue(0))
wDasErr = K_SetChnGAry (hFrame, MyChanGainQueue(0))
```

Once the channel-gain queue is formatted, your Visual Basic for Windows program can no longer read it. To read of modify the array after it has been formatted, you must use **K_RestoreChnGAry** as follows:

```
. . .
wDasErr = K_RestoreChnGAry (MyChannelGainQueue(0))
. . .
```

When you start the next analog input operation (using **K_IntStart**), channel 0 is sampled at a gain of 7, and channel 1 is sampled at a gain of 3.

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-4100 SERIES ERROR”
   End
End If
. . .
```

Programming in Microsoft Visual Basic for Windows

To program in Microsoft Visual Basic for Windows, you need the following files; these files are provided in the ASO-4100 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library
DASSUPRT.DLL	Dynamic Link Library
DAS4100.DLL	Dynamic Link Library
DASDECL.BAS	Include file; must be added to the project

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

4

Function Reference

The FCD functions are organized into the following groups:

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

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

Table 4-1. Functions

Function Type	Function Name	Page Number
Initialization	K_OpenDriver	page 4-31
	K_CloseDriver	page 4-6
	K_GetDevHandle	page 4-16
	K_FreeDevHandle	page 4-10
	K_DASDevInit	page 4-8
Operation	K_IntStart	page 4-25
	K_IntStatus	page 4-26
	K_IntStop	page 4-28
Frame Management	K_GetADFrame	page 4-12
	K_FreeFrame	page 4-11
	K_ClearFrame	page 4-5
Memory Management	K_IntAlloc	page 4-22
	K_IntFree	page 4-24
	K_MoveBufToArray	page 4-30
Buffer Address	K_SetBuf	page 4-38
	K_SetBufI	page 4-40
Channel and Gain	K_SetChn	page 4-42
	K_SetStartStopChn	page 4-53
	K_SetChnGAry	page 4-43
	K_FormatChnGAry	page 4-9
	K_RestoreChnGAry	page 4-33
	K_SetG	page 4-50
Clock	K_SetClk	page 4-45
	K_SetClkRate	page 4-46
	K_GetClkRate	page 4-14

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Trigger	K_SetTrig	page 4-54
	K_SetADTrig	page 4-36
	K_SetDITrig	page 4-48
	K_SetAboutTrig	page 4-34
	K_ClrAboutTrig	page 4-7
	K_SetPostTrigDelay	page 4-52
Miscellaneous	K_GetErrMsg	page 4-18
	K_GetVer	page 4-20
	K_GetShellVer	page 4-19

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-4100 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 device handle, a driver handle, a frame handle, or a memory handle. The actual variable is passed to the driver by reference.
p	Pointer to a variable	This data type is used for pointers to all types of variables, except handles (h). It is typically used when passing a parameter of any type to the driver by reference.
n	Number value	This data type is used when passing a number, typically a byte, to the driver by value.
w	16-bit word	This data type is typically used when passing an unsigned integer to the driver by value.
a	Array	This data type is typically used in conjunction with other prefixes listed here; for example, *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 3-1 on page 3-3 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 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 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*.

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 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

This function stops all current operations on the board specified by *hDev* and verifies that the board identified by the device handle is the board specified in the configuration file associated with the board.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

Purpose Converts the format of a channel-gain queue.

Prototype

C/C++
Not supported

Visual Basic for Windows
Declare Function K_FormatChnGAry Lib "DASSHELL.DLL" (*pArray* As Integer) As Integer

Parameters

pArray	Channel-gain queue starting address.

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

Remarks This function converts a channel-gain queue created in Visual Basic for Windows using 16-bit values to a channel-gain queue of 8-bit values that the **K_SetChnGAry** function can use, and stores the starting address of the converted channel-gain queue in *pArray.*

After you use this function, your program can no longer read the converted queue. You must use the **K_RestoreChnGAry** function to return the queue to its original format. Refer to page 4-33 for more information.

See Also K_SetChnGAry, K_RestoreChnGAry

Usage

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

```
...
Global ChanGainArray(16) As Integer   ' Chan/Gain array
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2   ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 7
wDasErr = K_FormatChnGAry (ChanGainArray(0))
```

K_FreeDevHandle

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 to your project)

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

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 to your project)

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

K_GetADFrame

Purpose

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

Prototype

C/C++

DASErr far pascal K_GetADFrame (DWORD *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 interrupt-mode 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 3-1 on page 3-3.

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 to your project)

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

K_GetClkRate

Purpose

Gets a count value that represents the conversion rate.

Prototype

C/C++

DASErr far pascal K_GetClkRate (DWORD *hFrame*,
DWORD far **pRate*);

Visual Basic for Windows

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

Parameters

hFrame — Handle to the frame that defines the operation.

pRate — Count value.
Value stored: **256** to **32768**, described as follows:

Conversion Rate	Sample Period	Count Value
64 Msamples/s	15.625 ns	256
32 Msamples/s	31.25 ns	512
16 Msamples/s	62.5 ns	1024
8 Msamples/s	125 ns	2048
4 Msamples/s	250 ns	4096
2 Msamples/s	500 ns	8192
1 Msamples/s	1 µs	16384
500 Ksamples/s	2 µs	32768

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions 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 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*.

See Also

K_FreeDevHandle

Usage

C/C++

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

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

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

K_GetErrMsg

Purpose

Gets the address of an error message string.

Prototype

C/C++

DASErr far pascal K_GetErrMsg (DWORD *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);
```

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 (0FF hex).

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_GetVer

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-4100 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 2.1, 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).

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_IntAlloc

Purpose

Dynamically allocates a buffer for an interrupt-mode 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:

Board	Number of Samples
DAS-4101/64K	1 to 65536
DAS-4101/256K	1 to 262144
DAS-4101/1M	1 to 1048526
DAS-4101/2M	1 to 2097150
DAS-4102/64K	1 to 131072
DAS-4102/256K	1 to 524288
DAS-4102/1M	1 to 2097150

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 dynamically 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 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*.

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

See Also K_IntFree, K_SetBuf

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_IntFree

Purpose

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

Prototype

C/C++

DASErr far pascal K_IntFree (WORD *hMem*);

Visual Basic for Windows

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

Parameters

hMem	Handle to interrupt buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

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

See Also

K_IntAlloc

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

Purpose

Starts an interrupt-mode 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 interrupt-mode operation defined by *hFrame*. Refer to page 3-6 for a discussion of the programming tasks associated with interrupt-mode operations.

See Also

K_IntStatus, K_IntStop

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_IntStatus

Purpose

Gets the status of an interrupt-mode 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 interrupt operation; see **Remarks** for value stored.
pCount	Current number of samples transferred into the array or buffer.

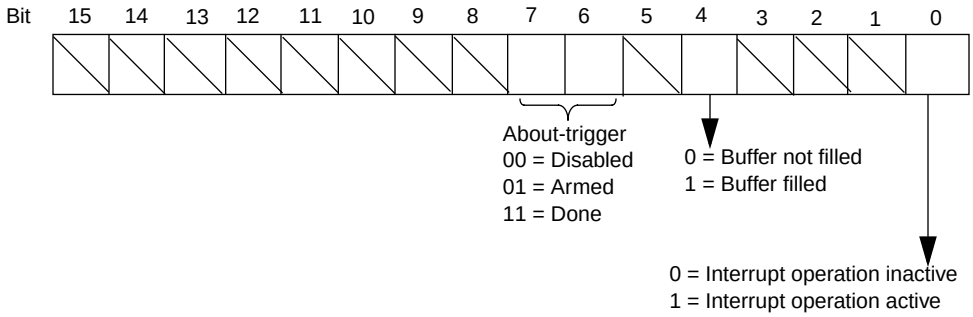
Return Value

Error/status code. Refer to Appendix A.

Remarks

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

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



The bits are described as follows:

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

See Also K_IntStart, K_IntStop

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_IntStop

Purpose

Stops an interrupt-mode operation.

Prototype

C/C++

DASErr far pascal K_IntStop (DWORD *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 interrupt operation. Refer to page 4-26 for more information on the status word returned.
pCount	Current number of samples transferred into the array or buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

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

If you are using an external start or about trigger, call this function if the trigger event does not occur.

See Also

K_IntStart, K_IntStatus

Usage

C/C++

```
#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 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. Valid values: **1** to **32768**

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function transfers the number of samples specified by *nCount* from the buffer at address *pSource* to the array at address *pDest*.

In Visual Basic for Windows, the buffer allocated through **K_IntAlloc** is not accessible to your program; you must use **K_MoveBufToArray** to move the data from the allocated buffer to the program's local array.

See Also

K_IntAlloc

Usage

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

```
...
Dim ADArray(10000) As Integer
...
wDasErr = K_IntAlloc (hAD, 8192, pBuf, hMem)
...
wDasErr = K_MoveBufToArray (ADArray(0), pBuf, 8192)
```

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:	**"DAS4100"** (for DAS-4100 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-4100 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-4100 Series boards, the string stored in *szDrvName* must be DAS4100. 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 CFG4100.EXE utility. Refer to your *DAS-4100 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 hDrv;
...
wDasErr = K_OpenDriver ("DAS4100", "DAS4100.CFG", &hDrv);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
DIM hDrv As Long
...
wDasErr = K_OpenDriver("DAS4100", "DAS4100.CFG", hDrv)
```

Purpose

Restores a converted channel-gain queue.

Prototype

C/C++
Not supported

Visual Basic for Windows
Declare Function K_RestoreChnGAry Lib "DASSHELL.DLL" (*pArray* As Integer) As Integer

Parameters

pArray	Channel-gain queue starting address.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function restores the channel-gain queue at the address specified by *pArray* to its original format so that it can be used by your Visual Basic for Windows program. The channel-gain queue was converted using **K_FormatChnGAry**.

Refer to page 4-9 for more information about the **K_FormatChnGAry** function.

See Also

K_FormatChnGAry, K_SetChnGAry

Usage

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

```
...
Global ChanGainArray(16) As Integer ' Chan/Gain array
...
wDasErr = K_RestoreChnGAry (ChanGainArray(0))
```

K_SetAboutTrig

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:

Board	Number of Sample
DAS-4101/64K	2 to 65534
DAS-4101/256K	2 to 262142
DAS-4101/1M	2 to 1023998
DAS-4101/2M	2 to 2047998
DAS-4102/64K	2 to 131070
DAS-4102/256K	2 to 524286
DAS-4102/1M	2 to 2047998

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*. At least one pre-trigger sample must be in the buffer.

Note that you cannot use an about trigger with an external start trigger.

See Also

K_ClrAboutTrig

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_SetADTrig

Purpose

Sets up an external analog start or about trigger.

Prototype

C/C++

DASErr far pascal K_SetADTrig (DWORD *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. Valid values: **0** for Positive edge **2** for Negative edge
nChan	Trigger channel. Valid values: **0** for ±4 V signal from Channel A connector **1** for ±4 V signal from Channel B connector (Valid for DAS-4102 boards only) **2** for ±16 V signal from Analog Trigger In connector
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*.

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

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

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

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_SetBuf

Purpose

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

Prototype

C/C++

DASErr far pascal K_SetBuf (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Visual Basic for Windows

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

Parameters

hFrame — Handle to the frame that defines the operation.

pBuf — Starting address of array or buffer.

dwSamples — Number of samples.
Valid values:

Board	Number of Samples
DAS-4101/64K	1 to 65536
DAS-4101/256K	1 to 262144
DAS-4101/1M	1 to 1048526
DAS-4101/2M	1 to 2097150
DAS-4102/64K	1 to 131072
DAS-4102/256K	1 to 524288
DAS-4102/1M	1 to 2097150

Return Value

Error/status code. Refer to Appendix A.

Remarks

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

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

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

The *pBuf* variable sets the value of the Buffer element; the *dwSamples* variable sets the value of the Number of Samples element.

See Also

K_IntAlloc, K_SetBufI

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_SetBufI

Purpose

Specifies the starting address of a locally dimensioned integer array and the number of samples in the array.

Prototype

C/C++
Not supported

Visual Basic for Windows
Declare Function K_SetBufI Lib "DASSHELL.DLL" Alias "K_SetBuf" (ByVal *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

Usage

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

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

K_SetChn

Purpose

Specifies a single channel.

Prototype

C/C++

DASErr far pascal K_SetChn (DWORD *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 the operation. Valid values: **0** for signal from Ch A connector **1** for signal from Ch B connector

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 the analog input signal from the Channel A connector on the board; software channel 1 corresponds to the analog input signal from the Channel B connector on the board.

The *nChan* variable sets the Channel element.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

Purpose

Specifies the starting address of a channel-gain array.

Prototype

C/C++

DASErr far pascal K_SetChnGAry (DWORD *hFrame*, void far **pArray*);

Visual Basic for Windows

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

Parameters

hFrame	Handle to the frame that defines the operation.
pArray	Channel-gain queue starting address.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the starting address of the channel-gain queue in *pArray.*

The *pArray* variable sets the Channel-Gain Queue element.

Refer to Chapter 3 for information on setting up a channel-gain queue.

If you created your channel-gain queue in Visual Basic for Windows, you must use **K_FormatChnGAry** to convert the channel-gain queue before you specify the address with **K_SetChnGAry.**

See Also

K_FormatChnGAry, K_RestoreChnGAry

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
//DECLARE AND INITIALIZE CHAN/GAIN PAIRS
//(GAINCHANTABLE-TYPE IS DEFINED IN dasadecl.h)
GainChanTable ChanGainArray = {2.  // # of entries
   0, 3,    // chan 0, gain 3
   1, 7);   // chan 1, gain 7

wDasErr = K_SetChnGAry (hAD, &ChanGainArray);
```

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

```
...
Global ChanGainArray(16) As Integer
...
'Create the array of channel/gain pairs
ChanGainArray(0) = 2  '# of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 7
wDasErr = K_FormatChnGAry (ChanGainArray(0))
wDasErr = K_SetChnGAry (hAD, ChanGainArray(0))
```

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 *nMode* variable sets the Clock Source element.

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 to your project)

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

K_SetClkRate

Purpose

Specifies a count value that represents the conversion rate.

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 — Count value.
Valid values: **256** to **32768**, described as follows:

Conversion Rate	Sample Period	Count Value
64 Msamples/s	15.625 ns	256
32 Msamples/s	31.25 ns	512
16 Msamples/s	62.5 ns	1024
8 Msamples/s	125 ns	2048
4 Msamples/s	250 ns	4096
2 Msamples/s	500 ns	8192
1 Msamples/s	1 µs	16384
500 Ksamples/s	2 µs	32768

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the count value, which is divided into 16.384 Gsamples/s, for the internal pacer clock in *dwDivisor.*

The *dwDivisor* variable sets the Pacer Clock Rate element.

Refer to page 2-9 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, 256);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
wDasErr = K_SetClkRate (hAD, 256)
```

K_SetDITrig

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

This function specifies the use of a digital trigger for the operation defined by *hFrame*.

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-4100 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 *nOpt* variable sets the value of the Trigger Polarity element; the *nChan* variable sets the value of the Trigger Channel element; the *nPattern* variable sets the value of the Trigger Pattern element.

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

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

K_SetG

Purpose

Sets the input range.

Prototype

C/C++

DASErr far pascal K_SetG (DWORD *hFrame*, short *nGain*);

Visual Basic for Windows

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

Parameters

hFrame — Handle to the frame that defines the operation.

nGain — Gain code.
Valid values: **0** to **15**, described as follows:

Analog Input Range	Gain Code	Analog Input Range	Gain Code
±100 mV	12	±800 mV	15
±125 mV	8	±1.0 V	5, 11
±250 mV	4	±1.25 V	6
±400 mV	13	±2.0 V	1, 7
±500 mV	0, 9, 14	±2.5 V	2
±625 mV	10	±4.0 V	3

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the gain code, which represents the input voltage range, for a single channel in *nGain*. Refer to Appendix C to understand the effect of input voltage ranges on the bandwidth of the DAS-4100 Series board.

The *nGain* variable sets the Gain element.

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 to your project)

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

K_SetPostTrigDelay

Purpose

Sets the number of post-trigger delay samples.

Prototype

C/C++

DASErr far pascal K_SetPostTrigDelay (WORD *hFrame*,
DWORD *nDelay*);

Visual Basic for Windows

Declare Function K_SetPostTrigDelay Lib "DAS4100.DLL"
(ByVal *hFrame* As Long, *nDelay* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nDelay	Post-trigger delay samples. Valid values: **0** to **8388608**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the number of post-trigger delay samples in *nDelay* .

You cannot specify a post-trigger delay if you are using an about trigger.

Usage

C/C++

```
#include "DASDECL.H"   // Use "DASDECL.HPP for C++
...
WORD nDelay;
...
wDasErr = K_SetPostTrigDelay (hFrame, hDelay);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

Purpose

Set the channel to be acquired.

Prototype

C/C++

DASErr far pascal K_SetStartStopChn (DWORD *hFrame*, short *nStart, short nStop*);

Visual Basic for Windows

Declare Function K_SetStartStopChn Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nStart* As Integer,
ByVal *nStop* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nStart	Start channel. Valid values: **0** or **1**
nStop	Stop channel. Valid values: **0** or **1**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the two channels which will be used. When *nStart* equals *nStop* then only that channel is acquired. If *nStart* does not equal *nStop,* then *nStart* must be less than *nStop*.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
wDasErr = K_SetStartStopChn (hAD, 0, 1)
```

K_SetTrig

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 or about 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 application program calls **K_IntStart**. An external trigger is either an analog trigger or a digital trigger. Refer to page 2-11 for more information on trigger sources.

You can specify an internal start trigger (*nMode* = **0**) for post-trigger, pre-trigger, and about-trigger acquisitions. You can specify an external start trigger (*nMode* = **1**) for post-trigger acquisitions only. If you want to use an external start trigger, ensure that the about trigger is disabled by using the **K_ClrAboutTrig** function. Refer to page 2-14 for more information on trigger acquisitions.

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.

Usage

C/C++

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

Visual Basic for Windows

(Add DASDECL.BAS to your project)

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

A

Error/Status Codes

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

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

Table A-1. Error/Status Codes

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

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

Error Code		Cause	Solution
Hex	Decimal		
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.
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.
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.

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

Error Code		Cause	Solution
Hex	Decimal		
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**.
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.

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

Error Code		Cause	Solution
Hex	**Decimal**		
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.
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.
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.
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.
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.

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

Error Code		Cause	Solution
Hex	**Decimal**		
6035	24629	**Driver in use:** The driver attempted to configure a device that had already been configured by a call to **K_OpenDriver**. (This can occur since, under Windows, it is possible to open the same driver from multiple programs that are running simultaneously.)	To continue using the driver with the same configuration, pass a null string as the second argument to **K_OpenDriver**. To use the driver with a different configuration, close any application programs currently accessing the driver, and then open the driver again (using **K_OpenDriver**).
6036	24630	**Illegal driver handle:** The specified driver handle is not valid.	Someone may have closed the driver; if so, use **K_OpenDriver** to reopen the driver with the desired driver handle. Try again using another driver handle.
6037	24631	**Driver not found:** The specified driver cannot be found.	Check your link statement to make sure the specified driver is included. Make sure that the device name string is entered correctly in **K_OpenDriver**.
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**.

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

Error Code		Cause	Solution
Hex	**Decimal**		
6039	24633	**Invalid destination pointer:** (Windows-based languages only) The pointer to the destination buffer (local array) that you passed as an argument to **K_MoveBufToArray** is invalid for the specified count. (The destination pointer, when added to the number of samples, exceeds the dimension of the local array.)	Check the dimension of the local array and the number of samples to transfer that you specified in **K_MoveBufToArray**.
603A	24634	**Illegal setup value:** An illegal value was passed to the function in which the error occurred.	Check the legal ranges of all parameters passed to this function.
603B	24635	**Error freeing buffer selector:** **K_DMAFree** or **K_IntFree** failed because one or more of the selectors that reference the memory buffer could not be freed.	Check that the memory buffer being freed was previously obtained through **K_DMAAlloc** or **K_IntAlloc.**
603C	24636	**Error allocating buffer selector:** **K_DMAAlloc** or **K_IntAlloc** failed because a selector could not be allocated from Window's Local Descriptor Table.	Close all applications and restart Windows. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
603D	24637	**Error allocating memory buffer:** **K_DMAAlloc** or **K_IntAlloc** failed because a necessary internal buffer could not be allocated to complete the operation.	Close all applications and restart Windows. If the error continues, contact the Keithley MetraByte Applications Engineering Department.
7000	28672	**No board number**: The board number field was missing or out of place in the specified configuration file.	Specify the board number in the configuration file.
7002	28674	**Illegal board numbe**r: The driver initialization function found an illegal board number in the specified configuration file.	Specify a legal board number: 0 to 1

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

Error Code		Cause	Solution
Hex	Decimal		
7003	28675	**Illegal base address**: The driver initialization function found an illegal base I/O address in the specified configuration file.	Specify a base I/O address in the inclusive range &H240 (576) to &H2F8 (760) in increments of 8H (8). Make sure that &H precedes hexadecimal numbers.
7004	28676	**Illegal memory address**: The driver initialization function found an illegal memory address in the specified configuration file.	Specify a memory address in the inclusive range &HA000 to &HDC00 in increments of 400H. Make sure that &H precedes hexadecimal numbers.
7005	28677	**Illegal interrupt level**: The driver initialization function found an illegal interrupt level in the specified configuration file.	Specify a legal interrupt level: 2, 5, 7, 10, 11, 12, or 15
7007	28679	**Illegal zero wait state**: The driver initialization function found an illegal input in the specified configuration file.	Specify enabled or disabled.
7009	28681	**Illegal A/D channel**: The specified analog input channel is illegal.	Make sure that the specified analog input channel is 0 (Channel A connector) or 1 (Channel B connector).
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 channel scan sequence**: You specified the channels and gains in the channel-gain queue in the wrong order.	In the channel-gain queue, make sure that you specify channel 0 (Channel A connector) and the gain for channel 0 before you specify channel 1 (Channel B connector) and the gain for channel 1.

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

Error Code		Cause	Solution
Hex	Decimal		
700C	28684	**Illegal about- and post-trigger delay combination**: The about trigger cannot be enabled while the post-trigger delay is specified.	Either disable the about trigger using **K_ClrAboutTrig** or do not specify a delay with **K_SetPostTrigDelay**.
700D	28685	**Illegal DAS Specification revision number**: This version of the DAS shell is not supported	Contact Keithley MetraByte for an upgrade.
700E	28686	**Resource busy**: The application program attempted to start an operation while a similar operation was in progress.	Use **K_IntStop** to stop the in-progress operation before initiating the second operation.
7010	28688	**Illegal number of post-trigger delay samples**: You specified an invalid number of post-trigger delay samples.	Use **K_SetPostTrigDelay** to specify a valid number of post-trigger delay samples (between 0 and 8,388,608).
7012	28690	**Illegal channel-gain array size**: The channel-gain queue cannot have more than two channels.	Shorten the channel-gain queue to one or two entries.
7015	28693	**Illegal number of about-trigger samples**: The number of about-trigger samples exceeds the buffer size.	Ensure that the number of about-trigger samples specified in **K_SetAboutTrig** is less than or equal to the size of your array or buffer as specified in **K_SetBuf** or **K_SetBufI**.
7017	28695	**Illegal calibration on board**: The EEPROM settings are not giving a consistent value.	Contact the Keithley MetraByte Applications Engineering Department.
7018	28696	**Memory map conflict or inconsistency**: The board could not map into the memory address configured.	Configure the jumpers on the board to choose another memory address.

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

Error Code		Cause	Solution
Hex	Decimal		
7019	28697	**Windows could not supply memory map:** Windows did not return the selector for the memory location of the board requested.	Check to make sure that your memory manager excludes the memory your board is using. For example, if you are using EMM386, your CONFIG.SYS file should contain a line similar to the following: DEVICE=C:\DOS\EMM386.EXE X=CC00-CFFF (Note this should be typed on one line.)
7022	28706	**No interrupt generated:** The DAS driver could not detect any interrupts from the 4100 board.	Make sure the board's internal jumper matches the settings in the configuration file. Also, make sure no other devices in the computer share the IRQ level.
7023	28707	**Illegal coupling specified**: The DC/AC coupling setting in the configuration file is illegal.	Specify a legal setting: DC or AC
7024	28708	**Illegal impedance specified**: The input impedance setting in the configuration file is illegal.	Specify a legal setting: 50 Ω to 1 MΩ
7025	28709	**Illegal number of samples**: You requested more samples than the board supports.	Make sure that you specify a legal number of samples.
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.

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 application programs that are running and using computer resources.
8016	32790	**Interrupt overrun**: The board communicated a hardware event to the software by generating a hardware interrupt, but the software was still servicing a previous interrupt. This is usually caused by a pacer clock rate that is too fast.	Check the maximum throughput rate for your computer's programming environment and use **K_SetClkRate** to specify an appropriate rate.
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.
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 onboard 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-4100 Series Function Call Driver can read and write counts only. When reading a value, you may want to convert the count to a more meaningful voltage value; when writing a value (as in **K_SetTrig**), you must convert the voltage value to a count value.

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

Converting Counts to Voltage

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

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

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

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

Gain Code	Input Range	Span (V)
0	±500 mV	1
1	±2 V	4
2	±2.5 V	5
3	±4 V	8
4	±250 mV	0.5
5	±1 V	2
6	±1.25 V	2.5
7	±2 V	4
8	±125 mV	0.25
9	±500 mV	1
10	±625 mV	1.25
11	±1 V	2
12	±100 mV	0.2
13	±400 mV	0.8
14	±500 mV	1
15	±800 mV	1.6

For example, assume that you want to read analog input data from a channel on a DAS-4100 Series board with an input range of ±1 V (span of 2 V); the count value is 72. The voltage is determined as follows:

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

Converting Voltage to Counts

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

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

For ±4 V trigger signal from Channel A or Channel B connector:

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

For ±16 V trigger signal from Analog Trigger In connector:

$$\text{Count} = \frac{V_{trig} \times 256}{32}$$

For example, assume that you want to specify an analog trigger level of 125 mV for the input signal coming into the Channel A connector on a DAS-4100 Series board configured for an input range type of ±200 mV (span of 0.4 V). The count value is determined as follows:

$$\frac{0.\dot{1}25 \text{ V} \times 256}{0.4 \text{ V}} = 80$$

Alternatively, assume that you want to specify an analog trigger level of 5 V for the trigger input signal coming into the Analog Trigger In connector on a DAS-4100 Series board. The count value is determined as follows:

$$\frac{\dot{5} \text{ V} \times 256}{32 \text{ V}} = 40$$

Index

A

B

C

D

E

F

H

I

K

M

O

P

R

S

T

V