Programmer's Manual

Producing Custom Software for Data Acquisition Systems

372195A-01

958293

Documents Related to Daq Products

Note: During software installation, Adobe® PDF versions of user's manuals will automatically install onto your hard drive as a part of product support. The default location is in the **Programs** group, which can be accessed from the *Windows Desktop*. Initial navigation is as follows:

Start [Desktop "Start" pull-down menu]
 ⇒ **Programs**
  ⇒ **IOtech DaqX Software**

You can also access the PDF documents directly from the data acquisition CD by using the <**View PDFs**> button located on the opening screen.

Refer to the PDF documentation for details regarding hardware and software pertinent to your data acquisition system.

A copy of the Adobe Acrobat Reader® is included on your CD. The Reader provides a means of reading and printing the PDF documents. Note that hardcopy versions of the manuals can be ordered from the factory.

How to use this Manual

This manual is for individuals who write their own programs. If you prefer to use existing *out-of-the box* software such as DaqView, ViewXL, DASYLab, SnapMaster, you do not need to read this manual.

This manual explains how to program data acquisition systems using various APIs. The programming languages used in the examples are C/C++ and Visual Basic. In addition to the information in this manual, you need to refer to your hardware user's manual. It may be helpful to read the DaqView user's manual to appreciate how a user-friendly data acquisition system looks from the user's point of view. You may also need to consult documentation pertaining to your specific computer system and programming environment.

In regard to this manual, you should read chapter 1, and then refer to additional chapters that are relevant to your programming environment.

The manual is organized as follows:

1. *Introduction* - The manual begins with an overview of issues related to data acquisition programming and takes a look at the options available for making the task as easy as possible.

2. *API Programming – General Models* – discusses aspects of the data acquisition environment, provides several API models and a summary of selected API functions. The chapter includes a section entitled, *Seven Easy Steps to Data Acquisition*.

3. *Using Multiple Devices*

4. *Daq API Command Reference*

Appendix A – *Appendix Removed. Outdated material.*

Appendix B, *Using Borland C++*

Appendix C, *Custom OEM Installation*, explains the procedures required for custom reseller hardware and driver installation and distribution.

Appendix D, *daq9513… Commands*, this appendix only applies to DaqBook/100 Series, DaqBook/200 Series, DaqBoard/100 Series and DaqBoard/200 Series devices. The appendix includes an API programming model for 9513 counter-timer chip devices and includes API reference material for the `daq9513…` command prototypes.

Note: During software installation, Adobe® PDF versions of user manuals will automatically install onto your hard drive as a part of product support. The default location is in the **Programs** directory, which can be accessed from the *Windows Desktop*. Refer to the PDF documentation for details regarding both hardware and software.

A copy of the Adobe Acrobat Reader® is included on your CD. The Reader provides a means of reading and printing the PDF documents. Note that hardcopy versions of the manuals can be ordered from the factory.

Reference Note: The **readme.file** on your install CD-ROM identifies the location of program examples that are included on the CD.

This page is intentionally blank.

Table of Contents

Reference Note:
The **readme.file** on your install CD provides the location of program examples that are included on the CD.

This page is intentionally blank.

Reference Note:
Your companion user's manual discusses hardware installation and setup, theory of operation, troubleshooting, and ready-to-run software. If you plan to use DaqView software [shipped with Daq device products], or you plan to use other ready-to-run software such as DASYLab or SnapMaster, you will not need to use this manual.

Programmers can use the Applications Program Interface (API) to customize software. To create effective programs, programmers must be familiar with the hardware and operation as described in the previous chapters of this document.

Note: The *readme* files on the install CD-ROM will keep you up-to-date as APIs continue to evolve.

Note: Daq PCMCIA is not supported under Windows95/NT drivers.

This manual serves both novice and experienced programmers.

- **As a tutorial** - The *Programming Models* chapter explains how to combine commands to do useful work in a typical data acquisition environment. Program excerpts illustrate concepts and can be modified as needed to use in your programs.

- **As a reference** – A great portion of the manual pertains to API commands. API definitions and parameter values are important to ensure proper syntax, and that software functions perform as intended.

Reference Note:

This manual contains no computer programming tutorials.
You may need to consult additional documentation.

API Features

The install CD-ROM includes several "drivers" to accommodate various programming environments.

The API has several features:
- Multi-device - can concurrently handle up to 4 devices of the Daq device family
- Larger buffer - can handle up to 2 billion samples at a time
- Enhanced acquisition and trigger modes
- Direct-to-disk capabilities
- Wait-on-event features
- Uses multi-tasking advantages of Windows 95/98/Me/NT/2000/XP

Language Support

The following three languages are supported:

C/C++   *Visual Basic*

The *Programming Language Support* folder is located in your installation directory.

You can access program-related files from *Windows Explorer*.

If you used the install default directory setting, the support folder will be located under a program folder name (such as **DaqX**) under the **Program Files** folder on the **C: drive**. The figure below illustrates the *default install location* for your **C/C++** and **Visual Basic** (VB) programming language support.

In the illustration, the **32-Bit Enhanced API** folder (for C/C++ support) has been expanded to reveal its contents.
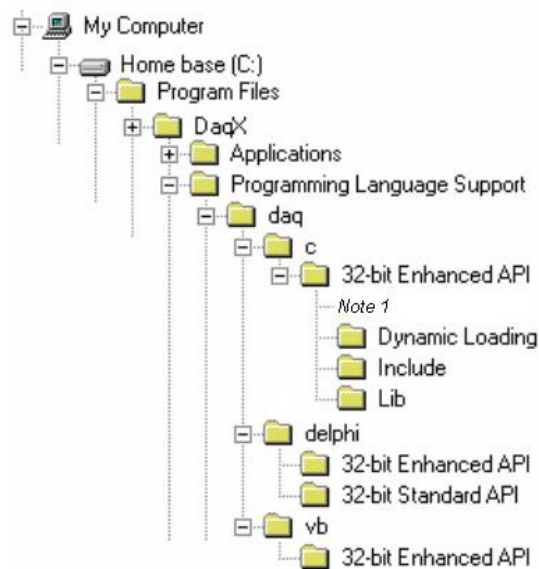
Default Install Locations for Programming Language Support

Note 1: In addition to folders labeled "Dynamic Loading," "Include," and "Lib,"
the 32-bit Enhanced API folder includes program examples for primary
data acquisition units (main units) and for DBKs.

Driver Installation

Driver installation uses a 32-bit setup on a Windows 95/98/Me or Windows NT/2000/XP system. When run, the setup routine will automatically detect the correct operating system and will install the appropriate driver.

A Note Regarding Devices and API Command Applicability

API commands cannot be used universally with all products. For example, a command that pertains only to analog output is of no use to a device that does not support analog I/O, or to a device that supports analog input, but not analog output. Thus it is important that you understand the features and capabilities of your hardware before using API Commands. Prior awareness can save a great deal of programming time and avoid possible frustration.

One category of functionality that sometimes causes confusion is that of *waveform/pattern output*. The following table indicates which of the 4 types of waveform output apply to various product groups. Waveform output definitions follow the table.

Waveform/Pattern Output Device Capabilities				
Product *	Static Waveform	Dynamic Waveform	Waveform Output from Disk	Streamed Digital Pattern Output
DaqBoard/500 Series	Yes	No	No	No
DaqBoard/1000 Series	Yes	Yes	Yes	No
DaqBoard/2000 Series	Yes	Yes	Yes	Yes
DaqBook/2000 Series	Yes	No	No	No
DaqLab/2000 Series	Yes	No	No	No
DaqScan/2000 Series	Yes	No	No	No
/3000 Series Devices	Yes	Yes	Yes	Yes

* Not all devices in a series have static waveform output capability. The feature is only present for devices that have built-in DAC channel output circuitry, or if the device has a DBK46 option installed. Consult your specific hardware documentation for detailed information.

Static Waveform: Data is downloaded to the internal memory on the device and then played out from the device's internal memory. Once the waveform output is downloaded and initiated additional output updates from the PC cannot be performed. However the waveform can be played out indefinitely from the internal memory on the device.

Dynamic Waveform: Data is downloaded to the device and can be continuously updated by the PC while the waveform output is active. Updates to the output can come from PC memory or from a file.

Waveform Output from Disk: Uses dynamic waveform output to continuously update outputs from a file. The file is on a PC, which contains waveform output values, as defined by the user.

Streamed Digital Pattern Output: This output can be via P3 pins, BNC connectors, or SCSI III, depending on the device. The feature allows streaming continuous digital data to the device's 16-bit P3 digital port [or to designated BNC connectors, if applicable]. Data can be streamed from either PC memory or from a file. The rate at which streamed output is updated is dictated by the current DAC waveform output clock source.   Digital data streaming can be performed by itself or in concert with analog waveform output generation.

Become familiar with your system components prior to writing a program. Know the capabilities of each device.

TIP: **Prior to writing a program, review the product specifications [found in your device hardware manual]. The specifications often make it obvious as to which API Commands could not possibly apply. In addition, the programmer needs to be aware of specification limits when entering values.**

Reference Note:
Specific Daq Device and DBK program examples are included on the install CD.
The install CD readme.file states the location of the examples.

This chapter shows how to combine API functions to perform typical tasks. Depending on your level of programming expertise, once you understand how the API works in conjunction with the hardware you can begin creating custom data acquisition programs.

This chapter is divided into two primary sections, as follows:

- **Data Acquisition Environment** outlines related concepts and defines Daq device capabilities the programmer must work with (the API, hardware features, and signal management).

- The **Models** section explains the sequence and type of operations necessary for data acquisition. Some models are provided in Visual Basic, while others appear in C/C++ code. These models provide the software building blocks to develop more complex and specialized programs.

Data Acquisition Environment

To write effective data acquisition software, programmers must understand:

- Software tools (the API documented in this manual and the programming language—you may need to consult documentation for your chosen language)

- Hardware capabilities and constraints

- General concepts of data acquisition and signal management

Application Programming Interface (API)

The API includes all the software functions needed for building a data acquisition system with the hardware described in this manual. The API Command Reference section of this manual includes details regarding how each function is used (parameters, hardware applicability, etc). In addition, you may need to consult your language and computer documentation.

Hardware Capabilities and Constraints

To program the system effectively, you must understand your Daq device and DBK hardware capabilities. Obviously you cannot program the hardware to perform beyond its design and specifications, but you also want to take full advantage of the system's power and features. In the User's Manual, you may need to refer to sections that describe your hardware's capability. In addition, you may need to consult your computer documentation. In some cases, you may need to verify the hardware setup, use of channels, and signal conditioning options (some hardware devices have jumpers and DIP switches that must match the programming, especially as the system evolves).

Signal Environment

Important data acquisition concepts *for programmers* are listed below and are discussed in your device user's manual.

- **Channel Identification**
- **Scan Rates and Sequencing**. With multiple scans, the time between scans becomes a parameter. This time can be a constant or can be dependent upon a trigger.
- **Counter/Timer Operation**
- **Triggering Options**. Triggering starts the A/D conversion. The trigger can be an external analog or TTL trigger, or a program controlled software trigger.

Parameters in the various A/D routines include: number of channels, number of scans, start of conversion triggering, timing between scans, and mode of data transfer. Up to 512 A/D channels can be sampled in a single scan. These channels can be consecutive or non-consecutive with the same or different gains. The scan sequence makes no distinction between local and expansion channels.

Seven Easy Steps to Data Acquisition

The sections that follow this one demonstrate various methods for designing and developing a data acquisition application. Though these models vary widely in their purpose and individually can seem quite complex most all employ a basic framework that is repeated elsewhere in other models. This section will discuss the basic framework required to develop a simple data acquisition application.   The basic framework outlined here will be a re-occuring theme through most, if not all, the subsequent programming models.

Each data acquisition task can be broken down into the following basic elements:

1. **Configuring Channels — What Type of Channels? How Many?**
2. **Configuring Acquisition Events — How Should the Acquisition Start and Stop?**
3. **Setting the Acquisition Rate — How Fast Should the Channels be Scanned?**
4. **Setting up the Buffer Model — How Should the Data be Stored?**
5. **Arming the Acquisition and Starting the Transfer**
6. **Triggering the Acquisition**
7. **Monitoring the Acquisition and Receiving Data**

While this basic framework is not comprehensive, it does provide a basic model and a theory of operation from which to start developing your data acquisition application. Provided with each step is a description of the task and its general function, as well as representative code snippets from which an application can be developed and a table of related API functions which may be used to implement the step.

1. *Configure Channels — What Type of Channels? How Many?*

Every data acquisition has one or more channels from which data is to be acquired. The channels to be scanned comprise the channel scan configuration. Channels are added to the channel scan sequence in the order that they are programmed. Even if some channels are repeated in the channel scan sequence they are added to the channel scan sequence in the programmed location.

NOTE: Hereafter, the term *Channel Scan* (or *Scan*) refers to the entirety of the channel scan sequence configuration. In other words, a *Channel Scan* (*Scan*) is comprised of all the channels programmed into the channel scan sequence.

Here we are only configuring channels on the main unit. If configuring DBK expansion options each DBK channel needs to be programmed into the channel scan sequence. You may also need to set more specific channel configuration options. For this, refer to the sections regarding your specific DBK expansion card.

The following shows how to program the channel scan sequence.

```
DaqAdcGain    gains[CHANCOUNT] = {DgainX1, DgainX2, DgainX8, DgainX4};
DWORD  channels[CHANCOUNT] = {0,1,2,3};
DWORD  flags[CHANCOUNT] =  {DafAnalog|DafBipolar,DafAnalog|DafUnipolar,
            DafAnalog|DafBipolar, DafAnalog|DafUnipolar};

err = daqAdcSetScan(handle, channels, gains, flags,CHANCOUNT);
```

Related API's*	Description
DaqAdcSetScan	Performs basic channel scan sequence configuration, sets up channel scan information
DaqAdcSetMux	Same as daqAdcSetScan but sets all channels to the same configuration parameters
DaqAdcSetOption	Used to configure more complex configuration options for individual channels
DaqAdcExpSetBank	Used to setup channel banks/blocks for use with DBK expansion

*See the *API Command Reference* chapter for detailed information.

2. *Configure Acquisition Events – How Should the Acquisition Start and Stop?*

In this section we describe how to configure the Starting and Stopping Events for your acquisition. Each acquisition needs to have well defined Start and Stop Events. A Start Event can be as simple as start the acquisition immediately upon arming. An acquisition Stop Event can be as simple as Stop upon disarming the acquisition. The number and type of start and stop events is dependent upon the capabilities of the data acquisition device and vary from product to product. Acquisitions may also be defined with pre-trigger data. The amount of pre-trigger data can vary.

```
err = daqAdcSetAcq(handle, DaamNShot, 0,SCANS);
err = daqSetTriggerEvent(handle,DatsSoftware, NULL,NULL,NULL,NULL,NULL,
    DaqStartEvent);
err = daqSetTriggerEvent(handle,DatsScanCount,NULL,NULL,
    NULL,NULL,NULL,NULL, DaqStopEvent);
```

Related API's*	Description
daqAdcSetAcq	Configures acquisition modes as well as pre and post trigger scan counts
daqSetTriggerEvent	Configures Start/Stop Trigger Events.
daqAdcSetTrig	Used to configure Start Trigger Event
daqAdcCalcTrig	Used to calculate Analog level in A/D counts for use in daqAdcSetTrig

*See the *API Command Reference* chapter for detailed information.

3. Set the Acquisition Rate – How Fast Should Channels be Scanned?

In this section we describe how to configure the rate at which data is acquired for your acquisition. Here the acquisition rate refers to the rate at which channel scans are acquired. Depending upon which API is used the acquisition scan rate can be selected by either frequency or period/interval. Note that the rate being programmed is the rate at which the entire channel *scan* is paced. For all Daq devices the scan pacing can be derived from either an internal clock or an external clock driven by an input controlled by an external source. The between-channel sampling interval is normally a fixed interval depending upon the type of device used. Daq devices are normally fixed at 10us sampling interval. The between-channel sampling interval for /2000 Series devices is software selectable for either 5us or 10us.
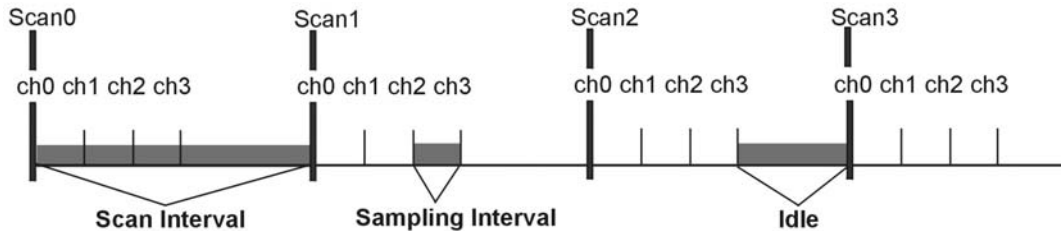

```
err = daqAdcSetFreq(handle, 1000.0);
```

Related API's*	Description
daqAdcSetFreq	Sets the acquisition rate for both pre-trigger and post-trigger acquisitions in Hz based off of the internal pacer clock of the device
daqAdcGetFreq	Returns the current acquisition rate setting.
DaqAdcSetClockSource	Sets the acquisition base clock source to either the internal pacer or to an external clock source.
daqAdcSetRate	Allows setting pre and post-trigger scan rates in either period or frequency format (pre-trigger rate cannot be set for most Daq device products, for Daq device products pre-trigger rate will follow post-trigger rate). This command may also be used to return current pre-trigger and post-trigger rate settings.

*See the *API Command Reference* chapter for detailed information.

4. Setting up the Data Buffer Model – How Should the Data be Stored?

In this section we describe how to configure the buffer model to be used for the acquisition. There are two basic buffer models from which to choose; the User Buffer Model and the Driver Buffer Model.

The User Buffer Model allows the user or application to allocate a buffer and pass the information describing the location and disposition of the buffer down to the driver so that the driver can place collected data into the buffer. However, the User Buffer Model requires that the maintenance of the buffer and buffer pointers be performed by the user or application. When using the User Buffer Model the application can query the driver as to the total amount of data that has been transferred into the buffer but the user/application is responsible for maintaining and updating the current read and write pointers into the buffer. The User Buffer Model can employ either linear or circular buffers depending on the needs of the application.

The Driver Buffer Model the user/application to hand off responsibility of buffer management to the driver.

```
WORD        buffer[SCANS * CHANCOUNT];
err = daqAdcTransferSetBuffer(handle, buffer,SCANS, DatmDriverBuffer);
```

User Buffer Model Operation

The User Buffer Model allows the user to specify either a linear buffer or circular buffer mode.

When the linear buffer mode is selected the driver will start filling the buffer from the beginning of the buffer with the newest available scans. Once the entire buffer has been filled to the number of scans specified the driver will stop writing scan data to the buffer even though the acquisition may continue.
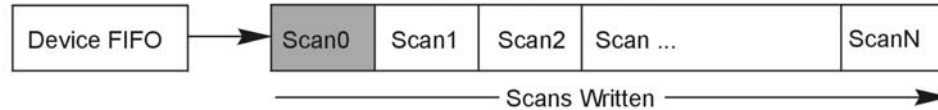
In the User Buffer Linear mode the driver will stop writing data to the user buffer once the total number of scans requested has been satisfied. If the acquisition continues to run after this point scan data will accumulate in the device FIFO until another buffer (or the same buffer) transfer has been started or the device FIFO overruns. The implication of this is that when using linear buffers it is important to make sure that a buffer transfer remains active during the course of the acquisition or data loss may result. In other words, if the acquisition continues past the end of the specified buffer another transfer (into the same or a different buffer) must immediately be initiated. While it is possible to ping pong linear buffers in this manner it is not the recommended. Linear buffers should normally be used only when a predetermined number of scans are to be collected for the acquisition. In this case the buffer scan length should be set to to be the same size as the aggregate scan size [of the total acquisition]. For more information about starting transfers see Step 5.

The User Buffer Model also has a Circular Buffer Mode. In the circular buffer mode the driver will continue to write scan data into the User Buffer until the acquisition terminates on its own or is aborted by the user/application. Unlike the Linear Buffer model when the end of the user buffer is reached the driver will continue to write scan data starting at the beginning of the buffer.

When using the Circular Buffer mode the application does not need to make sure that there is always a buffer ready to take the scan data because the driver simply continues to fill the specified buffer and, when necessary, will begin writing data at the beginning of the buffer. This model ensures that a device FIFO overrun never occurs because the driver always has place to store the data (as long as the interface is capable of the required throughput).   The application is required to monitor the transfer and remove and/or process data as it becomes available (See Step 7 on monitoring and receiving scan data). However a User Buffer overrun may occur if the controlling application cannot keep up in processing or removing the data from the buffer. Therefore the application should allocate a large enough buffer to alleviate any processing or other latencies that may be present in the system or the application. If making the buffer larger does not alleviate user buffer overrun problems then it may be necessary to upgrade the PC to a higher performance unit. The type of upgrade required will be highly dependent upon the nature of the application as well as operating environment in general. For instance, if taking data to disk then a faster HD and controller may be required. If mathematical manipulation of the data is taking place then a faster CPU may be in order. If graphics or video are used intensively then the solution may be a higher performance video card. It is important to remember that the application and other tasks within the system can have an impact on the overall performance of the data acquisition process.

Driver Buffer Model Operation

The Driver Buffer Model allows the flexibility of the User Buffer Model in Circular mode without the complication of having to manage the circular buffer at the application level. In fact, the Buffer Model is simply a special case of the circular buffer mode in that the driver handles the details of managing the circular buffer and "hands off" scan data to the application only when the application requests it. The scan data is handed off to the application in easy to use and manipulate linear buffers.

For more information on retrieving scan data from the Driver Buffer refer to Step 7, *Monitor the Acquisition and Receive the Data.*

Storing Data to Disk

Either buffer model allows the data to be streamed to a disk file in parallel to the transfer into the User or Driver Buffer. To enable this the daqAdcSetDiskFile function needs to be invoked before the acquisition is armed. When using this feature the scan data can be; appended to a current file, overwrite a current file or create a new file.
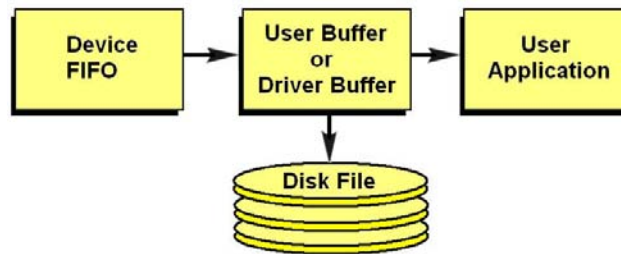
With either the User Buffer Model or the Driver Buffer Model the driver performs the transfer automatically. The format of the data in the disk file being written will be the same at it appears in the buffer. This will normally be raw scan data. No header or channel configuration information is stored in the raw data file.

Related API's*	Description
`daqAdcTransferSetBuffer`	Sets up the scan data buffer for the transfer. The buffer can be configured for either User Buffer or Driver Buffer modes. When User Buffer mode is selected the buffer can be set up to be either linear or circular in nature. The size of the scan data buffer (in scans) is also set here.
`daqAdcSetDiskFile`	Sets ups and enables taking the transfer scan data to a disk file.

*See the *API Command Reference* chapter for detailed information.

5. *Arming the Acquisition and Starting the Transfer*

Once the acquisition has been completely configured the acquisition can be armed. Arming the acquisition means that device will be configured according to the previous steps already outlined and the acquisition will taken from an "idle" to an "active" state. Once in the "active" state the acquisition may begin looking for the Trigger Event or begin collecting pre-trigger data (if pre-trigger has been configured). It is important to examine error return codes from the daqAdcArm command. The daqAdcArm command examines the entire configuration for any potential acquisition parameter conflicts.

```
err = daqAdcTransferStart(handle);
err = daqAdcArm(handle);
```

It is good practice to enable the transfer of data into the buffer first by calling daqAdcTransferStart command. This will ensure that a transfer is active and that the data buffer is ready to receive data so that when the acquisition is triggered the scan data can be immediately placed into the data buffer. Once the daqAdcArm command has been successfully invoked a trigger may occur at any time (obviously, once the trigger condition has been satisfied).

Related API's*	Description
`daqAdcTransferStart`	Enables scan data transfer into the buffer specified by the daqAdcTransferSetBuffer command.
`daqAdcArm`	Arms the acquisition. This command is a pivotal command around which the entire acquisition is configured. Once invoked this command not only arms the acquisition it configures the acquisition according to the acquisition configuration parameters set by prior commands. It is here where any potential configuration conflicts are flagged. Therefore it is important to check return codes from this command.

*See the *API Command Reference* chapter for detailed information.

6. Triggering the Acquisition

Once the acquisition has been armed it may be triggered at any time unless a pre-trigger data has been requested. If pre-trigger data has been requested the trigger event detection will be deferred until at least the specified amount of pre-trigger data has been collected. Any trigger event that occurs before the specified pre-trigger amount has been collected will be ignored. This deferring of the detection of the trigger event until the specified amount of pre-trigger data has been collected ensures that the acquisition will produce, at minimum, the requested pre-trigger amount.

The trigger event can be one of any valid trigger events for which the device is capable. For more information on which trigger events your device is capable of detecting please refer to the daqSetTriggerEvent command in the API definition.

Every device can be triggered via software:

err = daqAdcSoftTrig(handle);

Related API's*	Description
`daqAdcSoftTrig`	Triggers the acquisition if software triggering is enabled.

*See the *API Command Reference* chapter for detailed information.

7. Monitoring the Acquisition and Receiving Data

During the acquisition it may be necessary to monitor the progress of the acquisition and to collect the data and process it. The daqAdcTransferGetStat function returns a total transfer scan count as will as acquisition and transfer state information. Interpretation of the information returned greatly depends on the the buffer model selected as well as whether or not the buffer was set to linear or circular mode operation.

DWORD active,retCount;

DaqAdcTransferGetStat(handle,&active,&retCount);

The active parameter returns state information. The individual bits returned in this parameter indicate the current state of both the acquisition and the transfer. The retCount parameter returns a total running count of the amount of data transferred thus far. The following sections show how to interpret this data:

Monitoring User Buffer Model Transfers

The User Buffer Model allows the user to specify either a linear buffer or circular buffer mode.

When the linear buffer mode is selected the driver will start filling the buffer from the beginning of the buffer of size n (scans) with the newest available scans. Once the entire buffer has been filled to the number of scans specified by n the driver will stop writing scan data to the buffer even though the acquisition may continue.
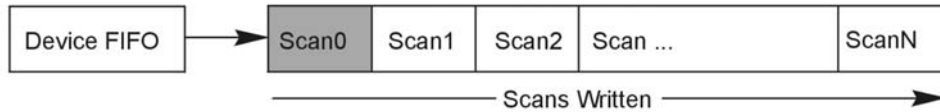
For monitoring progress into a linear User Buffer, the retCount parameter will return the current location (in scans) of the write pointer. This location indicates the next scan to be written as well as representing the total number of scans acquired thus far. For instance, if retCount = m then scan0 through scan$m$-1 can be processed.

The User Buffer Model also has a Circular Buffer Mode. In the circular buffer mode the driver will continue to write scan data into the User Buffer until the acquisition terminates on its own or is aborted by the user/application. Unlike the Linear Buffer model when the end of the user buffer is reached the driver will continue to write scan data starting at the beginning of the buffer.

When using the Circular Buffer mode the application does not need to make sure that there is always a buffer ready to take the scan data because the driver simply continues to fill the specified buffer and, when necessary, will begin writing data at the beginning of the buffer. This model ensures that a device FIFO overrun never occurs because the driver always has place to store the data (as long as the interface is capable of the required throughput).

The application is required to monitor the transfer and remove and/or process data as it becomes available (See Step 7 on monitoring and receiving scan data). However a User Buffer overrun may occur if the controlling application cannot keep up in processing or removing the data from the buffer. Therefore the application should allocate a large enough buffer to alleviate any processing or other latencies that may be present in the system or the application.

If making the buffer larger does not alleviate user buffer overrun problems then it may be necessary to upgrade the PC to a higher performance unit. The type of upgrade required will be highly dependent upon the nature of the application as well as operating environment in general. For instance, if taking data to disk then a faster HD and controller may be required.

If mathematical manipulation of the data is taking place then a faster CPU may be in order. If graphics or video are used intensively then the solution may be a higher performance video card. It is important to remember that the application and other tasks within the system can have an impact on the overall performance of the data acquisition process.

Monitoring and Receiving Driver Buffer Model Data

The Driver Buffer Model allows the flexibility of the User Buffer Model in Circular mode without the complication of having to manage the circular buffer at the application level. In fact, the Buffer Model is simply a special case of the circular buffer mode in that the driver handles the details of managing the circular buffer and "hands off" scan data to the application only when the application requests it. The scan data is handed off to the application in easy to use and manipulate linear buffers. Once handed off the data is removed from the Driver Buffer permanently. In other words requesting data from the Driver Buffer is a destructive read operation.

```
┌──────────────┐      ┌────────────────────────┐      ┌─────────────────────┐
│  Device FIFO │ ───▶ │ Internal Driver Buffer │ ───▶ │ User Request Buffer │
└──────────────┘      └────────────────────────┘      └─────────────────────┘
```

Upon request the Driver Buffer Model allows the application to request data from the internal Driver Buffer as follows:

- **<u>Return Scan Data Available</u>** Returns any unread scan data that is available in the Driver Buffer up to the requested amount. The application must ensure that it has enough space in the User Request Buffer to store any amount of unread scan data that may be present in the Driver Buffer up to the amount requested.

- **<u>Wait Until the Requested Amount is Available</u>** Waits until the requested amount is available in the Driver Buffer. When the amount of scan data available is greater than or equal to the amount requested the Driver Buffer will return the amount requested.

- **<u>Do Not Wait Until the Requested Amount is Available</u>** Upon receipt of the request if the amount of scan data available in the Driver Buffer is not greater than or equal to the amount requested then the Driver Buffer will return *no* scan data. If the amount requested is available at the time of the request the driver buffer will return the amount requested.

Models

Initialization and Error Handling

This section demonstrates how to initialize the Daq device and use various methods of error handling. Most of the example programs use similar coding as detailed in the following VB example. Functions used include:

- `VbdaqOpen&(daqName$)`
- `VbdaqSetErrorHandler&(errHandler&)`
- `VbdaqClose&(handle&)`

All Visual Basic programs should include the DaqX.bas file into their project. The DaqX.bas file provides the necessary definitions and function prototyping for the DAQX driver DLL.

```
handle& = VBdaqOpen&("DaqBook0")
ret& = VBdaqClose&(handle&)
```

```
┌──────────────────────┐
│      daqOpen         │  Open a data
└──────────────────────┘  acquisition session.
          │
          ▼
┌──────────────────────┐
│ daqSetErrorHandler   │  Set up error handling
└──────────────────────┘  (optional).
          │
          ▼
┌──────────────────────┐
│     User Code        │
└──────────────────────┘
          │
          ▼
┌──────────────────────┐
│      daqClose        │  Close the session.
└──────────────────────┘
```

The Daq device is opened and initialized with the `daqOpen` function. `daqOpen` takes one parameter—the name of the device to be opened. The device name can be accessed and changed via the Daq* Configuration utility located in the operating system's Control Panel (see following figure).
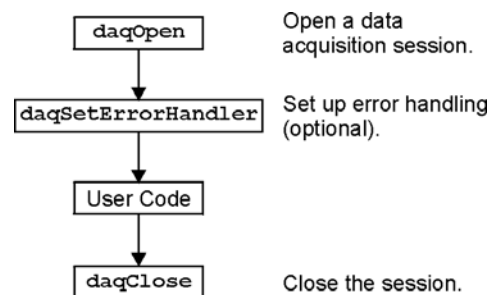
Control Panel
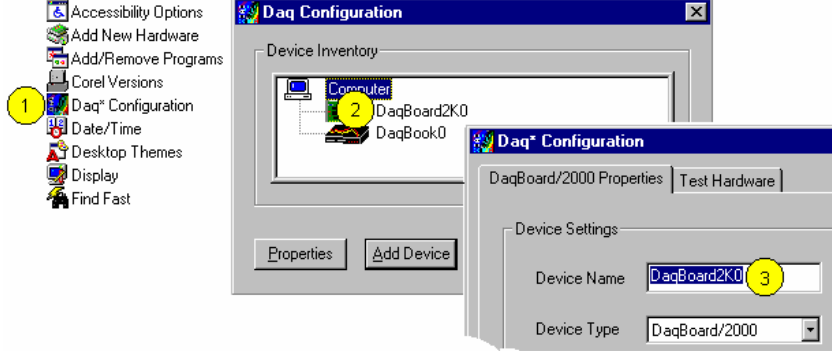
Accessing and Changing a Device Name Using the Control Panel

To change a device name by going through the Control Panel you:

(1) Go to the Control Panel and select "Daq* Configuration.
(2) Double-Click on the applicable device.
(3) Highlight the existing name and type in the new one. Then click the "OK" button, not shown.

The **daqOpen** call, if successful, will return a handle to the opened device. This handle may then be used by other functions to configure or perform other operations on the device. When operations with the device are complete, the device may then be closed using the **daqClose** function. If the device could not be found or opened, **daqOpen** will return -1.

The DAQX library has a default error handler defined upon loading. However; if it is desirable to change the error handler or to disable error handling, then the **daqSetErrorHandler** function may be used to setup an error handler for the driver. In the following example the error handler is set to **0** (no handler defined), which disables error handling.

```
ret& = VBdaqSetErrorHandler&(0&)
```

If there is a Daq device error, the program will continue. The function's return value (an error number or **0** if no error) can help you debug a program.

```
If (VBdaqOpen&("DaqBook0") <  0) Then
     "Cannot open DaqBook0"
```

Daq device functions return **daqErrno&**.

```
     Print "daqErrno& : "; HEX$(daqErrno&)
End If
```

The next statement defines an error handling routine that frees us from checking the return value of every Daq device function call. Although not necessary, this sample program transfers program control to a user-defined routine when an error is detected. Without a Daq device error handler, Visual Basic will receive and handle the error, post it on the screen and terminate the program. Visual Basic provides an integer variable (ERR) that contains the most recent error code. This variable can be used to detect the error source and take the appropriate action.

The function **daqSetErrorHandler** tells Visual Basic to assign ERR to a specific value when a Daq device error is encountered. The following line tells Visual Basic to set ERR to 100 when a Daq device error is encountered. (Other languages work similarly; refer to specific language documentation as needed.)

```
handle& = VBdaqOpen&("DaqBook0")
ret& = VBdaqSetErrorHandler&(handle&, 100)
   On Error GoTo ErrorHandler
```

The **On Error GoTo** command (in Visual Basic) allows a user-defined error handler to be provided, rather than the standard error handler that Visual Basic uses automatically. The program uses **On Error GoTo** to transfer program control to the label ErrorHandler if an error is encountered.

Daq device errors will send the program into the error handling routine. This is the error handler. Program control is sent here on error.

```
ErrorHandler:
    errorString$ = "ERROR in ADC1"
    errorString$ = errorString$ & Chr(10) & "BASIC Error :" + Str$(Err)
    If Err = 100 Then errorString$ = errorString$ & Chr(10) & "DaqBook Error
: " + Hex$(daqErrno&)
    MsgBox errorString$, , "Error!"
End Sub
```

One-Step Command Acquisitions

This section shows the use of several one-step analog input routines. These commands are easier to use than low-level commands but are less flexible in regard to scan configuration. These commands provide a single function call to configure and acquire analog input data. This example demonstrates the use of the 4 Daq device's one-step ADC functions. Functions used include:

- **VBdaqAdcRd&(handle&,chan&, sample%, gain&)**
- **VBdaqAdcRdN&(handle&,chan&, Buf%(), count&, trigger%, level%, freq!, gain&,flags&)**
- **VBdaqAdcRdScan&(handle&,startChan&, endChan&, Buf%(), gain&, flags&)**
- **VBdaqAdcRdScanN&(handle&,startChan& , endChan&, Buf%(), count&, triggerSource&, level%, freq!, gain&, flags&)**

This program will initialize the Daq device hardware, then take readings from the analog input channels in the base unit (not the expansion cards). For transporting data in and out of the Daq device driver, arrays are dimensioned.

daqAdcRd — Read 1 sample from 1 channel.

At this point, the data is in the buffer provided by the user in binary format.

daqAdcRdN — Read multiple samples from 1 channel.

At this point, the data is in the buffer provided by the user in binary format.

daqAdcRdScan — Read 1 sample from multiple channels.

At this point, the data is in the buffer provided by the user in binary format.

daqAdcRdScanN — Read multiple samples from multiple channels.

At this point, the data is in the buffer provided by the user in binary format.

User Code

```
Dim sample%(1), buf%(80), handle&,
   ret&, flags&, gain&
```

The following code assumes that the Daq device has been successfully opened and the **handle&** value is a valid handle to the device. All the following one-step functions define the channel scan groups to be analog unipolar input channels. Specifying this configuration uses the **DafAnalog** and the **DafUnipolar** values in the **flags** parameter. The **flags** parameter is a bit-mask field in which each bit specifies the characteristics of the channel(s) specified. In this case, the **DafAnalog** and the **DafUnipolar** values are added together to form the appropriate bit mask for the specified **flags** parameter.

The next line requests 1 reading from 1 channel with a gain of ×1. The variable **DgainX1&** is actually a defined constant from DaqX.bas, included at the beginning of this program.

Note: DafSigned does not work in conjunction with DaqBook/100.

```
ret& = VBdaqAdcRd&(handle& 0, sample%(0), DgainX1&,
   DafAnalog&+DafUnipolar&+DafSigned&)
Print Format$"&   ####"; "Result of AdcRd:"; sample%(0)
```

The next line requests 10 readings from channel 0 at a gain of ×1, using immediate triggering at 1 kHz.

```
ret& = VBdaqAdcRdN&(handle&,0, buf%(), 10, DatsImmediate&, 0, 1000!,
   DgainX1&, DafAnalog&+DafUnipolar&)
Print "Results of AdcRdN: ";
For x& = 0 To 9
    Print Format$ "####  "; buf%(x&);
Next x&
```
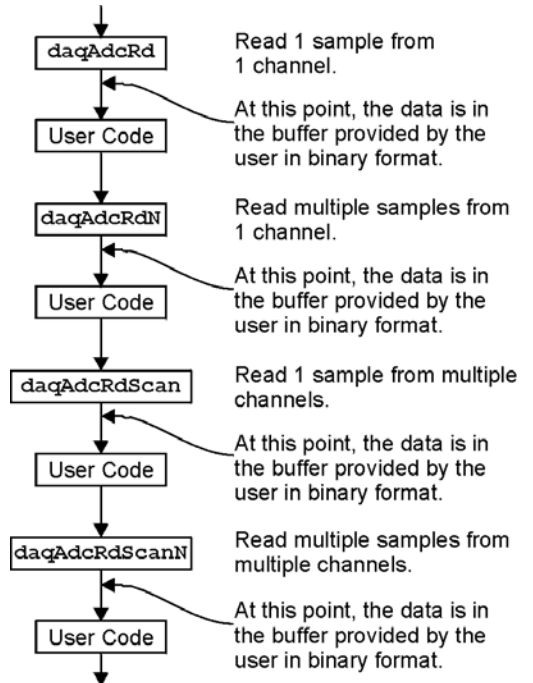
The program will then collect one sample of channels 0 through 7 using the `VBdaqAdcRdScan` function.

```
ret& = VBdaqAdcRdScan&(handle&,0, 7, buf%(), DgainX1&,
 DafAnalog&+DafUnipolar&)
Print "Results of AdcRdscan:"
For x& = 0 To 7
    Print Format$"& #  &  ####"; "Channel:"; buf%(x); "Data:"; buf%(x)
Next x&: Print
```

Counted Acquisitions Using Linear Buffers

This section sets up an acquisition that collects post-trigger A/D scans. This particular example demonstrates the setting up and collection of a fixed-length A/D acquisition in a linear buffer.

First, the acquisition is configured by setting up the channel scan group configuration, the acquisition frequency, the acquisition trigger and the acquisition mode. When configured, the acquisition is then armed by calling the `daqAdcArm` function.

At this point, the Daq device trigger is armed and A/D acquisition will begin upon trigger detection. If the trigger source has been configured to be `DatsImmediate&`, A/D data collection will begin immediately.

This example will retrieve 10 samples from channels 0 through 7, triggered immediately with a 1000 Hz sampling frequency and unity gain. Functions used include:

Function	Description
daqAdcSetMux	Define a channel scan group.
daqAdcSetFreq	Set the sampling frequency.
daqAdcSetAcq	Configure a counted acquisition for 10 post-trigger scans.
daqAdcSetTrig	Set the trigger event to be immediate.
daqAdcTransferSetBuffer	Configure an ADC transfer data buffer to be 10 scans long and terminate once the end of the buffer is reached.
daqAdcTransferStart	Initiate a transfer into the configured buffer.
daqAdcArm	Arm the acquisition. Since trigger source is immediate, the acquisition begins now.
daqWaitForEvent	Wait for the acquisition to complete.
User Code	Process the data.

- `VBdaqAdcSetMux&(handle&, startChan&, endChan&, gain&, flags&)`
- `VBdaqAdcSetFreq&(handle&,freq!)`
- `VBdaqAdcSetTrig&(handle&, triggerSource&, rising&, level%, hysteresis%,channel&)`
- `VBdaqAdcSetAcq&(handle&,mode&,preTrigCount&,postTrigCount&)`
- `VBdaqAdcTransferSetBuffer&(handle&,buf%(), scanCount&, transferMask&)`
- `VBdaqAdcTransferStart&(handle&)`
- `VBdaqAdcWaitForEvent&(handle&,daqEvent&)`

This program will initialize the Daq device hardware, then take readings from the analog input channels in the base unit (not the expansion cards). The functions used in this program are of a lower level than those used in the previous section and provide more flexibility.

```
Dim buf%(80), handle&, ret&, flags&
```

The following function defines the channel scan group. The function specifies a channel scan group from channel 0 through 7 with all channels being analog unipolar input channels with a gain of ×1. Specifying this configuration uses `DgainX1` in the gain parameter and the `DafAnalog` and the `DafUnipolar` values in the `flags` parameter. The `flags` parameter is a bit-mask field in which each bit specifies the characteristics of the specified channel(s). In this case, the `DafAnalog` and the `DafUnipolar` values are added together to form the appropriate bit mask for the specified `flags` parameter.

```
ret& = VBdaqAdcSetMux&(handle&,0, 7, DgainX1&, DafAnalog&+DafUnipolar&)
```
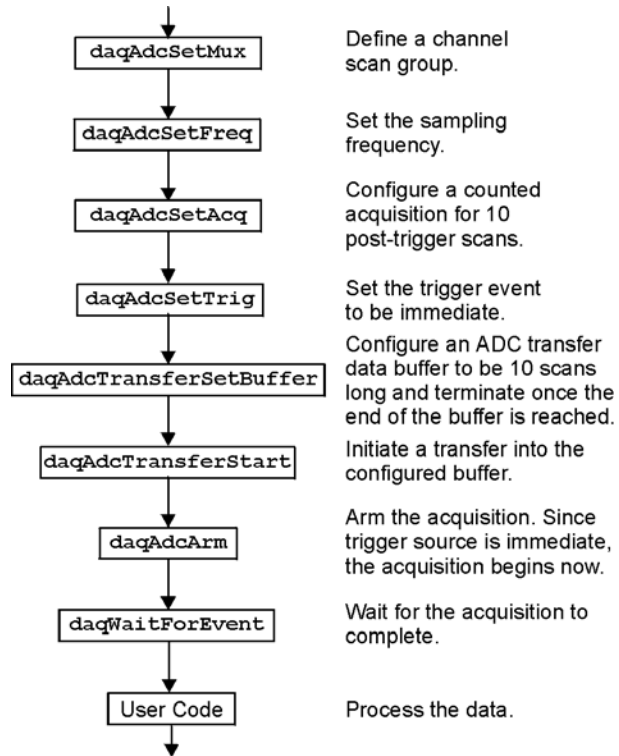
Next, set the internal sample rate to 1 kHz.

```
ret& = VBdaqAdcSetFreq&(handle&,1000!)
```

The acquisition mode needs to be configured to be fixed length acquisition with no pre-trigger scan data and 10 scans of post-trigger scan data. The mode is set to **DaamNShot&**, which will configure the acquisition as a fixed-length acquisition that will terminate automatically upon the satisfaction of the post-trigger count of 10.

```
ret& = VBdaqAdcSetAcq&(handle&,DaamNShot&, 0, 10)
```

The acquisition begins upon detection of the trigger event. The trigger event is configured with **daqAdcSetTrig**. The next line defines the trigger event (to be the immediate trigger source) that will start the acquisition immediately. The variable **DatsImmediate&** is a constant defined in DaqX.bas. Since the trigger source is configured as immediate, the other trigger parameters are not needed.

```
ret& = VBdaqAdcSetTrig&(handle&,DatsImmediate&, 0, 0, 0, 0)
```

A buffer now is configured to hold the A/D data to be acquired. Since this is to be a fixed length transfer to a linear buffer, the buffer cycle mode should be turned off with **DatmCycleOff&**. For efficiency, block update mode is specified with **DatmUpdateBlock&**. The buffer size is set to 10 scans.

Note: the user-defined buffer must have been allocated with sufficient storage to hold the entire transfer prior to invoking the following line.

```
ret& = VBdaqAdcTransferSetBuffer&(handle&,buf%(), 10,
  DatmUpDateBlock&+DatmCycleOff&)
```

With all acquisition parameters being configured, the acquisition can now be armed. Once armed, the acquisition will begin immediately upon detection of the trigger event. As in the case of the immediate trigger, the acquisition will begin immediately upon execution of the **daqAdcArm** function.

```
ret& = VBdaqAdcArm&(handle&)
```

After setting up and arming the acquisition, the data is immediately ready to be collected. Had the trigger source been anything other than immediate, the data would only be ready after the trigger had been satisfied. The following line initiates an A/D transfer from the Daq device to the defined user buffer.

```
ret& = VBdaqAdcTransferStart&(handle&)
```

Wait for the transfer to complete in its entirety, then proceed with normal application processing.
 This can be accomplished with the **daqWaitForEvent** command. The **daqWaitForEvent** allows the application processing to become blocked until the specified event has occurred. **DteAdcDone**, indicates that the event to wait for is the completion of the transfer.

```
ret& = VBdaqWaitForEvent(handle&,DteAdcDone&)
```

At this point, the transfer is complete; all data from the acquisition is available for further processing.

```
Print "Results of Transfer"
For i& = 0 To 10
    Print "Scan "; Format$(Str$(i& + 1), "00"); " -->";
    For k& = k& To k& + 7
        Print Format$(IntToUint&(buf%(k&)), "00000"); "  ";
    Next k&
    Print
Next i&
```

Indefinite Acquisition, Direct-To-Disk Using Circular Buffers

This program demonstrates the use of circular buffers in cycle mode to collect analog input data directly to disk. In cycle mode, this data transfer can continue indefinitely. When the transfer reaches the end of the physical data array, it will reset its array pointer back to the beginning of the array and continue writing data to it. Thus, the allocated buffer can be used repeatedly like a FIFO buffer.

The API has built-in direct-to-disk functionality. Therefore, very little needs to be done by the application to configure direct-to-disk operations.

First, the acquisition is configured by setting up the channel scan group configuration, the acquisition frequency, the acquisition trigger and the acquisition mode. Once configured, the transfer to disk is set up and the acquisition is then armed by calling the **daqAdcArm** function.

At this point, the Daq device trigger is armed and A/D acquisition to disk will begin immediately upon trigger detection.

This example will retrieve an indefinite amount of scans for channels 0 through 7, triggered via software with a 3000 Hz sampling frequency and unity gain. Functions used include:

Flowchart Step	Description
daqAdcSetScan	Configure a scan group of channels.
daqAdcSetFreq	Set the sampling frequency.
daqAdcSetAcq	Configure the acquisistion to be indefinite post-trigger.
daqAdcSetTrig	Configure the trigger event to be software trigger.
daqAdcTransferSetBuffer	Configure a circular acquisition buffer 10,000 scans in length.
daqAdcSetDiskFile	Open the disk file and make it ready to receive A/D data.
daqAdcArm	Arm the acquisition.
daqAdcTransferStart	Initiate data transfer to disk (no data will transfer until trigger event occurs).
daqAdcSoftTrig	Trigger the acquisition.
daqAdcWaitForEvent	Wait for data to become available.
daqAdcTransferGetStat	Check status of transfer.
User Terminator	User code to determine if transfer should stop. (No → loop back; Yes → continue)
daqAdcDisarm	Transfer is complete; disarm the acquisition.

- **VBdaqAdcSetScan&(handle&, startChan&, endChan&, gain&, flags&)**
- **VBdaqAdcSetFreq&(handle&,freq!)**
- **VBdaqAdcSetTrig&(handle&, triggerSource&, rising&, level%, hysteresis%,channel&)**
- **VBdaqAdcSetAcq&(handle&,mode&,preTrigCount&,postTrigCount&)**
- **VBdaqAdcTransferSetBuffer&(handle&,buf%(), scanCount&, transferMask&)**
- **VBdaqAdcTransferStart&(handle&)**
- **VBdaqAdcTransferGetStat&(handle&,status&,retCount&)**
- **VBdaqAdcWaitForEvent&(handle&,daqEvent&)**
- **VBdaqAdcSetDiskFile&(handle&,filename$,openMode&,preWrite&)**

This program will initialize the Daq device hardware, then take readings from the analog input channels in the base unit (not the expansion cards) and store them to disk automatically. The following lines demonstrate channel scan group configuration using the **daqAdcSetScan** command.

Note: flags may be channel-specific.
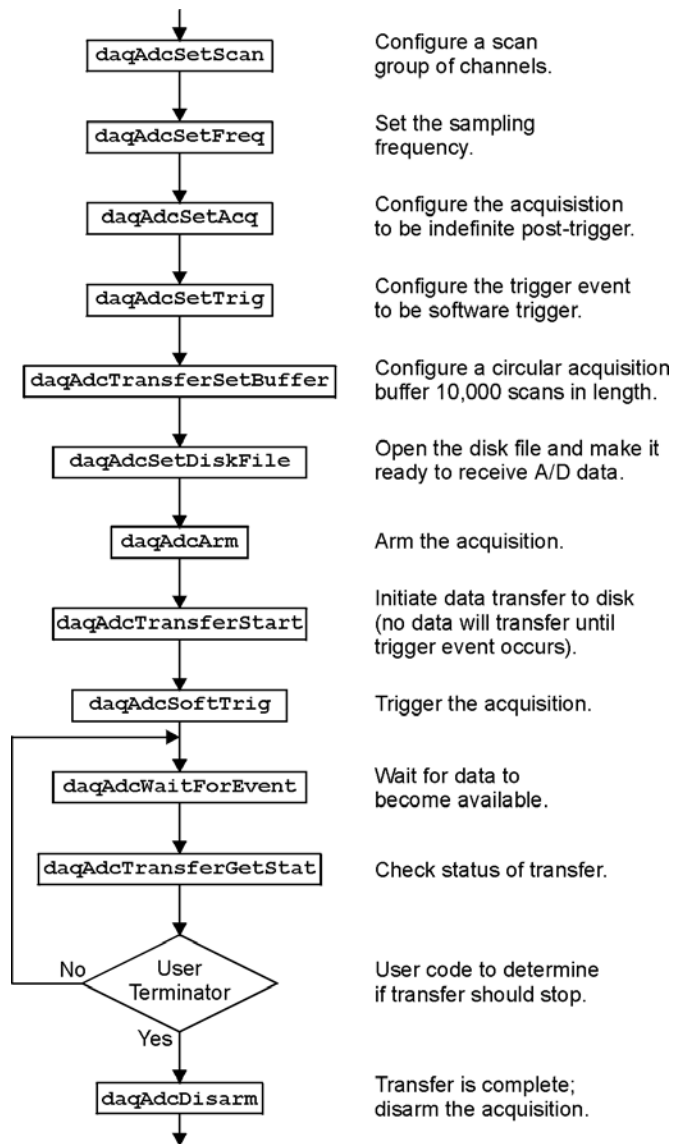
```
Dim handle&, ret&, channels&(8), gains&(8) flags&(8)
Dim buf%(80000), active&, count&
Dim bufsize& = 10000              ' In scans
' Define arrays of channels and gains : 0-7 , unity gain
For x& = 0 To 7
    channels&(x&) = x&
    gains&(x&) = DgainX1&
  flags&(x&) = DafAnalog& + DafSingleEnded& + DafUnipolar&
Next x&
' Load scan sequence FIFO
ret& = VBdaqAdcSetScan&(handle&,channels&(), gains&(), flags&(), 8)
```

Next, set the internal sample rate to 3 kHz.

```
ret& = VBdaqAdcSetFreq&(handle&,3000!)
```

The acquisition mode needs to be configured to be fixed-length acquisition with no pre-trigger scan data and 10 scans of post-trigger scan data. The mode is set to **DaamInfinitePost&**, which will configure the acquisition as having indefinite length and, as such, will be terminated by the application. In this mode, the pre- and post-trigger count values are ignored.

```
ret& = VBdaqAdcSetAcq&(handle&,DaamInfinitePost&, 0, 0)
```

The acquisition begins upon detection of the trigger event. The trigger event is configured with **daqAdcSetTrig**. The next line defines the trigger event to be the immediate trigger source which will start the acquisition immediately. The variable **DatsSoftware&** is a constant defined in DaqX.bas. Since the trigger source is configured as immediate, the other trigger parameters are not needed.

```
ret& = VBdaqAdcSetTrig&(handle&,DatsSoftware&, 0, 0, 0, 0)
```

A buffer now is configured to hold the A/D data to be acquired. This buffer is necessary to hold incoming A/D data while it is being prepared for disk I/O. Since this is to be an indefinite-length transfer to a circular buffer, the buffer cycle mode should be turned on with **DatmCycleOn&**. For efficiency, block update mode is specified with **DatmUpdateBlock&**. The buffer size is set to 10,000 scans. The buffer size indicates only the size of the circular buffer, not the total number of scans to be taken.

```
ret& = VBdaqAdcTransferSetBuffer&(handle&,buf%(), bufsize&,
  DatmUpDateBlock&+DatmCycleOn&)
```

Now the destination disk file is configured and opened. For this example, the disk file is a new file to be created by the driver. After the following line has been executed, the specified file will be opened and ready to accept data.

```
ret& = VBdaqAdcSetDiskFile&(handle&,"c:dasqdata.bin", DaomCreateFile&, 0)
```

With all acquisition parameters being configured and the acquisition transfer to disk configured, the acquisition can now be armed. Once armed, the acquisition will begin immediately upon detection of the trigger event. As in the case of the immediate trigger, the acquisition will begin immediately upon execution of the daqAdcArm function.

```
ret& = VBdaqAdcArm&(handle&)
```

After setting up and arming the acquisition, data collection will begin upon satisfaction of the trigger event. Since the trigger source is software, the trigger event will not take place until the application issues the software trigger event. To prepare for the trigger event, the following line initiates an A/D transfer from the Daq device to the defined user buffer and, subsequently, to the specified disk file. No data is transferred at this point, however.

```
ret& = VBdaqAdcTransferStart&(handle&)
```

The transfer has been initiated, but no data will be transferred until the trigger event occurs. The following line will signal the software trigger event to the driver; then A/D input data will be transferred to the specified disk file as it is being collected.

```
ret& = VBdaqAdcSoftTrig&(handle&)
```

Both the acquisition and the transfer are now currently active. The transfer to disk will continue indefinitely until terminated by the application. The application can monitor the transfer process with the following lines of code:

```
acqTermination& = 0
Do
    ' Wait here for new data to arrive
    ret& = VBdaqWaitForEvent(handle&,DteAdcData&)

    ' New data has been transferred - Check status
    ret& = VBdaqAdcTransferGetStat&(handle&,active&,retCount&)

    ' Code may be placed here which will process the buffered data or
    ' perform other application activities
    '
    ' At some point the application needs to determine the event on which
    ' the direct-to-disk acquisition is to be halted and set the
    ' acqTermination flag.

Loop While acqTermination& = 0
```

At this point the application is ready to terminate the acquisition to disk. The following line will terminate the acquisition to disk and will close the disk file.

```
ret& = VBdaqAdcDisarm&(handle&)
```

The acquisition as well as the data transfer has been stopped. We should check status one more time to get the total number of scans actually transferred to disk.

```
ret& = VBdaqAdcTransferGetStat(handle&,active&,retCount&)
```

The specified disk file is now available. The `retCount&` parameter will indicate the total number of scans transferred to disk.

Analog Output

The program DACEX1.BAS shows how to output analog voltages on analog output channels 0 and 1. These commands only have to be issued one time unless a related parameter is explicity changed. The output voltages will be sustained. This example demonstrates the use of the two digital-to-analog converters (values used assume bipolar mode). Functions used include:

- **VBdaqDacSetOutputMode&(handle&, DddtLocal&, 0, DdomVoltage&)**
- **VBdaqDacWt&(handle&, deviceType&, chan&, dataVal%)**
- **VBdaqDacWtMany&(handle, deviceTypes&(),chans&(), dataVals&())**

daqDacSetOutputMode — Set output mode to voltage.

daqDacWt — Output voltage on specific channel.

User Code

daqDacSetOutputMode — Set on both D/A channels.

daqDacWtMany — Output voltage on both D/A channels.

User Code
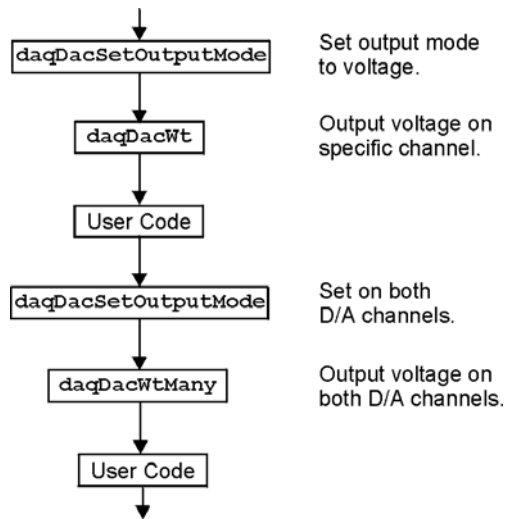
Assuming the voltage reference is connected to the internal default of 5 V, the next function will set channel 0 to an output voltage of 5 V. The values are set for a digital-to-analog converter with 16 bit resolution; 65535 represents full-scale. Channel 1 is equal to 0.

```
ret& = VBdaqDacSetOutputMode&(handle&, DddtLocal&, 0, DdomVoltage&)
ret& = VBdaqDacWt&(handle&, DddtLocal, 0, 65535)
```

The **daqDacWtMany** writes to both analog outputs simultaneously. The following lines sets channel 0 to 5 V and channel 1 to 2.5 V. At full-scale, a digital value of 65535 corresponds to 5 V; a digital value of 49152 corresponds to ½ of 5 V.

```
Dim deviceTypes&(1)
Dim chans&(1)
Dim dataVals%(1)
The VBdaqSetOutputMode puts the channel in a voltage mode.
ret& = VBdaqSetOutputMode&(handle&, DddtLocal&, 0, DdomVoltage&)
ret& = VBdaqSetOutputMode&(handle&, DddtLocal&, 1, DdomVoltage&)
deviceTypes&(0) = DddtLocal&
deviceTypes&(1) = DddtLocal&
chans&(0) = 0
chans&(1) = 1
dataVals&(0) = 65535
dataVals&(1) = 49152
ret& = VBdaqDacWtMany&(handle&, deviceTypes&(), chans&(), dataVals&(),2)
```

The following sets the outputs to 0 V.

```
Dim deviceTypes&(1)
Dim chans&(1)
Dim dataVals%(1)
deviceTypes&(0) = DddtLocal&
deviceTypes&(1) = DddtLocal&
chans&(0) = 0
chans&(1) = 1
dataVals&(0) = 32768
dataVals&(1) = 32768
ret& = VBdaqDacSetOutputMode&(handle&, DddtLocal&, 0, DdomVoltage&)
ret& = VBdaqDacSetOutputMode&(handle&, DddtLocal&, 1, DdomVoltage&)
ret& = VBdaqDacWtMany&(handle&, deviceTypes&(), chans&(), dataVals&(),2)
```

Generating DAC FIFO Waveforms

This program demonstrates the use of the DAC FIFO to generate waveforms. The DAC is configured for output on both channels, and the user waveform is constructed. Output begins after the waveform is assigned to a channel. At this point, the program continues while the waveforms are generated.

The following example shows how to generate a pre-defined waveform using these functions:

```
daqDacSetOutputMode
```
Configure DACs; use defined constant `DdomStaticWave` for both channels 0 and 1.

```
Set trigger frequency,
etc, for both channels
```

```
daqDacSetPredefWave
```
Specify predefined waveform for channel 1; use defined constant `Pdwtsine` for sine wave.

```
daqDacWaveArm
```
Start waveforms.

- **VBdaqDacWaveSetTrig&(handle&, deviceType&, chan&, triggerSource&, rising%)**
- **VBdaqDacWaveSetClockSource&(handle&, deviceType&, chan&, clockSource&)**
- **VBdaqDacWaveSetFreq&(handle&, deviceType&, chan&, freq!)**
- **VBdaqDacWaveSetMode&(handle&, deviceType&, chan&, mode&, updateCount&)**
- **VBdaqDacWaveSetBuffer&(handle&, deviceType&, chan&, buf%(), scanCount&, transferMask&)**
- **VBdaqDacWaveSetPredefWave&(handle&, deviceType&, chan&, waveType&, amplitude&, offset&, dutyCycle&, phaseShift&)**
- **VBdaqDacWaveArm&(ByVal handle&, ByVal deviceType&)**

When using the pre-defined waveform generation, program the waveform parameters common to both channels. The double star (**) indicates the value must be the same on both channels of a DaqBoard.

```
For chan = 0 To 1 Step 1
' set the output mode to static waveform
ret& = VBdaqDacSetOutputMode&(handle&, DddtLocal&, chan&, DdomStaticWave&)

' The trigger source must be set to immediate for static waveform.**
err& = VBdaqDacWaveSetTrig&(handle&, DddtLocal&, chan&, DdtsImmediate&, 1)

' set the internal dac clock
ret& = VBdaqDacWaveSetClockSource&(handle&, DddtLocal&, chan&,
 DdcsDacClock&)

' the frequency of the internal clock. **
ret& = VBdaqDacWaveSetFreq&(handle&, DddtLocal&, chan&, 10!)

' must be infinite for static mode
ret& = VBdaqDacWaveSetMode&(handle&, DddtLocal&, chan&, DdwmInfinite&, 0)
Next chan

' buffer cylce on, retransmit mode. **
' update count is the buffer length. **
ret& = VBdaqDacWaveSetBuffer&(handle&, DddtLocal&, chan&, buf0%(),
 updateCount&, DdtmCycleOn&)

' set the buffer for channel 1
ret& = VBdaqDacWaveSetBuffer&(handle&, DddtLocal&, chan&, buf1%(),
 updateCount&, DdtmCycleOn&)

' program the waveform parameters specific to dac channel 0
ret& = VBdaqDacWaveSetPredefWave&(handle&, DddtLocal&, 0, DdwtTriangle&,
 32768, 32768, 90, 0)

' program the waveform parameters specific to dac channel 1
ret& = VBdaqDacWaveSetPredefWave&(handle&, DddtLocal&, 1, DdwtSquare&,
 32768, 32768, 40, 0)

' buffer must be configured before the arm command is called. All channels
' will be armed.
ret& = VBdaqDacWaveArm(handle&, DddtLocal&)
```
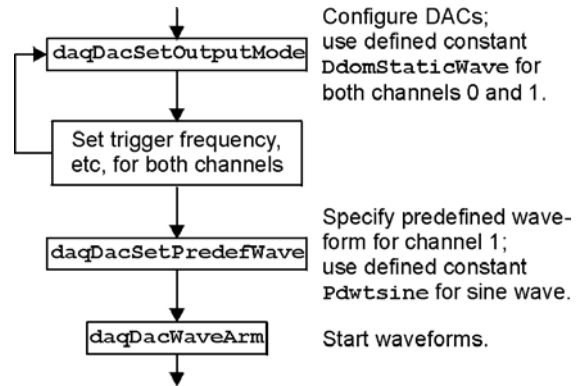
Digital I/O on P2

This program demonstrates the functions controlling digital I/O on the P2 connector of the device.   First, the 3 digital ports on the 8255 are configured as input, output, or both in the case of port C; then, appropriate I/O commands are issued. Functions used include:

- `VBdaqIOReadBit&(handle&, devType&, devPort&, whichDevice&, whichExpPort&, bitNum&, bitValue&)`
- `VBdaqIORead&(handle&, devType&, devPort&, whichDevice&, whichExpPort&, value&)`
- `VBdaqIOWriteBit&(handle&, devType&, devPort&, whichDevice&, whichExpPort&, bitNum&, bitValue&)`
- `VBdaqIOWrite&(handle&, devType&, devPort&, whichDevice&, whichExpPort&, value&)`
- `VBdaqIOGet8255Conf&(handle&, portA&, portB&, portCHigh&, portCLow&, config&)`

daqIOGet8255Conf — Configure mode of ports A, B, & C on 8255 chip.

daqIOWrite — Set configuration.

daqIOWrite — Write to port A (use defined constant DdpLocalA). Read from port B (use defined constant DdpLocalB).

daqIORead —

daqIOWriteBit — Write to high nibble of port C (use defined constant DdpLocalCHigh) (repeat for each bit).

daqIORead — Read from low nibble of port C (use defined constant DdpLocalCLow) (repeat for each bit).
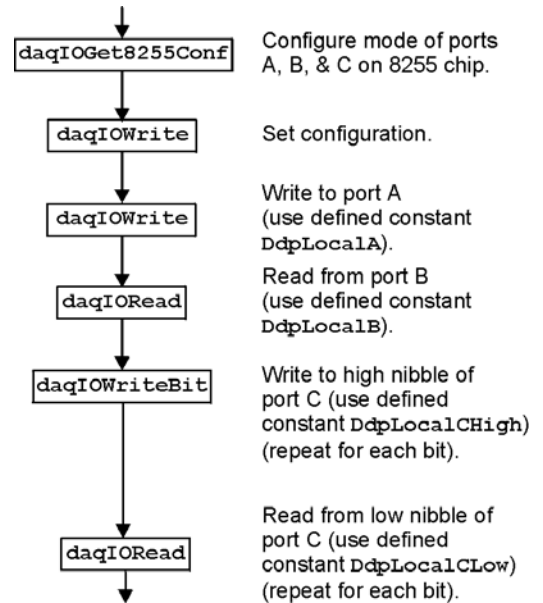
```
Dim config&, byteVal&, bitVal&, x%
Dim buf(10) As Byte, active&, retCount&
handle& = VBdaqOpen&("DaqBook0")
ret& = VBdaqSetErrorHandler&(handle&, 100)
On Error GoTo ErrorHandlerDIG1
ret& = VBdaqIOGet8255Conf&(handle&, 0, 1, 0, 1, config&)
```

The function daqIOGet8255Conf returns the appropriate configuration value to use in daqIOWrite. As shown above, the handle of the opened Daq device is the first parameter passed. The second, third, fourth, and fifth parameters respectively indicate: the 8255 port A value, the port B value, the high-nibble value of port C, and the low-nibble value of port C. The values for the parameters passed in the call shown above will return the configuration value (port A = OUTPUT, port B = INPUT, port C / high nibble = output, port C / low nibble = INPUT) in the config& parameter, which matches the current configuration of the 8255.

The daqIOWrite function writes the obtained configuration value to the selected port.

```
ret& = VBdaqIOWrite&(handle&, DiodtLocal8255&, Diodp8255IR&, 0, 0,_ config&)
```

Write hex 55 to port A on the Daq device's base unit.

```
ret& = VBdaqIOWrite&(handle&, DiodtLocal8255&, Diodp8255A&, 0,  0,_ &H55)
```

Read port B and put the value into the variable byteVal%.

```
 ret& = VBdaqIORead&(handle&, DiodtLocal8255&, Diodp8255B&, 0, 0,_ byteVal&)
Print "The value on digital port B : &H"; Hex$(byteVal&): Print
```

The following lines write to the high nibble of port C.

```
ret& = VBdaqIOWriteBit&(handle&,DiodtLocal8255&,Diodp8255CHigh&,0,_ 0,0,1)
ret& = VBdaqIOWriteBit&(handle&,DiodtLocal8255&,Diodp8255CHigh&,0,_ 0,1, 0)
ret& = VBdaqIOWriteBit&(handle&,DiodtLocal8255&,Diodp8255CHigh&,0,_ 0, 2, 1)
ret& = VBdaqIOWriteBit&(handle&,DiodtLocal8255&,Diodp8255CHigh&,0,_ 0, 3, 0)
Print "The high nibble of digital port C set to : 0101 (&H5)":  Print
```

The next lines read the low nibble of port C on the base unit.

```
      For x% = 0 To 3
ret& = VBdaqIOReadBit&(handle&, DiodtLocal8255&,_          Diodp8255CLow&,
  0, 0, x%, bitVal&)
Print "The value on bit "; x%; " of digital port C : &H";_  Hex$(bitVal&)
 Next x%
```

Using DBK Card Calibration Files

Software calibration functions are designed to adjust Daq device readings to compensate for gain and offset errors. Calibration constants are calculated at the factory by measuring the gain and offset errors of a card at each programmable gain setting. These constants are stored in a calibration text file that can be read by a program at runtime. This allows new boards to be configured for calibration by updating this calibration file rather than recompiling the program. Calibration constants and instructions are shipped with the related DBK boards. Programs like DaqView support this calibration and use the same constants.

The calibration operation removes static gain and offset errors that are inherent in the hardware. The calibration constants are measured at the factory and do not change during the execution of a program. These constants are different for each card and programmable-gain setting; they may even be different for each channel, depending on the design of the expansion card.

daqAdcSetScan	Configure scans.
daqAdcSetFreq	Set frequency.
daqAdcSetTrig	Define and arm trigger (use defined constant `DtsSoftware` for software trigger).
daqAdcSoftTrig	Software trigger.
daqAdcTransferSetBuffer	
daqAdcTransferStart	
daqWaitForEvent	Wait for the transfer to be complete.
daqAdcTransferGetStat	
daqCalSetupConvert	Perform calibration on readings at x1 gain.
daqCalSetupConvert	Perform calibration on readings at x2 gain.

Note: DBK19 is shipped with calibration constants. Other cards use on-board potentiometers to perform hardware calibration.

The calibration process has 3 steps:

- **Initialization** consists of reading the calibration file.
- **Setup** describes the characteristics of the data to be calibrated.
- **Conversion** does the actual calibration of the data.

```
[MAIN]
32760,32769
32801,32750
32740,32777
32810,32768

[EXP3]
32780,32779
32800,32756
32768,32780
32750,32742

[EXP5]
32752,32764
32783,32757
32749,32767
32777,32730
```

Function prototypes, return error codes, and parameter definitions are located in the DAQX.H header file for C (or similar files for other languages).

Cards that support the calibration functions are shipped with a diskette containing a calibration constants file. The name of the file will be the serial number of the card shipped with it. This file holds the calibration constants for each programmable-gain setting of that card. These constants should be copied to a calibration text file (DAQBOOK.CAL) located in the same directory as the program performing the calibration.

To set up the calibration file, perform the following steps:

1. Locate the diskette containing the calibration constants file.

2. Configure the card according to instructions found in the DBK documentation (included on your CD, or in hardcopy).

3. Edit the calibration file, DAQBOOK.CAL, using a text editor.

4. Add the card number information within brackets, as listed in the calibration file.

5. Add the calibration constants immediately after the card number. (These should be entered in the order given in the calibration file.)

6. Repeat steps 4 and 5 for each card.

7. Verify that no two cards are configured with the same card/channel number.

The table shows an example of a calibration file for configuring the main Daq device unit and two DBK19 cards connected to Daq device expansion channels 3 and 5.

The initialization function for reading in the calibration constants from the calibration text file is daqReadCalFile. The C language version of daqReadCalFile is similar to other languages and works as follows:

The filename with optional path information of the calibration file. If calfile is NULL or empty (""), the default calibration file DAQBOOK.CAL will be read. This function is usually called once at the beginning of a program and will read all the calibration constants from the specified file. If calibration constants for a specific channel number and gain setting are not contained in the file, ideal calibration constants will be used (essentially not calibrating that channel). If an error occurs while trying to open the calibration file, ideal calibration constants will be used for all channels and a non-zero error code will be returned by the **daqReadCalFile** function.

Once the calibration constants have been read from the cal file, they can be used by the **daqCalSetup** and **daqCalConvert** functions. The **daqCalSetup** function will configure the order and type of data to be calibrated. This function requires data to be from consecutive channels configured for the same gain, polarity, and channel type. The calibration can be configured to use only the gain calibration constant and not the offset constant. This allows the offset to be removed at runtime using the zero compensation functions described later in this section.

In this example, several Daq device channels will be read and calibrated. This example assumes the calibration file has been created according to the initializing calibration constants section of this chapter. Expansion cards can perform the same type of calibration if the calibration constants are available for the card and a specified channel number. First list the configuration:

Channel	Channel Type
0	Voltage1 @ X1 gain
1	Voltage2 @ X2 gain
2	Voltage3 @ X2 gain
3	Voltage4 @ X2 gain

Now specify the scan (the sequence of channel numbers and gains that are to be gathered as one burst of readings). In this example, all the channels at each gain will be read together (in consecutive order) to make the calibration easier.

Scan Position	Channel Type	Channel	Gain Code
0	Voltage1 @ X1 gain	0	DgainX1
1	Voltage2 @ X2 gain	1	DgainX2
2	Voltage3 @ X2 gain	2	DgainX2
3	Voltage4 @ X2 gain	3	DgainX2

Now configure the Daq device with this information, and read 5 scans of data:

```
Dim chans&(4), gains&(4), buf%(20)

handle& = VBdaqOpen&("DaqBook0")

' Set array of channels and gains
chans&(0) = 0
gains&(0) = DgainX1&
chans&(1) = 1
gains&(1) = DgainX2&
chans&(2) = 2
gains&(2) = DgainX2&
chans&(3) = 3
gains&(3) = DgainX2&

' Load scan sequence FIFO :
ret& = VBdaqAdcSetScan&(handle&, chans&(), gains&(), 4)

' Set Clock
ret& = VBdaqAdcFreq&(handle&, 10)

' Define and arm trigger :
ret& = VBdaqAdcSetTrig&(handle&, DtsSoftware&, 0, 0, 0, 0)

' Trigger
ret& = VBdaqAdcSoftTrig&(handle&)

' Read the data
' 5 indicates the number of scans
' single mode for scans less than 500
ret& = VBdaqAdcTransferSetBuffer&(handle&, buf%(), 5, DatmCycleOff& +
  DatmSingleMode&)

ret& = VBdaqAdcTransferStart&(handle&)

'specifies to wait for the transfer to be complete
ret& = VBdaqWaitForEvent&(handle&, DteAdcDone&)

ret& = VBdaqAdcTransferGetStat&(handle&, active&, retCount&)

' Print the first scan of unconverted data
PRINT "Before Calibration:"
PRINT "Channel 0 at x1 gain: "; buf%(0)
PRINT "Channel 1 at x2 gain: "; buf%(1)
PRINT "Channel 2 at x2 gain: "; buf%(2)
PRINT "Channel 3 at x2 gain: "; buf%(3)

'Perform zero compensation on readings sampled at x1 gain
ret& = VBdaqCalSetupConvert&(handle&, 4, 0, 1, 0, DgainX1&, 0, 1, 0, buf%(),
  5)

'Perform zero compensation on readings sampled at x2 gain
ret& = VBdaqCalSetupConvert&(handle&, 4, 1, 3, 0, DgainX2&, 1, 1, 0, buf%(),
  5)

' Print the first scan of converted data
PRINT "After Calibration:"
PRINT "Channel 0 at x1 gain: "; buf%(0)
PRINT "Channel 1 at x2 gain: "; buf%(1)
PRINT "Channel 2 at x2 gain: "; buf%(2)
PRINT "Channel 3 at x2 gain: "; buf%(3)
```

Zero Compensation

Zero compensation removes offset errors while a program is running. This is useful in systems where the offset of a channel may change due to temperature changes, long-term drift, or hardware calibration changes. Reading a shorted channel on the same card at the same gain as the desired channel removes the offset at run-time.

Note: Zero compensation is not available for all expansion cards. The DBK19 has channel 1 permanently shorted for zero compensation; other cards require a channel to be shorted manually.

The zero-compensation functions require a shorted channel and a number of other channels to be sampled from the same card at the same gain as the shorted channel. These functions will work with cards (such as the DBK12, DBK13, and DBK19) that have one analog path from the input to the A/D converter. Other cards do not support the zero compensation functions because they have offset errors unique to each channel. The DBK19 is designed with channel 1 already shorted for performing zero compensation.

Function	Description
daqAdcSetScan	Configure scans.
daqAdcSetTrig	Set trigger type (use defined constant **DtsImmediate** to trigger when armed).
daqAdcSetAcq	Set acquisition mode.
daqAdcSetClockSource	Configure the ADC clock source.
daqAdcSetFreq	Set the scan frequency.
daqAdcTransferSetBuffer	Configure user-allocated buffer to receive raw data.
daqAdcTransferStart	Begin the data transfer.
daqAdcArm	Arm the acquisition triggers (immediately if using **DtsImmediate** trigger).
daqWaitForEvent	Wait for transfer event **DteAdcDone** (acquisition complete).
daqZeroSetupConvert	Perform zero compensation on readings at x1 gain.
daqZeroSetupConvert	Perform zero compensation on readings at x2 gain.

The **daqZeroSetup** function configures the location of the shorted channel and the channels to be zeroed within a scan, the size of the scan, and the number of readings to zero compensate. (This function does not do the conversion.) A non-zero return value indicates an invalid parameter error.

In this example, several Daq device channels will be read using various gains and zero-compensated to remove any offset errors. This example assumes that channel 0 of the Daq device has been manually shorted. Expansion cards could perform the same type of zero compensation as this example by shorting a channel on the expansion card and specifying card channel numbers. First list the configuration:

Channel	Channel Type
0	Shorted Channel
1	Voltage1 @ X1 gain
2	Voltage2 @ X2 gain
3	Voltage3 @ X2 gain
4	Voltage4 @ X2 gain

Now specify the scan, the sequence of channel numbers, and gains that are to be gathered as one burst of readings. In this example, we will first read the shorted channel at each gain that we plan on using, in this case ×1 and ×2. All the channels at each gain will be read together to make the actual zero compensation easier.

Scan Position	Channel Type	Channel	Gain Code
0	Shorted Channel @ X1	0	DgainX1
1	Shorted Channel @ X2	0	DgainX2
2	Voltage1 @ X1 gain	1	DgainX1
3	Voltage2 @ X2 gain	2	DgainX2
4	Voltage3 @ X2 gain	3	DgainX2
5	Voltage4 @ X2 gain	4	DgainX2

```
Public Sub ZeroComp()
' Performs zero compensation on ADCs readings
Const ScanLength& = 6  'Total channels per scan
Const ScanCount& = 5    'Number of scans to acquire
Dim chan&(ScanLength), gain&(ScanLength)
Dim flag&(ScanLength), buf%(ScanLength * ScanCount)
Dim ret&, daqHandle&

daqHandle& = VBdaqOpen&("daqbook0")

' Channel zero must be shorted to ground
' Use DafClearLSNibble flag to clear 4 least significant
' bits when using 12-bit A/D converters
chan&(0) = 0: gain&(0) = DgainX1&: flag&(0) = DafBipolar& +
 DafClearLSNibble&
chan&(1) = 0: gain&(1) = DgainX2&: flag&(1) = DafBipolar& +
 DafClearLSNibble&
chan&(2) = 1: gain&(2) = DgainX1&: flag&(2) = DafBipolar& +
 DafClearLSNibble&
chan&(3) = 2: gain&(3) = DgainX2&: flag&(3) = DafBipolar& +
 DafClearLSNibble&
chan&(4) = 3: gain&(4) = DgainX2&: flag&(4) = DafBipolar& +
 DafClearLSNibble&
chan&(5) = 4: gain&(5) = DgainX2&: flag&(5) = DafBipolar& +
 DafClearLSNibble&
ret& = VBdaqAdcSetScan&(daqHandle&, chan&(), gain&(), flag&(), ScanLength)
ret& = VBdaqAdcSetTrig&(daqHandle&, DatsImmediate&, 1, 0, 0, 0)
ret& = VBdaqAdcSetAcq&(daqHandle&, DaamNShot&, 0, ScanCount)
ret& = VBdaqAdcSetClockSource&(daqHandle&, DacsAdcClock&)
ret& = VBdaqAdcSetFreq&(daqHandle&, 100!)ret& =
 VBdaqAdcTransferSetBuffer&(daqHandle&, buf%(), ScanCount, DatmCycleOff& +
 DatmUpdateSingle&)
ret& = VBdaqAdcTransferStart&(daqHandle&)
ret& = VBdaqAdcArm&(daqHandle&)
ret& = VBdaqWaitForEvent&(daqHandle&, DteAdcDone&)

' Print the first scan of unconverted data
Print "Channel zero shorted to ground"
Print "Channel 0 at X1 gain: "; IntToUint(buf%(0))
Print "Channel 0 at X2 gain: "; IntToUint(buf%(1))
Print
Print "Before zero compensation"
Print "Channel 1 at X1 gain: "; IntToUint(buf%(2))
Print "Channel 2 at X2 gain: "; IntToUint(buf%(3))
Print "Channel 3 at X2 gain: "; IntToUint(buf%(4))
Print "Channel 4 at X2 gain: "; IntToUint(buf%(5))
Print
' Perform zero compensation on readings sampled at x1 gain.
' 1 reading at position 2. Zero reading at position 0.
ret& = VBdaqZeroSetupConvert&(ScanLength, 0, 2, 1, buf%(), ScanCount)

' Perform zero compensation on readings sampled at x2 gain.
' 3 readings at position 3. Zero reading at position 1.
ret& = VBdaqZeroSetupConvert&(ScanLength, 1, 3, 3, buf%(), ScanCount)

' Print the first scan of converted data
Print "After zero compensation"
Print "Channel 1 at X1 gain: "; IntToUint(buf%(2))
Print "Channel 2 at X2 gain: "; IntToUint(buf%(3))
Print "Channel 3 at X2 gain: "; IntToUint(buf%(4))
Print "Channel 4 at X2 gain: "; IntToUint(buf%(5))
Print
' Close the device
ret& = VBdaqClose&(daqHandle&)
End Sub

Function IntToUint(intval As Integer) As Long
' Converts 16-bit signed integer to unsigned long integer
If 0 <= intval Then
```

```
        IntToUint = intval
    Else
        IntToUint = 65535 + CLng(intval) + 1
    End If
End Function
```

Linear Conversion

Several DBKs use conversions from A/D readings to corresponding values that are a linear (straight-line) relationship. (Non-linear relationships for RTDs and thermocouples require special conversion functions—refer to the Thermocouple and RTD Linearization section later in this chapter.) The linear conversion functions are built into the API.

Six parameters are used to specify a linear relationship: the A/D input range (minimum and maximum values), and the transducer input signal level and output voltage at two points in the range.

Three functions are used to perform linear conversions: daqCvtLinearSetup, daqCvtLinearConvert, and daqCvtLinearSetupConvert. These functions are defined in the following pages. After their definitions, parameter examples and a program example show how they work.
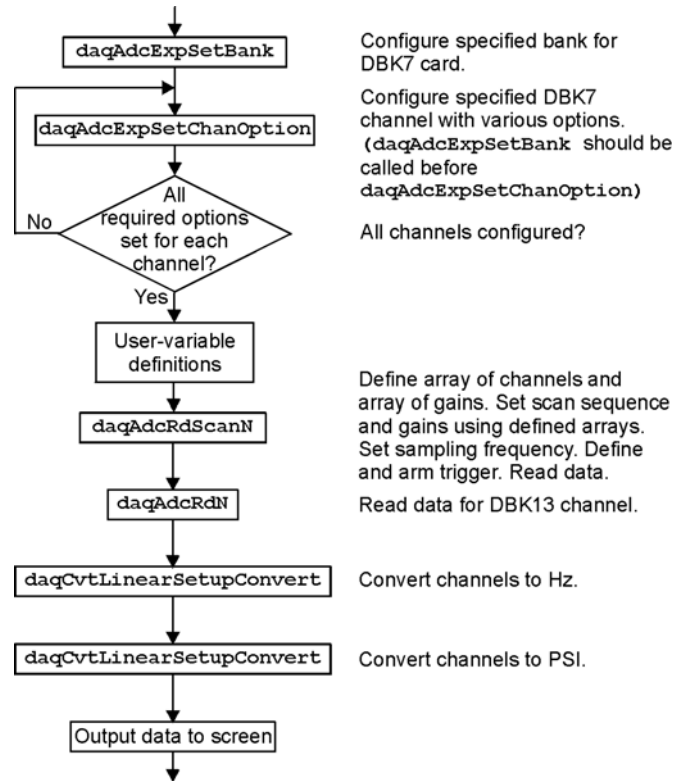
DBK7, programmed for 50 to 60 Hz:

Measurement	Signal	Voltage
1	50 Hz	-5 V
2	60 Hz	+5 V

The DBK7 output range is from -5 V to +5 V, and the Daq device must be configured for bipolar operation at a gain of ×1 for the DBK7 channels. Thus, the input range -5 V to +5 V corresponds to the **ADmin** and **ADmax** settings. When a DBK7 programmed for a 50 to 60 Hz range measures a 50 Hz input signal, it outputs -5 V. With a 60 Hz input signal, it outputs +5 V. Thus, **signal1** is 50, **voltage1** is -5, **signal2** is 60, and **voltage2** is 5.

Pressure-transducer:

Measurement	Signal	Voltage
1	0 psi	1 mV
2	1000 psi	4 mV

Assume that a pressure transducer outputs 1 to 4 mV to represent 0 to 1000 psi, and that a DBK13 with a gain of ×1000 is used with a Daq device in bipolar mode to measure the signal. In bipolar mode, at a gain of 1000, the analog signal input range is -5 to 5 mV and the output range from the DBK13 is -5 to 5 Volts. Thus, **ADmin** is -5.000, and **ADmax** is 5.000. A pressure of 0 psi generates an output of 1 mV, and 1000 psi generates 4 mV. Thus **signal1** is 0, **voltage1** is 1.000, **signal2** is 1000 and **voltage2** is 4.000.

This program uses the linear conversion functions to convert voltage readings from a DBK7 frequency-to-voltage card and a DBK13 voltage input card with a pressure transducer to actual frequencies (Hz) and pressures (psi).

```
                  Public Sub LinearConvert()
                  Dim buffer1%(80), buffer2%(80), flags&(3), hz!(20), psi!(10)
                  Dim ret&, handle&, chan&, x%

                  ' Initialize DaqBook
                  handle& = VBdaqOpen&("DaqBook0")

                  'Set Channel 16 to be a DBK7.  This will configure and auto-
                  'calibrate all channels on the DBK7 which includes channels
                  '16,17,18, and 19.  This step not required for a DBK13
                  ret& = VBdaqAdcExpSetBank(handle&, 16, DbankDbk7&)

                  'Set channel option common to all DBK7 channels.
                  'This step not required by a DBK13.
                  For chan& = 16 To 19
                     ret& = VBdaqAdcExpSetChanOption(handle&, chan&, DcotDbk7Slope&, 1)
                     ret& = VBdaqAdcExpSetChanOption(handle&, chan&, DcotDbk7DebounceTime&,
                   DcovDbk7DebounceNone&)
                     ret& = VBdaqAdcExpSetChanOption(handle&, chan&, DcotDbk7MinFreq&, 50!)
                     ret& = VBdaqAdcExpSetChanOption(handle&, chan&, DcotDbk7MaxFreq&, 60!)
                  Next chan&

                  'Channel configuration:
                  'DaqBook Channels 16, 17: DBK7  channels 0,1
                  'DaqBook Channel       32: DBK13 channel  0
                  'Configure the pacer clock, arm the trigger, and acquire 10
                  'scans. The gain setting of Dbk7X1 (X1 gain) will be applied
                  'to all channels.  The acquisition frequency is set to 100 Hz.
                  'All channels are unsigned - bipolar.
                  ret& = VBdaqAdcRdScanN&(handle&, 16, 17, buffer1%(), 10, DatsAdcClock&, 0,
                   0, 100!, Dbk7X1&, DafUnsigned& + DafBipolar&)

                  'Now do the same for the DBK13 channel, using gain Dbk13X1000
                  ret& = VBdaqAdcRdN(handle, 32, buffer2%(), 10, DatsAdcClock&, 0, 0, 100!,
                   Dbk13X1000&, DafUnsigned& + DafUnipolar&)

                  'Convert channels 16 and 17 to Hertz where -5 volts corresponds
                  'to 50 Hz and 5 volts corresponds to 60 Hz.
                  ret& = VBdaqCvtLinearSetupConvert(2, 0, 2, 50!, -5!, 60!, 5!, 1, buffer1%(),
                   10, hz!(), 20)

                  'Convert channel 32 to PSI where 1mV corresponds to 0 PSI and
                  '4 mV corresponds to 1000 PSI. DBK13 channel 0 has 1000x gain,
                  'so 1mV at Dbk13 input gives 1V output at DaqBook input.
                  ret& = VBdaqCvtLinearSetupConvert(1, 0, 1, 0!, 1!, 1000!, 4!, 1, buffer2%(),
                   10, psi!(), 10)

                  'Print results
                  Print "Results:"
                  For x = 0 To 9
                     Print Format(hz!(x * 2), "#0.00 Hz "); Format(hz!(x * 2 + 1), "#0.00 Hz
                   "); Format(psi(x), "0000.0 psi")
                  Next x

                  ret& = VBdaqClose(handle&)
            End Sub
```

Using Multiple Devices 3

Overview

This chapter applies to all devices that can be used with the DaqX API.   The purpose of this chapter is to show how devices can be used concurrently in either a synchronous or asynchronous fashion. Devices that have the ability to trigger from an external source or be clocked by an external source have the ability to be used in a synchronized system.

Asynchronous Operation

Asynchronous operation of devices is defined as the inability to synchronize the input of data between separate main unit devices. Data acquired with the devices is independent and the time-skew between devices is non-deterministic.

In the following sections, both synchronous and asynchronous operation will be covered. However, since synchronous operation requires more care in configuration most of the remaining material in the chapter will cover synchronous operation of multiple devices.

For either synchronous or asynchronous operation modes device configuration and data handling is very similar to the single device scenario. The DaqX API is a handle based API. This means that each device session has a handle assigned to it when it is opened. This device handle is then used to configure and acquire data from a device by referencing the device handle when calling the appropriate DaqX API functions.

Synchronous Operation

Synchronous operation of devices is defined as having the ability to synchronize the input of data between separate main unit devices. Data between devices is not time-skewed or the time-skew between devices is deterministic.

Asynchronous Operation of Multiple Devices

All DaqX compatible devices may be run concurrently in an asynchronous manner. When using the devices concurrently in an asynchronous manner the programming and data collection techniques are very similar to the single device Here, the triggering events, stop events and clocking of the input can be different between the devices.

As mentioned above, the DaqX API is a handle-based API. A handle-based API is an API which assigns a unique handle to each device in use. As in the single device scenario, each device needs to be opened using the **daqOpen** function. Each device should be opened using the alias name given to the device in the Daq Configuration Utility located in the Control Panel of WindowsNT/2000/XP operating system. As each device is opened a new, unique **handle** for each device is generated and returned by the **daqOpen** function.   The **handle** is a unique identifier for each device and should be used when referencing DaqX functions for the device.

Using this model, each device needs to be uniquely configured using the appropriate **handle** for the device. The acquisition of the data for each device also is managed independently for each device. The diagram below shows how the application should operate with two devices (designated as Device0 and Device1).
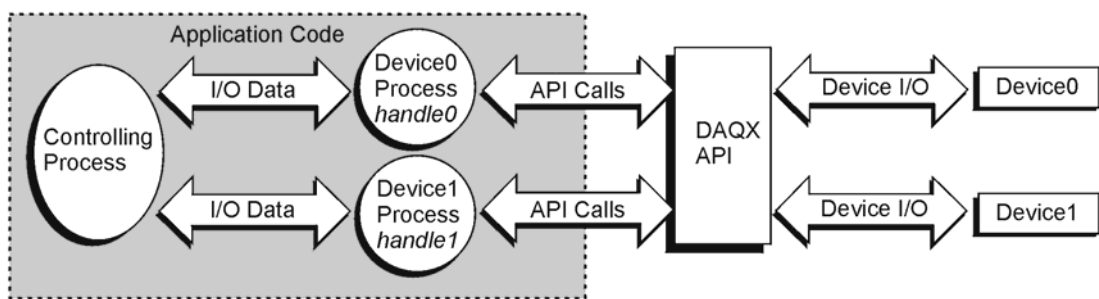
Using this model, each device needs to be opened and a device handle assigned. Here the Controlling Process would open each device session according to the device inventory list (this can be interrogated by inspecting the device inventory in the Daq Configuration utility located in the Control Panel of the operating system). The Control Process would then pass each device **handle** to the appropriate Device Process so that each device process can configure the device and acquire data from the device.

Note: While the above diagram does indicate the division of responsibilities within the application, the application may be organized in any fashion desirable. The individual processes need not be separate tasks or threads in order for the multi-device model to work well.

For discussion purposes, the responsibilities of the Controlling Process are as follows:

1. For each device, open the device session and retrieve the device **handle** (see **daqOpen**).

2. Pass each device **handle** to the appropriate Device Process so that the Device Process can configure the acquisition for its device.

3. Process any data returned from each Device Process and update any outputs required.

4. Optionally, write the input data for each device to disk.

Each device process is responsible for performing the following steps in order:

1. Configure the channel scan group for the device including expansion channels (see **daqAdcSetScan** for channel scan group configuration details).

2. Configure the acquisition clocking or rate to be used (see **daqAdcSetRate** and **daqAdcSetClockSource** for details on scan rate clocking configurations).

3. Configure the acquisition mode to set pre-trigger, post-trigger and update mode (see **daqAdcSetAcq** for details on configuring acquisition mode parameters).

4. Configure the means by which the acquisition should start and terminate by configuring the Trigger and Stop Events. (see **daqSetTriggerEvent** for more details on configuring the Trigger and Stop Events).

5. Configure the buffer model to be used for the device. (see **daqAdcTransferSetBuffer** for more details on configuration of the buffer model.

6. Initiate a transfer from the device (see **daqAdcTransferStart**) and arm the device to detect the trigger event (see **daqAdcArm**).

7. Monitor the status of the input data transfer and the acquisition (see **daqAdcTransferGetStat**) and optionally pass data back to the Controlling Process.

Since this section deals with asynchronous operation, each device may be independently programmed with respect to steps 1-7. When the devices are independently programmed each device may take on separate acquisition settings such as Trigger/Stop Events, Pre/Post-Trigger modes and buffer model settings. Independent device operation also implies that the data acquired from the different devices cannot be temporally correlated. As such, data from each device should be handled separately in either separate data buffers in memory or separate disk files. Sections to follow illustrate using multiple devices in a synchronized manner.

Synchronous Operation of Multiple Devices

The following sections describe methods for synchronizing input data with multiple DaqX compatible devices. Three synchronization methods will be discussed:

- o **Internal Clock Method**
- o **Master Clock Method**
- o **External Clock Method**

The synchronization method available depends upon the device, i.e., its ability to be externally triggered or externally clocked. The following table shows the capabilities of each device as well as the physical location of the external trigger and pacer input/output clocks.

Device	External TTL Trigger	External Acquisition Pacer Clock	Min. External Clock Interval	Acquisition Pacer Clock Output
WaveBook/512	Yes (pin 13 on DB25F)	No	NA	No
WaveBook/516	Yes (pin 13 on DB25F)	Yes (pin 20 on DB25F)	1us (1MHz)	No
DaqBooks	Yes (P1; pin 25)	No	NA	No
Daq PC-Card	Yes (P1; pin 25)	No	NA	No
DaqBoard(ISA)	Yes (P1; pin 25)	No	NA	No
TempBook	Yes	No	NA	No
DaqBoard/500 Series	Yes (pin 6 on SCSI ; TTLTRG on TB-100)	Yes (pin 2 on SCSI ; XAPCR on TB-100)	5us (200kHz)	Yes (pin 2 on SCSI ; XAPCR on TB-100)
DaqBoard/1000 Series	Yes (pin 6 on SCSI ; TTLTRG on TB-100)	Yes (pin 2 on SCSI ; XAPCR on TB-100)	5us (200kHz)	Yes (pin 2 on SCSI ; XAPCR on TB-100)
DaqBoard/2000 Series[1]	Yes (P1; pin 25) Note 2	Yes (P1; pin 20) Note 2	5us (200kHz)	Yes (P1; pin 20) Note 2
DaqBoard/2000c Series[1]	Yes (P1; pin 25) Note 2	Yes (P1; pin 20) Note 2	5us (200kHz)	Yes (P1; pin 20) Note 2
DaqBook/2000 Series	Yes (P1; pin 25) Note 3	Yes (P1; pin 20) Note 3	5us (200kHz)	Yes (P1; pin 20) Note 3
DaqScan/2000 Series	Yes (P1; pin 25)	Yes (P1; pin 20)	5us (200kHz)	Yes (P1; pin 20)
DaqLab/2000 Series	Yes (P1; pin 25)	Yes (P1; pin 20)	5us (200kHz)	Yes (P1; pin 20)
DaqBoard/3000 Series	Yes (pin 6 on SCSI; TTLTRG on TB-100)	Yes (pin 2 on SCSI ; XAPCR on TB-100)	1us (1MHz)	Yes (pin 2 on SCSI ; XAPCR on TB-100)

Synchronization Methods Available			
Device	Internal Clock Method	Master Clock Method	External Clock Method
WaveBook/512	Yes	No	No
WaveBook/516	Yes	Slave only	Yes
DaqBooks	Yes	No	No
Daq PC Card	Yes	No	No
DaqBoard(ISA)	Yes	No	No
TempBook	Yes	No	No
DaqBoard/500 Series	Yes	No	Yes
DaqBoard/1000 Series	Yes	Master or Slave	Yes
DaqBoard/2000 Series Note 1	Yes	Master or Slave	Yes
cPCI DaqBoard/2000c Series Note 1	Yes	Master or Slave	Yes
DaqBook/2000 Series	Yes	Master or Slave	Yes
DaqScan/2000 Series	Yes	Master or Slave	Yes
DaqLab/2000 Series	Yes	Master or Slave	Yes
DaqBoard/3000 Series	Yes	Master or Slave	Yes

Notes:

1. Excludes DaqBoard/2003 and cPCI DaqBoard/2003c.

2. The P1 connector for DaqBoard/2000 Series and DaqBoard/2000c Series boards is obtained by connecting a DBK200 Series option to the board's P4 connector.

3. DaqBook/2020 has a BNC connector for External TTL Trigger and a BNC connector for the external pacer clock. These BNC connectors are in addition to P1 connector pins 25 and 20.

4   Due to processing latencies, no software trigger sources should be used when attempting to do synchronous device acquisitions. When doing synchronous acquisitions the valid trigger modes are limited to External TLL, Analog Hardware, and Immediate trigger sources.
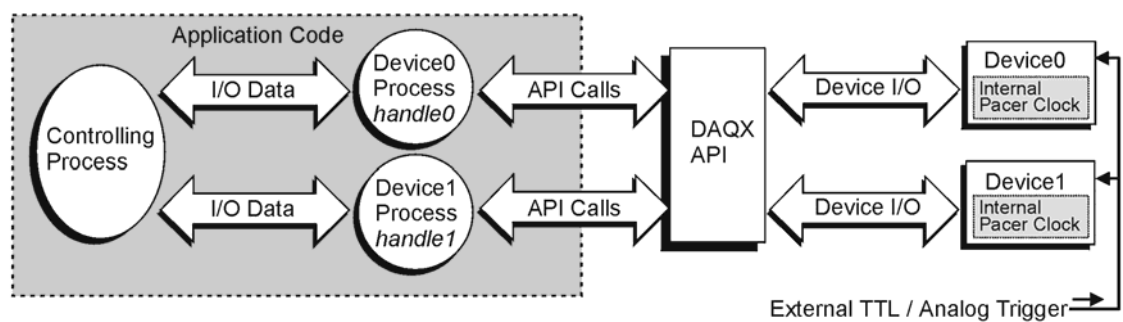
Internal Clock Method

This method synchronizes the devices by sharing the same external trigger event. The external trigger event can be either External TTL or Analog Hardware.   In this method, each device still runs independently on its own internal acquisition pacer clock but the acquisition on each device is initiated through the same external trigger event. Using this method it is important that all the devices internal pacer clocks are to the same rate (or at least evenly divisible by the highest rate) in order to ensure that the input data can be temporally correlated.

As mentioned above, the DaqX API is a handle-based API. A handle-based API is an API which assigns a unique handle to each device in use. As in the single device scenario, each device needs to be opened using the **daqOpen** function. Each device should be opened using the alias name given to the device in the Daq Configuration Utility located in the Control Panel of WindowsNT/2000/XP operating system. As each device is opened a new, unique **handle** for each device is generated and returned by the **daqOpen** function.   The **handle** is a unique identifier for each device and should be used when referencing DaqX functions for the device.

Using this model, each device needs to be configured using the appropriate **handle** for the device.   Some configuration parameters may differ but some parameters need to be configured specifically for this method. Using this model, each device needs to be uniquely configured using the appropriate **handle** for the device.   The acquisition of the data for each device also is managed independently for each device.

The diagram below shows how the application should operate with two devices.

Using this model, each device needs to be opened and a device handle assigned. Here the Controlling Process would open each device session according to the device inventory list (this can be interrogated by inspecting the device inventory in the Daq Configuration utility located in the Control Panel of the operating system). The Control Process would then pass each device **handle** to the appropriate Device Process.

Note: While the above diagram does indicate the division of responsibilities within the application, the application may be organized in any fashion desirable. The individual processes need not be separate tasks or threads in order for the multi-device model to work well.

For discussion purposes, the responsibilities of the Controlling Process are as follows:

1.   For each device, open the device session and retrieve the device **handle** (see **daqOpen**).

2.   Pass each device **handle** to the appropriate Device Process so that the Device Process can configure the acquisition for its device.

3.   Interleave the data from each Device Process and update any outputs as required.

4.   Optionally, write the input data from the devices and write the input data for each device to disk.

Each device process is responsible for performing the following steps (in order):

1. Configure the channel scan group for the device including expansion channels (see **daqAdcSetScan** for channel scan group configuration details).

2. Configure the acquisition for internal pacing and set the rate to be the same for all devices. (see **daqAdcSetRate** and **daqAdcSetClockSource** for details on scan rate clocking configurations).

3. Configure the acquisition mode to be a counted acquisition (**DaamNShot**) and set **no** pre-trigger and set the post-trigger count to be the same for all devices (see **daqAdcSetAcq** for details on configuring acquisition mode parameters).

4. Configure the trigger event to be External TTL (**DatsExternalTTL**) or Hardware Analog (**DatsHardwareAnalog**) and the set the sensitivity to be (rising, falling…) the same for all devices. Set the Stop Event to be counted (**DatsScanCount**). See **daqSetTriggerEvent** for more details on configuring the Trigger and Stop Events.

5. Configure the buffer model to be used for the device. (see **daqAdcTransferSetBuffer** for more details on configuration of the buffer model.

6. Initiate a transfer from the device (see **daqAdcTransferStart**) and arm the device to detect the trigger event (see **daqAdcArm**).

7. Monitor the status of the input data transfer and the acquisition (see **daqAdcTransferGetStat**) and optionally pass data back to the Controlling Process.

Since this section deals with synchronous operation, each device may be independently programmed with respect to steps 1, 5, 6 and 7. However, steps 2 through 4 need to be set as specified.

Note: In order for this method to work properly the actual acquisition scan rate settings need to be the same for all devices. Be sure to check that the actual scan rates set are equal for all devices (see **daqAdcSetRate** and **daqAdcGetFreq** for more information on retrieving the actual scan rate programmed for each device.

Master Clock Method

This method synchronizes the devices by setting all devices to run off a clock generated by a pre-selected Master device. Here the Master device is configured for internal clocking of the acquisition pacer clock. This pacer clock is then output by the Master device to the Slave devices that use the clock for acquisition pacing. In this method, each Slave device is configured for external acquisition pacing and set for immediate trigger. The Master device may be set for Analog Hardware, External TTL Level or immediate triggering. Once armed, the Master device will not drive the pacer clock output until the trigger event occurs. Likewise, the Slave units will not begin acquiring data until the external pacer clock pulses are generated (even though they were set to trigger immediately. When using this method it is important to ensure that the Master clock output is no faster than the largest minimum clock source for any of the Slave devices.

As mentioned above, the DaqX API is a handle-based API. A handle-based API is an API that assigns a unique handle to each device in use. As in the single device scenario, each device needs to be opened using the **daqOpen** function. Each device should be opened using the alias name given to the device in the Daq Configuration Utility located in the Control Panel of WindowsNT/2000/XP operating system. As each device is opened a new, unique **handle** for each device is generated and returned by the **daqOpen** function. The **handle** is a unique identifier for each device and should be used when referencing DaqX functions for the device.

Using this model, each device needs to be configured using the appropriate **handle** for the device. Some configuration parameters may differ but some parameters need to be configured specifically for this method.

Using this model, each device needs to be uniquely configured using the appropriate **handle** for the device. The acquisition of the data for each device also is managed independently for each device.

The diagram below shows how the application should operate.

Using this model, each device needs to be opened and a device handle assigned. Here the Controlling Process would open each device session according to the device inventory list (this can be interrogated by inspecting the device inventory in the Daq Configuration utility located in the Control Panel of the operating system). The Control Process would then pass each device **handle** to the appropriate Device Process.

Note: While the above diagram does indicate the division of responsibilities within the application, the application may be organized in any fashion desirable. The individual processes need not be separate tasks or threads in order for the multi-device model to work well.

Note: To ensure proper synchronization always configure the Slave devices before configuring the Master device.

For discussion purposes, the responsibilities of the Controlling Process are as follows:

1. For each device, open the device session and retrieve the device **handle** (see **daqOpen**).

2. Pass each device **handle** to the appropriate Device Process so that the Device Process can configure the acquisition for its device.

3. Interleave the data from each Device Process and update any outputs as required.

4. Optionally, write the input data from the devices and write the input data for each device to disk.

Each Slave Device Process should configure the Slave device(s) by performing the following steps in the order show:

1. Configure the channel scan group for the device including expansion channels (see **daqAdcSetScan** for channel scan group configuration details).

2. Configure the acquisition for external pacing (see **daqAdcSetClockSource** for details on scan rate clocking configurations).

3. Configure the acquisition mode to be a counted acquisition (**DaamNShot**) and set **no** pre-trigger and set the post-trigger count to be the same for all Slave devices (see **daqAdcSetAcq** for details on configuring acquisition mode parameters).

4. Configure the trigger event to be immediate (**DatsImmediate**). Set the Stop Event to be counted (**DatsScanCount**) (see **daqSetTriggerEvent** for more details on configuring the Trigger and Stop Events).

5. Configure the buffer model to be used for the device. (see **daqAdcTransferSetBuffer** for more details on configuration of the buffer model.

6. Initiate a transfer from the device (see **daqAdcTransferStart**) and arm the device to detect the trigger event (see **daqAdcArm**).

7. Monitor the status of the input data transfer and the acquisition (see **daqAdcTransferGetStat**) and optionally pass data back to the Controlling Process.

Since this method is a synchronous operation, each device may be independently programmed with respect to steps 1, 5, 6 and 7. However, steps 2 through 4 need to be set as specified.

The Master Device Process should configure the Master Device by performing the following steps (in order):

1. Configure the channel scan group for the device including expansion channels (see **daqAdcSetScan** for channel scan group configuration details).

2. Configure the acquisition for internal pacing. This setting will be the pacer clock setting for all devices in the system. (see **daqAdcSetRate** and **daqAdcSetClockSource** for details on scan rate clocking configurations).

3. Configure the acquisition mode to be a counted acquisition (**DaamNShot**) and set **no** pre-trigger and set the post-trigger count to be the same for all devices (see **daqAdcSetAcq** for details on configuring acquisition mode parameters).

4. Configure the trigger event to be Analog Hardware (**DatsHardwareAnalog**) or External TTL (**DatsExternalTTL**) if the acquisition is to begin on the detection of an external event. If it is desirable, however, to trigger the acquisition immediately then use an immediate trigger (**DatsImmediate**). Set the Stop Event to be counted (**DatsScanCount**). See **daqSetTriggerEvent** for more details on configuring the Trigger and Stop Events.

5. Configure the buffer model to be used for the device. (see **daqAdcTransferSetBuffer** for more details on configuration of the buffer model.

6. Initiate a transfer from the device (see **daqAdcTransferStart**) and arm the device to detect the trigger event (see **daqAdcArm**).

7. Monitor the status of the input data transfer and the acquisition (see **daqAdcTransferGetStat**) and optionally pass data back to the Controlling Process.

Since this method is a synchronous operation, each device may be independently programmed with respect to steps 1, 5, 6 and 7. However, steps 2 through 4 need to be set as specified.

External Clock Method

This method synchronizes the devices by sharing the same external clock source. In this method, each device will pace its acquisition on the same external TTL clock source. Here, the external clock source provided can be no faster than the largest of the minimum external clock interval on the system (see previous table).

As mentioned above, the DaqX API is a handle-based API. A handle-based API is an API which assigns a unique handle to each device in use. As in the single device scenario, each device needs to be opened using the **daqOpen** function. Each device should be opened using the alias name given to the device in the Daq Configuration Utility located in the Control Panel of WindowsNT/2000/XP operating system. As each device is opened a new, unique **handle** for each device is generated and returned by the **daqOpen** function. The **handle** is a unique identifier for each device and should be used when referencing DaqX functions for the device.

Using this model, each device needs to be configured using the appropriate **handle** for the device. Some configuration parameters may differ but some parameters need to be configured specifically for this method.

Using this model, each device needs to be uniquely configured using the appropriate **handle** for the device. The acquisition of the data for each device also is managed independently for each device.

The following diagram shows how the application should operate for two devices.

Using this model, each device needs to be opened and a device handle assigned. Here the Controlling Process would open each device session according to the device inventory list (this can be interrogated by inspecting the device inventory in the Daq Configuration utility located in the Control Panel of the operating system). The Control Process would then pass each device **handle** to the appropriate Device Process.

Note: While the above diagram does indicate the division of responsibilities within the application, the application may be organized in any fashion desirable. The individual processes need not be separate tasks or threads in order for the multi-device model to work well.

For discussion purposes, the responsibilities of the Controlling Process are as follows:

1. For each device, open the device session and retrieve the device **handle** (see **daqOpen**).

2. Pass each device **handle** to the appropriate Device Process so that the Device Process can configure the acquisition for its device.

3. Interleave the data from each Device Process and update any outputs as required .

4. Optionally, write the input data from the devices and write the input data for each device to disk.

Each Device Process should configure its device by performing the following steps (in order):

1. Configure the channel scan group for the device including expansion channels (see **daqAdcSetScan** for channel scan group configuration details).

2. Configure the acquisition for external pacing. (see **daqAdcSetClockSource** for details on scan rate clocking from an external source).

3. Configure the acquisition mode to be a counted acquisition (**DaamNShot**) and set **no** pre-trigger and set the post-trigger count to be the same for all devices (see **daqAdcSetAcq** for details on configuring acquisition mode parameters).

4. Configure the trigger event to be Analog Hardware (**DatsHardwareAnalog**) or External TTL(**DatsExternalTTL**) if the acquisition is to begin on the detection of an external event. If it is desirable, however, to trigger the acquisition immediately then use an immediate trigger (**DatsImmediate**).   Set the Stop Event to be counted (**DatsScanCoun**). See **daqSetTriggerEvent** for more details on configuring the Trigger and Stop Events.

5. Configure the buffer model to be used for the device. (see **daqAdcTransferSetBuffer** for more details on configuration of the buffer model.

6. Initiate a transfer from the device (see **daqAdcTransferStart**) and arm the device to detect the trigger event (see **daqAdcArm**).

7. Monitor the status of the input data transfer and the acquisition (see **daqAdcTransferGetStat**) and optionally pass data back to the Controlling Process.

Since this method is a synchronous operation, each device may be independently programmed with respect to steps 1, 5, 6 and 7. However, in steps 2 through 4, configurations must be as specified.

Overview

This chapter details the particular commands used to program Daq device applications. The first section of this chapter describes briefly the layout of the API Command entries, and is followed by two useful reference sections—first, a table describing the naming conventions of the predefined parameter values, and second, a section breifly describing the use of mask and flag values. After this, two tables of contents are offered: an alphabetical listing of the API commands, and a listing of API commands grouped into catagories of similar function. The API command entries begin after the table of contents, and are themselves arranged in alphabetical order. Following the commands, the API Error Code table is included, detailing the meaning of possible error return messages.

Note: The term "DaqBoard/2000 Series" refers to both the PCI and compact PCI (cPCI) versions of the boards, unless otherwise stated. For example: The use of "DaqBoard/2001" refers to both the DaqBoard/2001 and to the cPCI DaqBoard/2001c.

Command Information Layout

daqAPICommand {the API Command name, often referred to as a "Function"}

{related commands, if applicable} *Also See*: **daqAPICommand2, daqAPICommand3**

Format

```
daqAPICommand(parameter1, parameter2, parameter3)
```

Purpose

`daqAPICommand` does …. {this is a brief description of the command's purpose}.

Parameter Summary

Parameter	Type	Description
parameter1	DataTypeA	Briefly states the function of parameter1.
parameter2	DataTypeB	Briefly states the function of parameter2.
parameter3	DataTypeC	Briefly states the function of parameter3.

Parameter Values

parameter1: valid value range for parameter1

parameter2: valid value range for parameter2

parameter3: valid value range for parameter3

Parameter Type Definitions

Data Type Definitions - **DataTypeA**	
Description	**Description**
ParamValue1	Describes result of setting DataTypeA to parameter value 1
ParamValue2	Describes result of setting DataTypeA to parameter value 2
ParamValue3	Describes result of setting DataTypeA to parameter value 3

Returns

```
ReturnMessage1    Return Message definition
```

"Returns" lists possible return messages that could result from the use of the command. An error message is a return; and all Error information can be found in the Daq Error Table. Not that some commands have no returns.

Function Usage

The function usgage section provides additional information about the various parameters and their possible values. The information usually expands upon that which is presented in the Parameter Summary and Parameter Values tables. For clarity, this section may include sub-sections which discuss groups of parameters; such as parameters that deal with input and output, or parameters which must be used together to accomplish a result (such as a parameter which arms a counter, and another that configures the counter's starting value). The Function Usage subsections may also discuss broader Daq programming concepts.

Prototypes

The prototypes section contains examples in C/C++ or VisualBasic.   Examples follow.

C/C++

```
daqAPIcommand(DataTypeA parameter1, DataTypeB parameter2, DataTypeC
parameter3, TypeD parameter4, Type5 parameter5);
```

Visual BASIC

```
VBdaqAPICommand&(DataTypeA parameter1&, DataTypeB parameter2&,
DataTypeC parameter3&,)
```

Program References

This section gives a list of programs which use the function. Programs are separated by a comma and space.

C/C++ programs end in ".CPP"; for example: `DAQADCEX01.CPP`

VisualBasic programs end in ".FRM" and are followed by "(VB)" for easy recognition; for example: `DAQEX.FRM  (VB)`

Note that program references are not included with every function.

Predefined Parameter Definitions

Many of the predefined parameter values available in the DaqX header files (.h, .bas, .pas) follow a naming convention based upon the type of function or operation they are affecting. These conventions usually take the form of a prefix on the parameter value's name.

Prefix	Data Type	Meaning
Derr...	DaqError	Error return code defintions
Dte...	DaqTransferEvent	Acqusition transfer data transfer event definitions
Dwm...	DaqWaitMode	Wait mode definitions for acquisition transfer events
Dhi...	DaqHardwareInfo	Hardware information parameter definitions
Ddi...	DaqInfo	General Daq and expansion device information definitions
Dds...	DaqDetectSensor	Sensor detection definitions
Dhef...	DaqHardwareExtFeatures	Extended hardware features not available in base type such as I/O and memory module support
DaqProtocol	DaqProtocol	Protocol definition for the hardware interface available for the device
Daet...	DaqAdcExpType	Analog expansion unit bank type definitions for smart DBK cards (DBK4, DBK7, DBK5,DBK2, DBK50 etc)
Doct...	DaqAdcExpType	Analog option unit type definitions for WBK option cards (WBK11,WBK12,WBK13 etc)
Dmct...	DaqAdcExpType	Analog module unit type definitions for WBK modules (WBK10,WBK14,WBK15,WBK16 etc)
Dcot...	DaqOptionType	Channel option type definitions for WBK card options
Ddcot...	DaqOptionType	Option type defnitions for DBK card options
Dmot...	DaqOptionType	Option type definitions for WBK expansion modules
Dbot...	DaqOptionType	Option type definitions for base (main) unit WaveBooks
Dcof...	DaqChanOptionFlagType	Flag to indicate whether to apply the option to the channel specified or the entire module defined by the channel
Dcov...	DaqChanOptionValue	Option value definitions for DBK and WBK option types
Dgain...	DaqAdcGain	Base (main) unit analog gain codes for DaqBook/DaqBoard products
Pgain...	DaqAdcGain	Base (main) unit analog gain codes for PersonalDaq products
Tgain...	DaqAdcGain	Base (main) unit analog gain codes for TempBook products
Wgc...	DaqAdcGain	Analog gain codes for WaveBook products
Dbk...	DaqAdcGain	Analog gain codes for DaqBook/DaqBoard DBK expansion modules
Tbk...	DaqAdcGain	Analog gain codes for TempBook Thermocouple channels
Daf...	DaqAdcFlag	Channel definition flags use to configure basic channel types
Dar...	DaqAdcRange	A/D range definitions
DaqType...	DaqChannelType	DBK channel type definitions
Dats...	DaqAdcTriggerSource	Acqusition trigger event (start/stop) source definitions
Dets...	DaqEnhTrigSensT	Defines trigger sensitivities according to the trigger source definition (rising/falling, above/below etc)

Prefix	Data Type	Meaning
Dacs...	DaqAdcClockSource	Acquisition clock (pacing) source definitions
Darm...	DaqAdcRateMode	Rate mode selection (Period, Frequency, External etc)
Daas...	DaqAdcAcqState	Acquisition state for which the function or operation is to apply (Pre-trigger, Post-trigger, etc)
Daam...	DaqAdcAcqMode	Defines the valid acquisition modes for the device (counted, infinite, pre and post trigger, re-arm)
Daom...	DaqAdcOpenMode	Defines acquisition to file open modes (append, create, write)
Datm...	DaqAdcTransferMask	Defines acqusition and transfer events
Daaf...	DaqAdcActiveFlag	Defines acquisition transfer and acqusition states and events
Dabtm...	DaqAdcBufferXferMask	Defines acqusition data buffer allocation and usage
Dardf...	DaqAdcRawDataFormatT	Defines raw format types
Daca...	DaqAdcCvtAction	Defines conversion actions
Dddt...	DaqDacDeviceType	Defines D/A converter device types
Ddom...	DaqDacOutputMode	Defines D/A converter output modes such as direct (DC) and static/dynamic waveform modes
Ddts...	DaqDacTriggerSource	Defines trigger event definitions
Ddcs...	DaqDacClockSource	Defines clock source types for waveform output to a D/A channel
Ddwm...	DaqDacWaveformMode	Defines waveform output modes (static, dynamic, from file)
Ddwt...	DaqDacWaveType	Defines predefined waveform types
Ddtm...	DaqDacTransferMask	Defines waveform output events
Ddwdf...	DaqDacWaveFileDataFormat	Defines the data file format types for waveform output from file
Ddaf...	DaqDacActiveFlag	Defines waveform transfer and waverform states and events
Dcal...	DcalType	Defines calibration operation for the specified channel on the device
Dci...	DaqCalInputT	Defines calibration input channel source
Dctt...	DaqCalTableTypeT	Defines calibration table selections (Factory/User)
Dco...	DaqCalOperationT	Defines calibration operations available for a calabrating a device
Dcopt...	DaqCalOptionT	Defines the calibration options
Diodt...	DaqIODeviceType	Defines I/O device types
Diodp...	DaqIODevicePort	Defines Local I/O ports
Dioep...	DaqIOExpansionPort	Defines Expansion I/O ports
Dc0c...	DaqCntr0Config	Configuration parameters for Counter 0
Diooc...	DaqIOOperationCode	Defines I/O operation codes
Dtod...	Daq9513TimeOfDay	Defines time of day parameters for programming time-of-day on 9513
Dgc...	Daq9513GatingControl	Defines Gating Control parameters for 9513 counters
Dcs...	Daq9513CountSource	Defines counter source for 9513 counter channel
Doc...	Daq9513OutputControl	Defines timer output control definitions for 9513
Dmcc...	Daq9513MultCtrCommand	Defines multi-counter commands for 9513
Dtst...	DaqTestCommand	Defines test types for device testing

Mask and Flag Definitions

There are a number of parameter types which represent bit-masked values. These bit-masks are used to represent complex settings or states which may not be easily represented by a single value. These parameter data types can be identified by the postfix *...Flag* or *...Mask* in the data type name. Special care should be execercised when using these types since one parameter value can represent a number of different states or conditions. These parameter types have a set of pre-defined bit-mask enumerations which represent indvidual states or conditions. These pre-defined enumerations can be found with the …Flag and …Mask type definitions found in the DaqX header files (DaqX.c, DaqX.bas, DaqX.pas)

Setting/Constructing Mask and Flag Values

Mask and Flag parameter values can be constructed by "adding" the desired flags together: The following illustrates how to do this in C/C++ and Visual Basic:

C/C++

Value = Flag1 + Flag2 + Flag3 + ... Flagn;	// C language format
channelFlags = DafAnalog + DafBipolar + DafUnsigned;	// C language example

Visual Basic

Value = Flag1 + Flag2 + Flag3 + ... Flagn	' Visual Basic format
channel Flags& = DafAnalog& + DafBipolar& + DafUnsigned&	' Visual Basic example

An equivalent method is to logically "or" the flags together. See language documentation to use the "or"ing method.

Reading/Interpreting Mask and Flag Values

Reading and interpretting mask and flag parameters returned by the DaqX driver can be a bit trickier. What is needed here is to logically "and" the flags that are of interest with the flags returned from the DaqX API. Those flag bits which are set (bit value =1)

C/C++

```
    Flags = Flag1 + Flag2 + Flag3 + ... Flagn;                    // C language format
    If ((Flags & returnedFlags) == Flags) ;                       // C language format
    {
            // Code for flagsSet condition
    } else {
            // Code for flags NOT set condition
    }

    // C Language Example
    acqFlags = DaafTriggered + DaafAcqActive;                     // C language
example
    if ((acqFlags & acqReturned) == acqFlags) {
    {
            printf("Acquisition has been triggered and post-trigger data is being acquired\n");
    } else {
            if (acqReturned & DaafAcqActive) {
                    printf(" Acquistion is active but has not been triggered\n");
            } else {
    printf("Acquisition is not active\n");
    }
}
```

Visual Basic

```
    Value = Flag1 + Flag2 + Flag3 + ... Flagn                    ' Visual Basic format
    channel Flags& = DafAnalog& + DafBipolar& + DafUnsigned&     ' Visual Basic
example
```

API Commands, Alphabetic Listing

daqAdcArm
daqAdcBufferRotate
daqAdcCalcTrig
daqAdcDisarm
daqAdcExpSetBank
daqAdcGetFreq
daqAdcGetScan
daqAdcRd
daqAdcRdN
daqAdcRdScan
daqAdcRdScanN
daqAdcSetAcq
daqAdcSetClockSource
daqAdcSetDataFormat
daqAdcSetDiskFile
daqAdcSetFreq
daqAdcSetMux
daqAdcSetRate
daqAdcSetScan
daqAdcSetSetpoints
daqAdcSetTrig
daqAdcSetTrigEnhanced
daqAdcSoftTrig
daqAdcTransferBufData
daqAdcTransferGetStat
daqAdcTransferSetBuffer
daqAdcTransferStart
daqAdcTransferStop
daqAutoZeroCompensate
daqCalClearCalTable
daqCalConvert
daqCalGetCalEquation
daqCalGetConstants
daqCalPerformSelfCal
daqCalSaveCalTable
daqCalSaveConstants
daqCalSelectCalTable
daqCalSelectInputSignal
daqCalSetCalEquation
daqCalSetCalPoints
daqCalSetConstants
daqCalSetup
daqCalSetupConvert
daqClose
daqCreateDevice
daqCvtChannelType
daqCvtHardwareType
daqCvtLinearConvert
daqCvtLinearSetup
daqCvtLinearSetupConvert
daqCvtRawDataFormat
daqCvtRtdConvert
daqCvtRtdSetup

daqCvtRtdSetupConvert
daqCvtSetAdcRange
daqCvtTCConvert
daqCvtTCConvertF
daqCvtTCSetup
daqCvtTCSetupConvert
daqCvtTCSetupConvertF
daqDacSetOutputMode
daqDacTransferGetStat
daqDacTransferStart
daqDacTransferStop
daqDacWaveArm
daqDacWaveDisarm
daqDacWaveGetFreq
daqDacWaveSetBuffer
daqDacWaveSetClockSource
daqDacWaveSetDiskFile
daqDacWaveSetFreq
daqDacWaveSetMode
daqDacWaveSetPredefWave
daqDacWaveSetTrig
daqDacWaveSetUserWave
daqDacWaveSoftTrig
daqDacWt
daqDacWtMany
daqDefaultErrorHandler
daqDeleteDevice
daqFormatError
daqGetChannelType
daqGetDeviceCount
daqGetDeviceInfo
daqGetDeviceInventory
daqGetDeviceList
daqGetDeviceProperties
daqGetDriverVersion
daqGetHardwareInfo
daqGetHardwareType
daqGetInfo
daqGetLastError
daqIOGet8255Conf
daqIORead
daqIOReadBit
daqIOWrite
daqIOWriteBit
daqOnline
daqOpen
daqProcessError
daqReadCalFile
daqSetDefaultErrorHandler
daqSetErrorHandler
daqSetOption
daqSetTimeout
daqSetTriggerEvent

daqTest
daqWaitForEvent
daqWaitForEvents
daqZeroConvert
daqZeroSetup
daqZeroSetupConvert

Note: for 9513 chip-based devices refer to Appendix D.

API Commands, Grouped by Function

Many API commands cannot be used universally across all products. For example, a command that pertains only to analog output is of no use to a device that does not support analog I/O. Thus it is important to understand the features and capabilities of your hardware before using API Commands. Prior awareness can save a great deal of programming time and sidestep possible frustration.

General Commands	Calibration, Intialization, Handling, Getting Information …
Functional Category	*Command*

Calibration
(Software Calibration)
	`daqCalClearCalTable`
	`daqCalConvert`
	`daqCalGetCalEquation`
	`daqCalGetConstants`
	`daqCalPerformSelfCal`
	`daqCalSaveCalTable`
	`daqCalSaveConstants`
	`daqCalSelectCalTable`
	`daqCalSelectInputSignal`
	`daqCalSetCalEquation`
	`daqCalSetCalPoints`
	`daqCalSetConstants`
	`daqCalSetup`
	`daqCalSetupConvert`
	`daqReadCalFile`

Error Handling	`daqDefaultErrorHandler`
	`daqFormatError`
	`daqGetLastError`
	`daqProcessError`
	`daqSetDefaultErrorHandler`
	`daqSetErrorHandler`

Event Handling	`daqSetTimeout`
	`daqWaitForEvent`
	`daqWaitForEvents`

Get Information, Utility Function
(also see "Initialization, Device Initialization")
	`daqGetChannelType`
	`daqGetDriverVersion`
	`daqGetHardwareInfo`
	`daqGetHardwareType`
	`daqGetInfo`

Table Continued . . .

Initialization (Device Initialization)	`daqClose`
	`daqCreateDevice`
	`daqDeleteDevice`
	`daqGetDeviceCount`
	`daqGetDeviceInfo`
	`daqGetDeviceInventory`
	`daqGetDeviceList`
	`daqGetDeviceProperties`
	`daqOnline`
	`daqOpen`

Read/Write, General I/O	`daqIOGet8255Conf`
	`daqIORead`
	`daqIOReadBit`
	`daqIOWrite`
	`daqIOWriteBit`

Test	`daqTest`

Conversion Commands	**Conversions and Zero Offset**
Functional Category	*Command*
Calibration	*See "General Commands"*
Convert Type and Sub Type to String	`daqCvtChannelType` `daqCvtHardwareType`

Data Format	`daqAdcSetDataFormat`
	`daqCvtRawDataFormat`
	`daqCvtSetAdcRange`

Linear Conversion	`daqCvtLinearConvert`
	`daqCvtLinearSetup`
	`daqCvtLinearSetupConvert`

RTD Conversion	`daqCvtRtdConvert`
	`daqCvtRtdSetup`
	`daqCvtRtdSetupConvert`

Thermocouple Conversion	`daqCvtTCConvert`
	`daqCvtTCConvertF`
	`daqCvtTCSetup`
	`daqCvtTCSetupConvert`
	`daqCvtTCSetupConvertF`

Zero Offset	`daqAutoZeroCompensate`
	`daqZeroConvert`
	`daqZeroSetup`
	`daqZeroSetupConvert`

ADC Commands	Scan Settings, Expansion, Data Transfer, Triggering …
Functional Category	*Command*
Arm and Disarm	`daqAdcArm` `daqAdcDisarm`
Buffer Manipulation	`daqAdcBufferRotate`

Data Transfer
without Buffer Allocation
	`daqAdcTransferBufData`
	`daqAdcTransferGetStat`
	`daqAdcTransferSetBuffer`
	`daqAdcTransferStart`
	`daqAdcTransferStop`

Direct-to-Disk	`daqAdcSetDiskFile`
Expansion Configuration, ADC	`daqAdcExpSetBank` `daqSetOption`

Scan
 	
    Count, Rate, Source
	`daqAdcGetFreq`
	`daqAdcGetScan`
	`daqAdcSetAcq`
	`daqAdcSetClockSource`
	`daqAdcSetFreq`
	`daqAdcSetMux`
	`daqAdcSetRate`
	`daqAdcSetScan`

Scanning, One-Step	`daqAdcRd`
	`daqAdcRdN`
	`daqAdcRdScan`
	`daqAdcRdScanN`

Setpoints (Configuring Setpoints) *Applies to /3000 Series Devices Only*	`daqAdcSetSetpoints`

Trigger	`daqAdcCalcTrig`
	`daqAdcSetTrig`
	`daqAdcSetTrigEnhanced`
	`daqAdcSoftTrig`
	`daqSetTriggerEvent`

DAC Waveform Commands	Settings, Data Transfer, Waveform Control …
Functional Category	*Command*
Arm and Disarm	`daqDacWaveArm` `daqDacWaveDisarm`

Buffer Management	`daqDacWaveSetBuffer`
	`daqDacWaveSetDiskFile`
	`daqDacWaveSetPredefWave`
	`daqDacWaveSetUserWave`

Data Transfer [of Dynamic Waveform]	`daqDacTransferGetStat`
	`daqDacTransferStart`
	`daqDacTransferStop`

Output Mode	`daqDacSetOutputMode` *Global Configuration* `daqDacWt` *Voltage Output Mode* `daqDacWtMany` *Voltage Output Mode*
Trigger	`daqDacWaveSetTrig` `daqDacWaveSoftTrig`

Update Rate and Count	`daqDacWaveGetFreq`
	`daqDacWaveSetClockSource`
	`daqDacWaveSetFreq`
	`daqDacWaveSetMode`

daqAdcArm

Format

```
daqAdcArm(DaqHandleT handle);
```

Purpose

`daqAdcArm` arms an ADC acquisition by enabling the currently defined ADC configuration for an acquisition.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the configured ADC acquisition is to be armed

Parameter Values

`handle:` obtained from the `daqOpen` function

Returns

`DaqError`   See Daq Error Table

For more details on error messages, please refer to the Daq Error Table.

Function Usage

ADC acquisition will occur when the trigger event (as specified by `daqAdcSetTrig`) is satisfied. The ADC acquisition configuration must be specified prior to the `daqAdcArm` function. For a previously configured acquisition, the `daqAdcArm` function will use the specified parameters. If no previous configuration was given, or it is desirable to change any or all acquisition parameters, then those functions (relating to the desired ADC acquisition configuration) must be issued prior to calling `daqAdcArm`. As a general rule the following needs to be done before arming the acquisition:

Configure the channels to be scanned (daqAdcSetScan, daqAdcSetMux, daqSetOption)

Configure the acquisition rate or frequency (daqAdcSetFreq, daqAdcSetRate)

Configure the acquisition mode (daqAdcSetAcq)

Configure the acquisition buffer (daqAdcTransferSetBuffer)

Enable the data transfer (daqAdcTransferStart)

Any errors in these pre-requisites are deferred to the `daqAdcArm` call.

The `daqAdcArm` function can not be used in conjunction with `daqAdcRd`... functions . These functions are single step functions and automatically arm the device.

The device acquisition configuration is programmed into the device upon execution of the `daqAdcArm` function. Consequently, some errors in programming the acquisition configuration will be reported upon return of this function.

For more information on acquisition setup, see *Seven Easy Steps to Data Acquisition* in Chapter 2.

Prototypes

C/C++

```
daqAdcArm(DaqHandleT handle);
```

Visual BASIC

```
VBdaqAdcArm&(ByVal handle&)
```

Program References

```
DAQADCEX1.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK54EX.CPP, DAQADCEX07.CPP, DAQEX.FRM (VB)
```

daqAdcBufferRotate

Format

```
daqAdcBufferRotate(handle, buf, scanCount, chanCount, retCount)
```

Purpose

daqAdcBufferRotate linearizes a circular buffer acquired via a transfer in cycle mode.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device whose ADC transfer buffer will be rotated
buf	PWORD	Pointer to the buffer being rotated
scanCount	DWORD	Total number of scans in the buffer
chanCount	DWORD	Number of channels in each scan
retCount	DWORD	Last value returned in the retCount parameter of the daqAdcTransferGetStat function

Parameter Values

handle: obtained from the daqOpen function

buf: must be a valid pointer to memory whose size is at least equal to
    (scanCount * chanCount * 2)

scanCount: valid length of buffer from 1 to 4,294,967,295 scans; however, memory limitations apply

chanCount: defined by channel configuration; see daqAdcSetScan for details

retCount: valid range of 1 to 4,294,967,295

Returns

DerrNoError   No error

Function Usage

This function will organize the circular buffer chronologically. In other words, it will order the data from oldest-first to newest-last in the buffer. daqAdcBufferRotate is used primarily with pre-trigger scans

When scans are configured using daqAdcTransferSetBuffer with a DatmCycleOn value for the transferMask parameter, the buffer is set up as a circular buffer--once it is full, it is re-used, starting at the beginning of the buffer. Thus, when the acquisition is complete, the buffer may have been written over many times and the location of the last acquired scan may be at any point within the buffer.

For example, suppose a buffer is set to hold 60 scans, and an acquisition of 1000 scans is triggered. The buffer is first filled with scans 1 through 60. Once the end of the buffer is reached, new scans are written at the beginning of the buffer: scan 61 overwrites scan 1, scan 62 overwrites scan 2, and so on, until scan 120 overwrites scan 60. At this point, the end of the buffer has been reached again--so, scan 121 is stored at the beginning of the buffer, overwriting scan 61. This process of writing and overwriting the buffer continues until all 1000 scans have been acquired. At this point, the buffer has the following contents:

Buffer Position	1	2	3	...	39	40	41	42	...	59	59	60
Scan	961	962	963	...	999	1000	941	942	...	958	959	960

Since the total number of scans is not an even multiple of the buffer size, the oldest scan is not at the beginning of the buffer, and the last scan is not at the end of the buffer. The daqAdcBufferRotate function can rearrange the scans into a more natural order, writing the final scan into the final buffer position. This results in the follwoing arrangement:

Buffer Position	1	2	3	...	39	40	41	42	...	59	59	60
Scan	941	942	943	...	979	980	981	982	...	998	999	1000

If the total number of acquired scans is less than the buffer size, then the scans will not overwrite earlier scans and the buffer is already in a natural chronological order. In this case, `daqAdcBufferRotate` does not modify the buffer.

In WaveBook/512 applications, `daqAdcBufferRotate` will only work on unpacked samples.

Prototypes

C/C++

```
daqAdcBufferRotate(DaqHandleT handle, PWORD buf, DWORD scanCount, DWORD
chanCount, DWORD retCount);
```

Visual BASIC

```
VBdaqAdcBufferRotate&(ByVal handle&, buf%, ByVal scanCount&, ByVal
chanCount&, ByVal retCount&)
```

Program References

```
None
```

daqAdcCalcTrig

Format

```
daqAdcSetTrig(handle, bipolar, gainVal, voltageLevel, triggerLevel)
```

Purpose

`daqAdcSetTrig` configures the device for enhanced triggering.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the trigger level will be calculated
bipolar	BOOL	Boolean that, when `true`, sets trigger to bipolar, and when `false`, sets trigger to unipolar
gainVal	FLOAT	Gain value of the trigger channel
voltageLevel	FLOAT	Voltage level to trigger at
triggerLevel	PWORD	A pointer to the returned count which is used to program the trigger with the `daqAdcSetTrig` function

Parameter Values

handle: obtianed from the `daqOpen` function

bipolar: valid values are either `true` (≠ 0) or `false` (= 0)

gainVal: valid values range from 1, 2, 4, 8, 16, 32, 64

voltageLevel: valid values range from –10 to +10 Volts

triggerLevel: pointer to a value from 0 to 65,535

Returns

DaqError   See Daq Error Table

For more details on error messages, please refer to the Daq Error Table.

Function Usage

This function has been obsoleted by the `daqSetTriggerEvent` function, and is presented here only as a reference. See `daqSetTriggerEvent` for more details.

The `daqAdcCalcTrig` function calculates the trigger level and source for an analog trigger. The result of `daqAdcCalcTrig` is stored in the `triggerLevel` parameter. The value of this parameter can then be passed to the `daqAdcSetTrig` function to configure the analog trigger. All of this can be accomplished using the `daqSetTriggerEvent` function.

Prototypes

C/C++

```
daqAdcCalcTrig(DaqHandleT handle, BOOL bipolar, FLOAT gainVal, FLOAT
voltageLevel, PWORD triggerLevel);
```

Visual BASIC

```
VBdaqAdcCalcTrig&(ByVal handle&, ByVal bipolar&, ByVal gainVal!, ByVal
voltageLevel!, triggerLevel&)
```

Program References

```
None
```

daqAdcDisarm

Format

```
daqAdcDisarm(handle)
```

Purpose

`daqAdcDisarm` disarms an ADC acquisition, if one is currently active.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which to disable ADC acquisitions

Parameter Values

handle: obtained from the `daqOpen` function

Returns

DerrNoError   No error

Funtion Usage

If the specified trigger event has not yet occurred when the `daqAdcDisarm` function is called, the trigger event will be disabled and no ADC acquisition will be performed. If the trigger event has occurred when the `daqAdcDisarm` function is called, the acquisition will be halted and the data transfer stopped and no more ADC data will be collected.

Protypes

C/C++

```
daqAdcDisarm(DaqHandleT handle);
```

Visual BASIC

```
VBdaqAdcDisarm&(ByVal handle&)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK53EX.CPP, DBK04EX.CPP, DBK07EX.CPP, DBK08EX.CPP, DBK09EX.CPP,
DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP, DBK17EX.CPP, DBK18EX.CPP,
DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP, DBK44EX.CPP, DBK45EX.CPP,
DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP, DBK53_54EX.CPP
```

daqAdcExpSetBank

Also See: `daqSetOption`

Format

```
daqAdcExpSetBank(handle, chan, bankType)
```

Purpose

`daqAdcExpSetBank` internally programs intelligent DBK card channels so the device's gains may be set just before the acquisition.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which to set the expansion bank
chan	DWORD	First channel number on the DBK card (channel numbers are grouped as 16 channels per bank)
bankType	DaqAdcExpType	Type of channel bank

Parameter Values

handle: obtained from the `daqOpen` function

chan: valid values from 0 to 271 and are defined by channel configuration; see `daqAdcSetScan` for details

bankType: see table below

Parameter Type Definitions

bankType-(DaqAdcExpType)	
Definition	**Description**
DaetNotDefined	Bank is unknown or undefined the bank
DaetDbk50	Dbk50 or Dbk51 option
DaetDbk5	Dbk5 option
DaetDbk2	Dbk2 option
DaetDbk4	Dbk4 option
DaetDbk7	Dbk7 option
DaetDbk55	Dbk55 option
DaetDbk90	Dbk90 option

Returns

DerrInvChan   Invalid channel number

For more details on error messages, please refer to the Daq Error Table.

Funtion Usage

A bank consists of 16 main unit channels, allocated to a number of connect DBK cards. DBK cards in a single bank must be all of the same type. The `daqAdcExpSetBank` function must be called once for each card in the bank. For example, if four 4-channel cards (such as a DBK7) are used in the first expansion bank, you must call `daqAdcExpSetBank` 4 times with channels 16, 20, 24, and 28.

Prototypes

C/C++

```
daqAdcExpSetBank(DaqHandleT handle, DWORD chan, DaqAdcExpType bankType);
```

Visual BASIC

```
VBdaqAdcExpSetBank&(ByVal handle&, ByVal chan&, ByVal bankType&)
```

Program References

```
DBK04EX.CPP, DBK07EX.CPP, DBK50EX.CPP
```

daqAdcGetFreq

Format

```
daqAdcGetFreq(handle, freq)
```

Purpose

`daqAdcGetFreq` reads the sampling frequency of the pacer clock.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which to get the current frequency setting
freq	PFLOAT	A variable to hold the currently defined sampling frequency in Hz

Parameter Values

`handle:` obtained from the `daqOpen` function
`freq:` must be a valid pointer to a variable defined as a single precision (4-byte) floating point value

Returns

DaqError   See Daq Error Table

Function Usage

This function can be used after calling the `daqAdcSetScan` and `daqAdcSetRate` functions to retrieve the pacer clock's sampling frequency. Valid values for the `freq` parameter can range from 0.2 mHz to 1.0 MHz, but they are dependent on the device being used--see the discussion of actual scan rates in the entry for `daqAdcSetRate`. If using a DaqBook or DaqBoard(ISA) device, `daqAdcGetFreq` assumes that the 1 MHz/10 MHz jumper is set to the default position of 1 MHz.

This function is here for reference only, since it closely resembles the `daqAdcSetRate` function with its `mode` parameter set to `DarmFrequency`. It is recommended that the `daqAdcSetRate` function be used to retrieve the current acquisition frequency setting.

Prototypes

C/C++

```
daqAdcGetFreq(DaqHandleT handle, PFLOAT freq);
```

Visual BASIC

```
VBdaqAdcGetFreq&(ByVal handle&, freq!)
```

Program References

None

daqAdcGetScan

Format

```
daqAdcGetScan(handle, channels, gains, flags, chanCount)
```

Purpose

daqAdcGetScan reads the current scan group, which consists of all configured channels.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which the current scan configuration will be retrieved
channels	PDWORD	A pointer to an array which holds up to 512 channel numbers, or 0 if the channel information is not desired
gains	DaqAdcGain	A pointer to an array which holds up to 512 gain values, or 0 if the channel gain information is not desired
flags	PDWORD	A pointer to channel configuration flags in the form of a bit mask
chanCount	PDWORD	A pointer to a variable which will hold the number of values returned in the channels and gains arrays

Parameter Values

handle: obtained from the daqOpen parameter

channels: must be a valid pointer to an array which can hold up to chanCount channel numbers
(4 bytes/channel); valid values for channel numbers range from 0 to 512

gains: must be a valid pointer to an array which can hold up to chanCount gain definitions
(4 bytes/gain); see ADC Gain Definition table for gains values

flags: must be a valid pointer to an array which can hold up to chanCount flag configurations
(4 bytes/flag); see ADC Flag Definition table for flags values

chanCount: defined by channel configuration; see daqAdcSetScan for details

Returns

DerrNoError   No error

Function Usage

The returned parameter settings directly correspond to those set using the daqAdcSetScan function. For further description of these parameters, refer to daqAdcSetScan.

Flags may have the `DafSSHHold` bit set.

Prototypes

C/C++

```
daqAdcGetScan(DaqHandleT handle, PDWORD channels, DaqAdcGain *gains,
PDWORD flags, PDWORD chanCount);
```

Visual BASIC

```
VBdaqAdcGetScan&(ByVal handle&, channels&(), gains&(), flags&(),
chanCount&)
```

Program References

None

daqAdcRd

Format

```
daqAdcRd(handle, chan, sample, gain, flags)
```

Purpose

`daqAdcRd` takes a single reading from the given local A/D channel using a software trigger.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which the ADC reading will be acquired
chan	DWORD	A single channel number for which the sample will be taken
sample	PWORD	A pointer to a value where an acquisition sample is stored
gain	DaqAdcGain	The channel's gain setting
flags	DWORD	Channel configuration flags in the form of a bit mask

Parameter Values

handle: obtained from the `daqOpen` function

chan: must be a valid channel number for the device

sample: must be a valid pointer to a short integer variable (2 bytes)

gain: see ADC Gain Definition table for `gain` parameter values

flags: see ADC Flag Definition table for `flags` parameter values

Returns

DerrInvGain   Invalid gain

DerrInvChan   Invalid channel

DerrNoError   No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

This function will use a software trigger to immediately trigger and acquire one sample from the specified acquisition channel.

Prototypes

C/C++

```
daqAdcRd(DaqHandleT handle, DWORD chan, PWORD sample, DaqAdcGain gain,
DWORD flags);
```

Visual BASIC

```
VBdaqAdcRd&(ByVal handle&, ByVal chan&, sample%, ByVal gain&, ByVal
flags&)
```

Program References

```
None
```

daqAdcRdN

Format

```
daqAdcRdN(handle, chan, buf, scanCount, triggerSource, rising, level,
freq, gain, flags)
```

Purpose

daqAdcRdN takes multiple scans from a single acquisition channel.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which the ADC channel samples will be acquired
chan	DWORD	A single channel number
buf	PWORD	Array to which the acquisition scans will be returned
scanCount	DWORD	Number of scans to be taken
triggerSource	DaqAdcTriggerSource	The trigger source
rising	BOOL	Boolean flag to indicate whether the edge for the trigger source is rising or falling
level	WORD	The trigger level, if an analog trigger is specified
freq	FLOAT	The sampling frequency in Hz
gain	DaqAdcGain	The channel's gain setting
flags	DWORD	Channel configuration flags in the form of a bit mask

Parameter Values

handle: obtained form the daqOpen function

chan: must be a valid channel number for the device

buf: must be a valid pointer to memory whose size is at least equal to [scanCount * (the sample size, normally 2 bytes)]

scanCount: valid values range from 1 to 4,294,967,295 scans; however, memory limitations apply.

triggerSource: see table below

rising: valid values are either true ($\neq 0$) or false ($= 0$)

level: valid values range from 0 to 65,535, representing the trigger level in raw, unsigned A/D counts

freq: see the daqAdcSetRate function for details.

gain: see ADC Gain Definition table for gain definitions

flags: see ADC Flag Definition table for flag configurations

Parameter Type Definitions

triggerSource-(DaqAdcTriggerSource)	
Definition	**Description**
DatsImmediate	Begins taking post trigger scans immediately upon daqAdcArm function
DatsAdcClock	Begins taking post trigger scans upon detection of next pacer clock pulse
DatsGatedAdcClock	Begins taking post trigger scans upon detection of next gated pacer clock pulse
DatsExternalTTL	Begins taking post trigger scans on the selectable edge of an external TTL signal on pin 25 of P1
DatsHardwareAnalog	Begins taking post trigger scans upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)

Returns

DerrFIFOFull	Buffer overrun
DerrInvGain	Invalid gain
DerrIncChan	Invalid channel
DerrInvTrigSource	Invalid trigger
DerrInvLevel	Invalid level

For more details on error messages, please refer to the Daq Error Table.

Function Usage

This function will:

Configure the pacer clock

Configure the channel with the specified `gain` parameter

Configure the channel options with the channel `flags` specified

Arm the trigger

Acquire `count` scans from the specified A/D channel

The `freq` parameter can have a range of values between 0.2 mHz and 1.0 MHz, but they are dependant on the device being used--see the discussion of actual scan rates in the entry for `daqAdcSetRate`.

Prototypes

C/C++

```
daqAdcRdN(DaqHandleT handle, DWORD chan, PWORD buf, DWORD scanCount,
DaqAdcTriggerSource triggerSource, BOOL rising, WORD level, FLOAT freq,
DaqAdcGain gain, DWORD flags);
```

Visual BASIC

```
VBdaqAdcRdN&(ByVal handle&, ByVal chan&, buf%, ByVal scanCount&, ByVal
triggerSource&, ByVal rising&, ByVal level%, ByVal freq!, ByVal gain&,
ByVal flags&)
```

Program References

```
None
```

daqAdcRdScan

Format

```
daqAdcRdScan(handle, startChan, endChan, buf, gain, flags)
```

Purpose

`daqAdcRdScan` immediately activates a software trigger to acquire one scan consisting of each channel. The scan begins with `startChan` and ends with `endChan.`

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which the ADC scan will be acquired
startChan	DWORD	The starting channel of the scan group
endChan	DWORD	The ending channel of the scan group
buf	PWORD	An array into which the acquisition scans will be placed
gain	DaqAdcGain	The channel's gain setting
flags	DWORD	Channel configuration flags in the form of a bit mask

Parameter Values

`handle:` obtained from the `daqOpen` function

`startChan:` must be a valid channel number for the device

`endChan:` must be a valid channel number for the device and greater than `startChan`

`buf:` must be a valid pointer to memory whose size is at least equal to:

[(`endChan` - `startChan`)* (the sample size, normally 2 bytes)]

`gain:` see ADC Gain Definition table for `gain` parameter values

`flags:` see ADC Flag Definition table for `flags` parameter values

Returns

DerrInvGain	Invalid gain
DerrIncChan	Invalid channel
DerrNoError	No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

This function will use a software trigger to immediately trigger and acquire one scan. This scan will sample each channel selected, starting with `startChan` and ending with `endChan.` The `gain` setting will be applied to all channels.

Channels must all be of the same channel type.

Prototypes

C/C++

```
daqAdcRdScan(DaqHandleT handle, DWORD startChan, DWORD endChan, PWORD buf,
DaqAdcGain gain, DWORD flags);
```

Visual BASIC

```
VBdaqAdcRdScanN&(ByVal handle&, ByVal startChan&, ByVal endChan&, buf%,
ByVal scanCount&, ByVal gain&, ByVal flags&)
```

Program References

```
None
```

daqAdcRdScanN

Also See: **daqAdcSetMux, daqAdcSetClockSource, daqAdcSetTrig, daqAdcRdN**

Format

    daqAdcRdScanN(handle, startChan, endChan, buf, scanCount, triggerSource,
    rising, level, freq, gain, flags)

Purpose

daqAdcRdScanN reads multiple scans from a set of consecutive acquisition channels.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which acquisition scans will be acquired
startChan	DWORD	The starting channel of the scan group
endChan	DWORD	The ending channel of the scan group
buf	PWORD	An array into which the acquisition scans will be placed
scanCount	DWORD	The number of scans to be read
triggerSource	DaqAdcTriggerSource	The trigger source
rising	BOOL	Boolean flag to indicate the rising or falling edge for the trigger source
level	WORD	The trigger level if an analog trigger is specified
freq	FLOAT	The sampling frequency in Hz
gain	DaqAdcGain	The channel's gain setting
flags	DWORD	Channel configuration flags in the form of a bit mask.

Parameter Values

handle: obtained from the daqOpen function

startChan: must be a valid channel number for the device

endChan: must be a valid channel number for the device and greater than startChan

buf: must be a valid pointer to memory whose size is at least equal to:

    [(endChan – startChan) * scanCount * (the sample size, normally 2 bytes)]

scanCount: valid values range from 1 to 4,294,967,295 scans; however, memory limitations apply.

triggerSource: see table below

rising: valid values are either true (≠ 0) or false (= 0)

level: valid values range from 0 to 65,535, representing the trigger level in unsigned A/D counts

freq: see the daqAdcSetRate function for details

gain: see ADC Gain Definition table for gain parameter values

flags: see ADC Flag Definition table for flags parameter values

Parameter Type Definitions

triggerSource-DaqAdcTriggerSource	
Definition	**Description**
DatsImmediate	Begins taking post trigger scans immediately upon daqAdcArm function
DatsAdcClock	Begins taking post trigger scans upon detection of next pacer clock pulse
DatsGatedAdcClock	Begins taking post trigger scans upon detection of next gated pacer clock pulse
DatsExternalTTL	Begins taking post trigger scans on the selectable edge of an external TTL signal on pin 25 of P1
DatsHardwareAnalog	Begins taking post trigger scans upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)

Returns

`DerrInvGain`	Invalid gain
`DerrInvChan`	Invalid channel
`DerrInvTrigSource`	Invalid trigger
`DerrInvLevel`	Invalid level
`DerrFIFOFull`	Buffer overrun
`DerrNoError`	No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

This function will configure the pacer clock, arm the trigger and acquire count scans consisting of each channel, starting with `startChan` and ending with `endChan`. The `gain` and `flags` settings will be applied to all channels.

The `freq` parameter is used to set the acquisition frequency. Its valid values can range from 0.2 mHz to 1.0 MHz, but they are dependent on the device being used--see the discussion of actual scan rates in the entry for `daqAdcSetRate`.

Prototypes

C/C++

```
daqAdcRdScanN(DaqHandleT handle, DWORD startChan, DWORD endChan, PWORD
buf, DWORD scanCount, DaqAdcTriggerSource triggerSource, BOOL rising, WORD
level, FLOAT freq, DaqAdcGain gain, DWORD flags);
```

Visual BASIC

```
VBdaqAdcRdScanN&(ByVal handle&, ByVal startChan&, ByVal endChan&, buf%,
ByVal scanCount&, ByVal triggerSource&, ByVal rising&, ByVal level%, ByVal
freq!, ByVal gain&, ByVal flags&)
```

Program References

None

daqAdcSetAcq

Format

```
daqAdcSetAcq(handle, mode, preTrigCount, postTrigCount)
```

Purpose

`daqAdcSetAcq` configures the acquisition mode and the pre- and post-trigger scan durations.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the ADC acquisition is to be configured
mode	DaqAdcAcqMode	Selects the mode of the acquisition
preTrigCount	DWORD	Number of pre-trigger acquisition scans to be collected
postTrigCount	DWORD	Number of post-trigger acquisition scans to be collected

Parameter Values

handle: obtained from the `daqOpen` function.

mode: see table below

preTrigCount: valid value range from 1 to 100,000

postTrigCount: valid values range from 1 to 4,294,967,295; however, memory limitations may apply

Parameter Type Definitions

mode-(DaqAdcAcqMode)	
Definition	**Description**
DaamNShot	Triggers and continues acquisition until specified number of post-trigger scans is reached, then disarms acquisition
DaamNShotRearm	Triggers and continues acquisition until specified number of post-trigger scans is reached, then re-arms acquisition with original parameters (WaveBook only)
DaamInfinitePost	Once triggered, collects scans until disarmed by the daqAdcDisarm function
DaamPrePost	Collects specified number pre-trigger scans, triggers acquisition, collects specified number of post-trigger scans, and disarms

Returns

DerrNoError   No error

Function Usage

Configuring the Acquisition Mode

The `mode` parameter defines certain characteristics of the data acquisiton. Depending on its parameter values, `mode` can be used to detemine if pre- or post-trigger data will be recorded, how many pre- or post-trigger scans will be collected, and when the acquisition will terminate. The acquistion mode may also specify that the acquisition should be automatically re-armed (using the same acquisition parameters) once the initial acquisition has completed. Each block of aquired trigger scans is placed into the buffer sequentially. The `preTrigCount` and `postTrigCount` parameters specify the respective durations of the pre-trigger and post-trigger acquisition states.

Parameter values for `mode` can be defined as follows:

DaamNShot -- Once triggered, this mode continues acquisition until the specified post-trigger count has been satisfied. Once the post-trigger count has been satisfied, the acquisition will automatically disarm. This mode specifies no pre-trigger and will stop acquiring once the post-trigger scan count has been satisfied. If the acquisition is stopped by means other than reaching the scan count, the stop trigger detection will occur after count scans are acquired.

DaamNShotRearm -- Once triggered, this mode continues the acquisition for the specified post-trigger count, then re-arms the acquisition with the same acquisition configuration parameters as before. The automatic re-arming of the acquisition may be disabled at any time by issuing the `daqAdcDisarm` function. Other than the re-arming feature, this mode works like `DaamNShot.` Upon re-arming, data collection will only be reinitiated when the trigger event re-occurs after the previous acquisition post-trigger count has been satisfied.

The DaamNShotRearm command can only be used with WaveBooks.

DaamInfinitePost -- Once triggered, this mode begins continuous acquisition until explicitly terminated by the `daqAdcDisarm` function.

`DaamPrePost` -- This mode begins collecting the specified number of pre-trigger scans immediately upon issuance of the `daqAdcArm` function. The trigger will not be enabled until the specified number of pre-trigger scans has been collected. Once triggered, the acquisition will continue collecting post-trigger data until the post-trigger count has been satisfied. Once the post-trigger count has been satisfied, the acquisition will automatically disarm itself and terminate.

Daq 2000 Series devices, DaqBoard/1000 Series devices, and WaveBooks [with WBK 30's] in cycle mode both return the exact pre-trigger amount of data specifies. Other devices record data from before the pre-trigger event (i.e., all scans from arming to disarming). See daqAdcBufferRotate for more details.

Relationship to Trigger Start/Stop Events

The `daqAdcSetAcq` function sets the style or mode of the acquisition. However, it does not specify trigger start events, nor does it specify trigger stop events--although it does specify stop conditions (if those stop conditions are scan counts or infinite). Specific trigger start and stop events are defined by other functions:

Trigger start events (*EventA)* can be defined using `daqAdcSetTrigEnhanced` and `daqSetTriggerEvent`.

Trigger stop events (*EventB)* - can be defined using the `daqSetTriggerEvent`.

The following table shows the relationship between trigger event definition and acquisition mode parameter settings for possible acquisition configurations:

Desired Acquisition Without Pre-Trigger Scans	mode	preTrigCount	postTrigCount
Trigger acq on *EventA* and stop on scan *n* after trigger	DaamNShot	Ignored	*n*
Trigger acq on *EventA* and continue indefinitely until disarmed by application (daqAdcDisarm)	DaamInfinitePost	Ignored	Ignored
Trigger acq on *EventA*, stop on scan *n* after trigger, then re-arm to trigger on *EventA* (repeat until disarmed) (**Note 1**)	DaamNShotRearm	Ignored	Ignored
Trigger acq on *EventA* and stop on *EventB**	DaamInfinitePost	Ignored	Ignored
With Pre-Trigger Scans	**mode**	**preTrigCount**	**postTrigCount**
Take *m* pretrigger scans, start acq on *EventA*, and stop on scan *n* after trigger	DaamPrePost	*m*	*n*
Take *m* pretrigger scans, start acq on *EventA*, and stop on *EventB* (**Note 2**)	DaamPrePost	*m*	Ignored

Note 1: WaveBook products only.

Note 2: Applies to DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series, DaqBoard/2000 Series, DaqBoard/1000 Series, and DaqBoard/500.

Prototypes

C/C++

```
daqAdcSetAcq(DaqHandleT handle, DaqAdcAcqMode mode, DWORD preTrigCount,
DWORD postTrigCount);
```

Visual BASIC

```
VBdaqAdcSetAcq&(ByVal handle&, ByVal mode&, ByVal preTrigCount&, ByVal
postTrigCount&)
```

Program References

DDAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK53_54EX.CPP, DAQEX.FRM (VB)

daqAdcSetClockSource

Also See: **daqAdcSetFreq, daqAdcGetFreq, daqAdcSetRate**

Format

```
daqAdcSetClockSource(handle, clockSource)
```

Purpose

daqAdcSetClockSource sets up the clock source to be used to drive the acquisition frequency.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which to set the ADC clock source
clockSource	DaqAdcClockSource	Specifies the clock source for acquisitions

Parameter Values

handle: obtained from the daqOpen function
clockSource: see table below

Parameter Type Definitions

clockSource–(DaqAdcClockSource)		
Definition	**Devices**	**Description**
DacsAdcClock	All	Use the internal pacer clock on the device (see daqAdcSetRate)
DacsGatedAdcClock	DaqBook, DaqBoard (ISA), TempBook	The internal clock is gated through a external input
DacsTriggerSource	DaqBook, DaqBoard (ISA), TempBook	Takes one scan per trigger and stays armed
DacsExternalTTL	Daq Devices [per note 1] , WaveBook/516	Use an external clock input for acquisition clock
DacsAdcClockDiv2	Daq Devices [per note 1] Does not apply to DaqBoard/500 Series	Uses internal clock divided by 2; used in 100kHz mode (Does not apply to DaqBoard/500 Series)
DacsRisingEdge	Daq Devices [per note 1]	External Clock Detection Flag DacsRisingEdge = 0
DacsFallingEdge	Daq Devices [per note 1]	External Clock Detection Flag DacsFallingEdge = 0x100
DacsOutputDisable	Daq Devices [per note 1]	Internal Clock Output Control Flag Disables the Adc internal clock output (see note 2).
DacsOutputEnable	Daq Devices [per note 1]	Internal Clock Output Control Flag; Enables the Adc internal clock output (see note 2). See Note 3.
Note 1:	The description "Daq Devices [per note 1]" indicates the following devices: 2000 Series DaqBooks, DaqBoards, and DaqLabs; DaqBoard/1000 Series; DaqBoard/500 Series, DaqScan/2000 Series. Certain devices within a series may not apply in all cases. Refer to user's manual for your specific product in regard to device capabilities.	

Note 2:	To enable the pacer output capabilities of the device you must include the clock source with a parameter that enables the output clock; in other words, you have to write your **daqAdcSetClockSource** command as follows:
	
	 <u>C/C++ Style</u>:
	
	     daqAdcSetClockSource(handle, DacsAdcClock | DacsOutputEnable);
	
	 <u>Visual Basic Style</u>:
	
	     VbdaqAdcSetClockSource(handle, DacsAdcClock + DacsOutputEnable)
	
	 The **DacsOutputEnable** parameter is defined in the header file in the Daqx.bas module (VB):
	 Output Control Flags
	     Global Const DacsOutputDisable = 0 Disables the Adc internal clock output.
	     Global Const DacsOutputEnable = &H1000 Enables the Adc internal clock output.	

Note 3:	For DaqBoard/500 Series: In regard to DacsOutputEnable the resulting frequency from the Adc internal clock output will equal the Scan Frequency multiplied by the number of channels in the scan list.	

Returns

DerrNoError   No error

Prototypes

C/C++

daqAdcSetClockSource(DaqHandleT handle, DaqAdcClockSource clockSource);

Visual BASIC

VBdaqAdcSetClockSource&(ByVal handle&, ByVal clockSource&)

Program References

DAQADCEX05.CPP, DBK12_13EX.CPP

daqAdcSetDataFormat

Also See: **daqAdcSetMux, daqAdcSetClockSource, daqAdcSetTrig, daqAdcRdN**

Format

```
daqAdcSetDataFormat(handle, rawFormat, postProcFormat)
```

Purpose

`daqAdcSetDataFormat` sets the format for both the raw and post-acquisition data returned by the acquisition transfer functions.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	The handle to the device for which to set the data format
rawFormat	DaqAdcRawDataFormatT	Specifies the raw data format
postProcFormat	DaqAdcPostProcDataFormatT	Parameter value specifying the data formats available

Parameter Values

handle: obtained from the `daqOpen` function
rawFormat: see table below
postProcFormat: see table below

Parameter Type Definitions

rawFormat-(DaqAdcRawDataFormatT)	
Definition	**Description**
DardfNative	Data format follows the native-data format of the device it originated from.
DardfPacked	Raw acquisition values are compressed, using empty bits in native format (WaveBook/512 only)

postProcFormat-(DaqAdcPostProcFormatT)	
Definition	**Description**
DappdfRaw	Post-acquisition data follows the `rawFormat` parameter --this is the default setting.
DappdfTenthsDegC	Data format is in tenths of a degree Celsius

Returns

DerrNoError   No error

Function Usage

Raw Data Formats

The `rawFormat` parameter indicates how the raw data format will be presented. Normally, the raw-data format represents the data from the A/D converter. The default value for this parameter is `DardfNative` where the raw-data format follows the native-data format of the A/D for the particular device. The parameter can also be set to `DardfPacked`, which compresses raw A/D values in order to make full use of all unused bits in any native format that leaves unused bits in the byte-aligned count value. For instance, a 12-bit raw A/D value (which would normally be represented in a 16-bit word, 2-byte count value) will be compressed so that 4 12-bit A/D raw counts can be represented in 3 16-bit word count values. Currently, only the WaveBook/512 supports this packed format (used with the generic functions of the form `daqAdcTransfer...`).

Post-Acquisition Data Formats

The `postProcFormat` parameter specifies the format for which post-acquisition data will be presented. It is only valid for TempBook devices reading thermocouples with one-step functions. The `postProcFormat` format is used by the one-step functions of the form `daqAdcRd…`. The default value is `DappdfRaw`, where the post-acquisition data format will follow the `rawFormat` parameter. The `DappdfTenthsDegC` parameter specifies the data format to be in tenths of a degree on a Celsius scale.

Note! Certain devices may be limited to the types of raw and post-acquisition data formats which can be presented.

Prototypes

C/C++

```
daqAdcSetDataFormat(DaqHandleT handle, DaqAdcRawDataFormatT rawFormat,
DaqAdcPostProcDataFormatT postProcFormat);
```

Visual BASIC

```
VBdaqAdcSetDataFormat&(ByVal handle&, ByVal rawFormat&, ByVal
postProcFormat&)
```

Program References

```
None
```

daqAdcSetDiskFile

Format

```
daqAdcSetDiskFile(handle, filename, openMode, preWrite)
```

Purpose

`daqAdcSetDiskFile` sets a destination file for ADC data transfers. ADC direct-to-disk data transfers will be directed to the specified disk file.

File writing only takes place on a **daqAdcTransferGetStat** call.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device which will perform direct-to-disk ADC acquisition
filename	LPSTR	String representing the path and name of the file where the raw ADC acquisition data will be placed
openMode	DaqAdcOpenMode	Specifies how to open the file for writing
preWrite	DWORD	Specifies the amount to pre-write (in bytes) the file

Parameter Values

handle: obtained from the `daqOpen` command

filename: string of characters representing a file name—no effective range of values applies

openMode: see table below

preWrite: valid values range from 0 to 4,294,967,295; however, disk memory limitations may apply

Parameter Type Definitions

openMode-(DaqAdcOpenMode)	
Definition	**Description**
DaomCreateFile	Creates a new file for incoming ADC transfer data.
DaomWriteFile	Writes incoming ADC transfer data over an existing file.
DaomAppendFile	Appends incoming ADC transfer data to an existing file.

Returns

DerrNoError   No error

Function Usage

The `daqAdcSetDiskFile` function sends acquisition scan data to a disk file specified by the user. Data written to the disk file is the same raw scan data that is read into the buffer. After the data is written to the disk, it is passed on to whatever application makes use of the buffer. The following diagram illustrates the transfer of data:
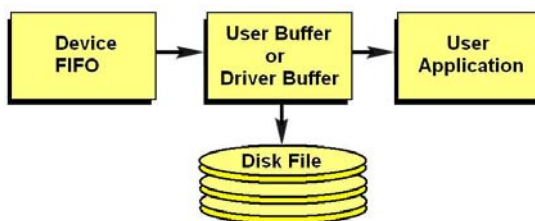
daqAdcSetDiskFile only establishes where and how the data will be sent. The writing of data takes place only after calling either the **daqAdcTransferGetStat** or **daqWaitForEvents** functions.

The `filename` parameter is a string representing the path and name of the file to be opened.

The `openMode` parameter indicates how the file is to be opened for writing data. Valid file open modes are defined as follows:

DaomCreateFile- Create a new file for subsequent ADC transfers. This mode does not require that the file exist beforehand.

DaomWriteFile - Rewrite or write over an existing file. This operation will destroy the original contents of the file.

DaomAppendFile - Open an existing file to append subsequent ADC transfers. This mode should only be used when the existing file has a similar ADC channel group configuration as the subsequent transfers.

The `preWrite` parameter may be used to specify the extent that the file will be pre-written before the actual data collection begins. This may increase the data-to-disk performance of the acquisition, if the amount of data to be collected is known beforehand. If pre-writing is not desired, then the `preWrite` parameter should be set to 0.

Prototypes

C/C++

```
daqAdcSetDiskFile(DaqHandleT handle, LPSTR filename, DaqAdcOpenMode
openMode, DWORD preWrite);
```

Visual BASIC

```
VBdaqAdcSetDiskFile&(ByVal handle&, ByVal filename$, ByVal openMode&,
ByVal preWrite&)
```

Program References

```
DAQADCEX04.CPP, DAQEX.FRM (VB)
```

daqAdcSetFreq

Format

```
daqAdcSetFreq(handle, freq)
```

Purpose

`daqAdcSetFreq` calculates and sets the frequency of the internal scan pacer clock of the device using the frequency specified in Hz.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device whose acquisition frequency is being set
freq	FLOAT	The sampling frequency in Hz

Parameter Values

handle: obtained from the `daqOpen` frequency
freq: valid values range from 0.2 mHz to 1.0 MHz, depending on the device

Returns

DerrNoError   No error

Function Usage

This function follows closely that of the `daqAdcSetRate` function with the mode parameter set to `DarmFrequency`. Valid values for the `freq` parameter range from 0.2 mHz to 1.0 MHz, but are dependent on the device being used--see the discussion of actual scan rates in the entry for `daqAdcSetRate`.

This function is here for reference only. It is recommended that the `daqAdcSetRate` function be used instead.

Prototypes

C/C++

```
daqAdcSetFreq(DaqHandleT handle, FLOAT freq);
```

Visual BASIC

```
VBdaqAdcSetFreq&(ByVal handle&, ByVal freq!)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK53_54.CPP, DAQEX.FRM (VB)
```

daqAdcSetMux

Also See: **daqAdcSetScan, daqAdcGetScan**

Format

    daqAdcSetMux(handle, startChan, endChan, gain, flags)

Purpose

daqAdcSetMux sets a simple scan sequence of local A/D channels from startChan to endChan with the specified gain value.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device whose ADC channel scan group is being configured
startChan	DWORD	The starting channel of the scan group
endChan	DWORD	The ending channel of the scan group
gain	DaqAdcGain	The gain value for all channels
flags	DWORD	Channel configuration flags in the form of a bit mask

Parameter Values:

handle: obtained from the daqOpen function

startChan: must be a valid channel number for the device

endChan: must be a valid channel number for the device and greater than startChan

gain: see ADC Gain Definition table for gain parameter values

flags: see ADC Flag Definition table for flags parameter values

Returns

DerrInvGain	Invalid gain
DerrInvChan	Invalid channel
DerrNoError	No error

Reference Note:
For more details on error messages, please refer to the Daq Error Table.

Function Usage

This function provides a simple alternative to daqAdcSetScan if only consecutive channels need to be acquired. The flags parameter is used to set channel-dependent options. The gain and flags parameters will apply to all channels in the specified range.

Prototypes

C/C++

```
daqAdcSetMux(DaqHandleT handle, DWORD startChan, DWORD endChan, DaqAdcGain
gain, DWORD flags);
```

Visual BASIC

```
VBdaqAdcSetMux&(ByVal handle&, ByVal startChan&, ByVal endChan&, ByVal
gain&, ByVal flags&)
```

Program References

```
DAQEX.FRM (VB)
```

daqAdcSetRate

Also See: **daqAdcSetAcq, daqAdcSetTrig, daqAdcArm, daqAdcSetFreq, daqAdcGetFreq**

Format

```
daqAdcSetRate(handle, mode, state, reqValue, actualValue)
```

Purpose

`daqAdcSetRate` configures the acquisition scan rate using the selected device's built-in acquisition pacer clock.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to device for which to set the scan rate
mode	DaqAdcRateMode	Sets the acquisition mode in terms of period or frequency
state	DaqAdcAcqState	Indicates the acquisition state which scan rate applies to (either pre-trigger or post-trigger)
reqValue	FLOAT	Variable which indicates the requested acquisition scan rate
actualValue	PFLOAT	Variable which indicates the actual acquisition scan rate

Parameter Values

handle: obtained from the `daqOpen` function.

mode: see table below

state: see table below

reqValue: valid values for freq range from 0.0 to 1,000,000.0

actualValue: a pointer to a floating point number, ranging from 0.0 to 1,000,000.0

Parameter Type Definitions

mode-(DaqAdcRateMode)	
Definition	**Description**
DarmPeriod	Sets requested scan rate in terms of period (nanoseconds)
DarmFrequency	Sets requested scan rate in terms of frequency (hertz, or channel scans per second)
DarmExtClockPacer	Sets external pacer clock divisor (WaveBook 516 only)

state-(DaqAdcAcqState)	
Definition	**Description**
DaasPreTrig	Scan rate configuration is applied to pre-trigger acquisition state
DaasPostTrig	Scan rate configuration is applied to post-trigger acquisition state

Returns

DerrNoError   No error

Function Usage

The `daqAdcSetRate` function should be used if internally pacing an acquisition with a device's built-in pacer clock. This function configures the internal pacer clock to the appropriate scanning frequency or period. When the internal pacing clock fires, a new channel scan will be initiated, starting with the first channel in the channel scan group (see the `daqAdcSetScan` function for channel configuration). The specific period or frequency during which these scans will be taken is determined by the acquisition scan rate setting.

> **For correct results, the `daqAdcSetRate` function must be called after `daqAdcSetClockSource`, `daqAdcSetScan` (or `daqAdcSetMux`), and `daqAdcSetTrig`.**

Setting the Scan Rate

The `daqAdcSetRate` function sets the scan rate interval for a channel scan group. The scan rate is set by the `reqValue` parameter. Whether the value is given in terms of frequency or period, the requested scan rate is programmed into the built-in pacer clock as a scan interval timer. Once set, this scan interval timer will fire at the requested rate to initiate the scanning of the channel scan group.
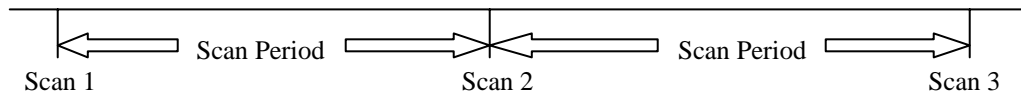
When the scan interval timer fires, the first channel in the channel scan group is sampled. The subsequent channels in the scan group are then sampled at the specified sampling interval for the device. The sampling of the individual channels continues until the last channel in the channel scan group has been sampled. The period between the sampling of the last channel in the channel scan group and the next firing of the scan interval timer is "dead time", where no channel sampling is performed.
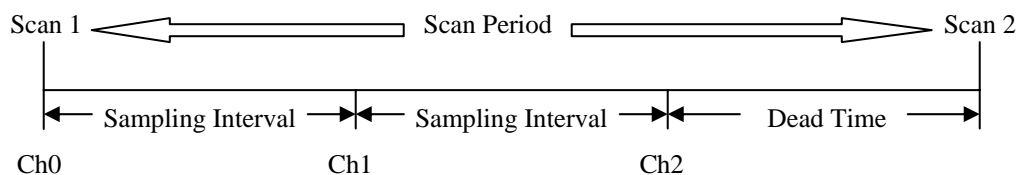
This function does not set the sampling interval between individual channels within the scan group. Channel sampling interval (if programmable) can be set globally by the `daqSetOption` function, or individually by the `flags` parameter of the scan-setting API commands (see `daqAdcSetScan`).

Scan Rate Mode

The `mode` parameter allows the setting of the scan rate in either period or frequency. The possible values for this parameter are as follows:

`DarmPeriod` – This value defines the requested scan rate to be in nanoseconds. In this case, the `reqValue` parameter will be interpreted as the interval between channel scans in nanoseconds.

`DarmFrequency` – This value defines the requested scan rate to be in frequency. In this case, the `reqValue` parameter will be interpreted as the frequency of the scan rate in hertz (or channel scans per second).

`DarmExtClockPacer` -- Sets external pacer clock divisor. Here `reqValue` defines the pacer clock divisor value (1 to 255) when the clock source is defined as external (by setting `DacsExternalTTL` in the `daqAdcSetClockSource` function). When used as external clock divisor, one scan will be initiated every `reqValue` pulses on the external clock input.

The `DarmExtClockPacer` parameter value can only be used with the WaveBook/516.

Scan Rate State

The `state` parameter indicates the acquisition state for which the channel scan rate applies. The possible values for this parameter are as follows:

`DaasPreTrig` – This value causes the scan rate configuration to be applied to the pre-trigger acquisition state. All scanning before the trigger event will be scanned at the rate configured.

`DaasPostTrig` – This value cause the scan rate configuration to be applied to the post-trigger acquisition state. All scanning after the trigger event will be scanned at the rate configured.

> **Only the WaveBook products support different pre- and post-trigger scan rates.**
> **If using a product other than the WaveBook products with pre-trigger configured, the pre-trigger scan rate will follow that of the configured post-trigger scan rate.**

Actual Scan Rate

The `reqValue` parameter represents the desired scan rate. However, the requested scan rate may not be attainable. This is due to the fact that the requested rate may not be evenly divisible by the scan rate setting resolution (see the following table concerning scan rate settings) If this is the case, the actual rate will be set to the next highest scan rate value which is evenly divisible by the scan rate setting resolution.

Another potential reason for having an actual rate different than the requested rate is that the total requested scan rate of the channel scan group exceeds the maximum scan rate for which the device is capable. Each channel in the channel scan group takes a fraction of the total scan rate. That fraction of time is equal to the sampling interval for the channel. For instance, if 2 channels are configured on a WaveBook (1 Mhz max), then each channel will require 1μs sampling interval time--the channel scan group will require 2 μs to sample the entire scan. So, the maximum settable scan rate for this 2 channel scan group would be 2 μs (or 500 Khz).

The `actualValue` parameter returns the actual scan rate, after any adjustments. The format of this parameter will follow that of the `reqValue` parameter, describing the scan rate in either frequency (Hertz) or period (nanoseconds). The format for both `reqValue` and `actualValue` parameters is set using the `mode` parameter.

Product	Maximum Aggregate Rate	Scan Rate Settings (Pacer Clock Resolution)	Sampling Interval/Channel
DaqBook/100 Series DaqBook/200 Series	100 kHz (10μs)	10 μs (with JP5 set to 100 kHz) 1 μs (with JP5 set to 1 MHz)* 0.1 μs (with JP5 set to 10 MHz)	10 μs
DaqBoards(ISA)	100 kHz (10 μs)	1 μs	10 μs
TempBooks	100 kHz (10 μs)	1 μs (with JP5 set to 1 MHz)	10 μs
Daq PC Cards	100 kHz (10 μs)	1 μs	10 μs
WaveBooks	1 MHz (1 μs)	1 μs	1 μs
DaqBoard/500 Series	≤ 200 kHz (5 μs) depending on gain	1 μs	≥ 5 μs depending on gain
DaqBoard/2000 Series cPCI DaqBoard/2000c Series DaqBoard/1000 Series	200 kHz (5 μs) or 100 kHz (10μs) programmable	1 μs	5 μs or 10 μs (programmable)
DaqBook/2000 Series DaqLab/2000 Series DaqScan/2000 Series	200 kHz (5 μs), 100 kHz (10μs) or 1kHz(1000us) programmable	1 μs	5 μs, 10 μs, or 1000us (programmable)

> **For DaqBook/100 Series, DaqBook/200 Series, and TempBooks, this function assumes that the device's JP5 jumper is set to the default setting of 1 MHz. The other settings can be used but they will impact the actual rate that the unit is programmed.**
> **For 100 kHz, the actual rate will be 10 times slower than the `reqValue` programmed.**
> **For 10 MHz, the actual rate will be 10 faster than the `reqValue` programmed. Also, the returned `actualValue` value will be in error in a similar fashion.**

Prototypes

C/C++

```
daqAdcSetRate(DaqHandleT handle, DaqAdcRateMode mode, DaqAdcAcqState
state, FLOAT reqValue, PFLOAT actualValue);
```

Visual BASIC

```
VBdaqAdcSetRate(ByVal handle&, ByVal mode&, ByVal state&, ByVal reqValue!,
actualValue!);
```

Program References

```
None
```

daqAdcSetScan

Format

```
daqAdcSetScan(handle, channels, gains, flags, chanCount)
```

Purpose

`daqAdcSetScan` configures an acquisition scan group consisting of multiple channels.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which acquisition scan group will be configured
channels	PDWORD	An array of channel numbers
gains	DaqAdcGain	Pointer to an array of gain values
flags	PDWORD	Pointer to an array of channel configuration flags in bit mask form.
chanCount	DWORD	The number of elements in the channels, gains and flags arrays

Parameter Values

handle: obtained from the `daqOpen` function

channels: must be a valid pointer to an array of valid channel numbers for the device; valid values for channel numbers vary per device. See the parameter type definitions table below. Consult your hardware manual if needed.

gains: see table below

flags: see table below

chanCount: valid values vary in range, according to product type.

Parameter Type Definitions

flags		
Definition	**Description**	**Valid Channel Values**
Analog/High Speed Digital Flag		
DafAnalog	Channel is Analog on P1	Analog P1 Channels (0-271)
DafHighSpeedDigital	Channel is HS Digital I/O on P3	Digital P3 HS Chan (0)
Data Representation Flags (DaqBook/2000 Series; 1000, 2000, 3000 Series DaqBoards; DaqLab/2000 Series, DaqScan/2000 Series and cPCI DaqBoard/2000c Series)		
DafUnsigned	Channel ADC data is represented as a 16-bit unsigned integer.	Any channel
DafSigned	Channel ADC data is represented as a 16-bit signed integer.	Any channel
Unipolar/Bipolar Flag (DaqBook/2000 Series; 1000, 2000, 3000 Series DaqBoards; DaqLab/2000 Series, DaqScan/2000 Series and cPCI DaqBoard/2000c Series)		
DafUnipolar	Channel voltage range is from 0 to +2x(Device Max/Gain)V	Analog P1 Channels (0-271)
DafBipolar	Channel voltage range is -(Device Max/Gain)V to +(Device Max/Gain)V	Analog P1 Channels (0-271)
Single Ended/Differential Flag		
DafSingleEnded	Channel is single ended inputs	Analog P1 (0-271)
DafDifferential	Channel is differential inputs	Analog P1 (0-7) (Main unit only)
P2 Digital Channel Flags (DaqBook/2000 Series; 1000, 2000, 3000 Series DaqBoards; DaqLab/2000 Series, DaqScan/2000 Series and cPCI DaqBoard/2000c Series)		
DafP2Local8	Channel is Digital on the Local P2	Local Digital P2 Channels (0-4)
DafP2Exp8	Channel is Digital on an Expansion P2	N/A
P3 Digital Channel Flags (DaqBook/2000 Series, DaqBoard/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series and cPCI DaqBoard/2000c Series)		
DafP3Local16	Channel is HS Digital I/O P3	Digital 16-bit P3 Channel (0)
Counter Type Flags (DaqBook/2000 Series; 1000, 2000, 3000 Series DaqBoards; DaqLab/2000 Series, DaqScan/2000 Series and cPCI DaqBoard/2000c Series)		
DafCtrPulse	P3 Counter Channel will Tally Pulses	Counter Channels (0-3)
DafCtrTotalize	P3 Counter Channel will Return Frequency	Counter Channels (0-3)
Counter Edge Flags		
DafCtrRisingEdge	P3 Counter Channel will Count on Rising Edge	Counter Channels (0-3)
DafCtrFallingEdge	P3 Counter Channel will Count on Falling Edge	Counter Channels (0-3)

Counter Channel Flags (DaqBook/2000 Series; 1000, 2000, 3000 Series DaqBoards; DaqLab/2000 Series, DaqScan/2000 Series and cPCI DaqBoard/2000c Series,WaveBook/516; WBK17)		
DafCtr16	Channel is the 16-bit Counter	Counter Channels (0-3) Note 1
DafCtr32Low	Channel is the Lower 16 bits of the 32-bit Counter	32-bit Counter Low Word (0,2) Note 1
DafCtr32High	Channel is the Higher 16 bits of the 32-bit Counter	32-bit Counter High Word (1,3) Note 1
Note 1: The channel numbers provided for "Counter Channel Flags," apply to the DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series; "1000, 2000, 3000 Series DaqBoards;" and cPCI DaqBoard/2000c Series products only.		

WBK17 Digital Output Port Flags		
DafDigital8	Digital Output Port (Byte)	Reading of Digital Output
DafDigital16	Digital Output Port and Detector (Byte) << 8	Reading of Digital Output

SSH Hold/Sample Flag - For Internal Use Only		
DafSSHSample	Internal use only	N/A
DafSSHHold	Internal use only	N/A

Sampling Interval Control (Applies to the following Series Devices: /3000, /2000, and DaqBoard/1000)		
DafSettle1us	Allow 1 µs sampling interval (*3000 Series Only*) **In order to obtain the maximum sampling rate for 3000 Series devices, DafSettle1us must be used!**	DafSettle1us applies only to valid Analog Channels on 3000 Series devices.
DafSettle5us	Allow 5 µs sampling interval	Any valid Analog, Digital, or Counter Channel
DafSettle10us	Allow 10 µs sampling interval	Any valid Analog, Digital or Counter Channel
DafSettle1ms	Allow 1000us (1ms) sampling interval	For use with DBK90 Channels

Clear or shift the least significant nibble (DaqBook/1xx,DaqBoard/1xx)		
DafIgnoreLSNibble	Ignore the least significant nibble	Any valid Analog, Digital, or Counter Channel
DafClearLSNibble	Clear the least significant nibble	Any valid Analog, Digital, or Counter Channel
DafShiftLSNibble	Shift the least significant nibble	Any valid Analog, Digital, or Counter Channel

Thermocouple Type Connected to Input		
DafTcTypeNone	No Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeJ	J Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeK	K Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeT	T Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeE	E Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeN28	N28 Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeN14	N14 Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeS	S Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeR	R Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcTypeB	B Type Thermocouple	Any DBK19, DBK52,DBK81-84,DBK90
DafTcCJC	CJC Channel	Any DBK19, DBK52,DBK81-84,DBK90
64 Channel Cards (DBK90 only)		
Daf64ChannelExp	Specifies 64-channel local addressing	DBK90 only

Setpoint flags	Setpoint Flags are used with /3000 Series devices only	
Definition	**Description**	
DafSetPointEnable	Enable Setpoint detection.	= 0x08000000
DafSetPointStatus	Enable the control status read in the scan list.	=0x8001

Note: Also see **daqADCSetSetpoints**   (for /3000 Series Devices Only)

gains—(DaqAdcGain)

Base Unit

Definition	Description	Applies to …
DgainX1	Main Unit-signal gain x 1	Daq products.
DgainX2	Main Unit-signal gain x 2	Daq products.
DgainX4	Main Unit-signal gain x 4	Daq products.
DgainX8	Main Unit-signal gain x 8	Daq products.
DgainX16	Main Unit-signal gain x 16	DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series , DaqBoard/2000 Series, DaqBoard/1000 Series and cPCI DaqBoard/2000c Series products.
DgainX32	Main Unit-signal gain x 32	DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series , DaqBoard/2000 Series, DaqBoard/1000 Series and cPCI DaqBoard/2000c Series products.
DgainX64	Main Unit-signal gain x 64	DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series , DaqBoard/2000 Series, DaqBoard/1000 Series and cPCI DaqBoard/2000c Series products.
WgcX1	Main Unit-signal gain x 1	WaveBook products.
WgcX2	Main Unit-signal gain x 2	WaveBook products.
WgcX5	Main Unit-signal gain x 5	WaveBook products.
WgcX10	Main Unit-signal gain x 10	WaveBook products.
WgcX20	WBK Module - signal gain x 20	WBK11, WBK12, WBK13, and WBK14 .
WgcX50	WBK Module - signal gain x 50	WBK11, WBK12, WBK13, and WBK14 .
WgcX100	WBK Module - signal gain x 100	WBK11, WBK12, WBK13, and WBK14 .
WgcX200	WBK Module - signal gain x 200	WBK10A with WBK11, WBK12, or WBK13 installed; and WBK14.

DBK4-Filter

Definition	Description
Dbk4FilterX1	DBK4-DBK4-LPF is on, signal gain x 1
Dbk4FilterX10	DBK4-LPF is on, signal gain x 10
Dbk4FilterX100	DBK4-LPF is on, signal gain x 100
Dbk4FilterX1000	DBK4-LPF is on, signal gain x 1000
Dbk4FilterX2	DBK4-LPF is on, signal gain x 2
Dbk4FilterX20	DBK4-LPF is on, signal gain x 20
Dbk4FilterX200	DBK4-LPF is on, signal gain x 200
Dbk4FilterX2000	DBK4-LPF is on, signal gain x 2000
Dbk4FilterX4	DBK4-LPF is on, signal gain x 4
Dbk4FilterX40	DBK4-LPF is on, signal gain x 40
Dbk4FilterX400	DBK4-LPF is on, signal gain x 400
Dbk4FilterX4000	DBK4-LPF is on, signal gain x 4000
Dbk4FilterX8	DBK4-LPF is on, signal gain x 8
Dbk4FilterX80	DBK4-LPF is on, signal gain x 80
Dbk4FilterX800	DBK4-LPF is on, signal gain x 800
Dbk4FilterX8000	DBK4-LPF is on, signal gain x 8000

DBK4-Bypass[1]

Definition	Description
Dbk4BypassX1_583	DBK4-LPF is off, signal gain x 1
Dbk4BypassX15_83	DBK4-LPF is off, signal gain x 10
Dbk4BypassX158_3	DBK4-LPF is off, signal gain x 100
Dbk4BypassX1583	DBK4-LPF is off, signal gain x 1000
Dbk4BypassX3_166	DBK4-LPF is off, signal gain x 2
Dbk4BypassX31_66	DBK4-LPF is off, signal gain x 20
Dbk4BypassX316_6	DBK4-LPF is off, signal gain x 200
Dbk4BypassX3166	DBK4-LPF is off, signal gain x 2000
Dbk4BypassX6_332	DBK4-LPF is off, signal gain x 4
Dbk4BypassX63_32	DBK4-LPF is off, signal gain x 40
Dbk4BypassX633_2	DBK4-LPF is off, signal gain x 400
Dbk4BypassX6332	DBK4-LPF is off, signal gain x 4000
Dbk4BypassX12_664	DBK4-LPF is off, signal gain x 8
Dbk4BypassX126_64	DBK4-LPF is off, signal gain x 80
Dbk4BypassX1266_4	DBK4-LPF is off, signal gain x 800
Dbk4BypassX12664	DBK4-LPF is off, signal gain x 8000

DBK7

Dbk7X1	DBK7- signal gain x 1 (Use with DBK55 also)

DBK8

Dbk8X1	DBK8- signal gain x 1

[1] Bypassing the filter adds a hardware gain of x 1.583

DBK9	
Dbk9VoltageA	DBK9-RTD reading A
Dbk9VoltageB	DBK9-RTD reading B
Dbk9VoltageC	DBK9-RTD reading C
Dbk9VoltageD	DBK9-RTD reading D

DBK12	
Dbk12X1	DBK12-signal gain x 1
Dbk12X2	DBK12-signal gain x 2
Dbk12X4	DBK12-signal gain x 4
Dbk12X8	DBK12-signal gain x 8
Dbk12X16	DBK12-signal gain x 16
Dbk12X32	DBK12-signal gain x 32
Dbk12X64	DBK12-signal gain x 64

DBK13	
Definition	**Description**
Dbk13X1	DBK13-signal gain x 1
Dbk13X10	DBK13-signal gain x 10
Dbk13X100	DBK13-signal gain x 100
Dbk13X1000	DBK13-signal gain x 1000
Dbk13X2	DBK13-signal gain x 2
Dbk13X20	DBK13-signal gain x 20
Dbk13X200	DBK13-signal gain x 200
Dbk13X2000	DBK13-signal gain x 2000
Dbk13X4	DBK13-signal gain x 4
Dbk13X40	DBK13-signal gain x 40
Dbk13X400	DBK13-signal gain x 400
Dbk13X4000	DBK13-signal gain x 4000
Dbk13X8	DBK13-signal gain x 8
Dbk13X80	DBK13-signal gain x 80
Dbk13X800	DBK13-signal gain x 800
Dbk13X8000	DBK13-signal gain x 8000

DBK14 Bipolar	
Definition	**Description**
Dbk14BiCJC	DBK14-Bipolar CJC
Dbk14BiTypeJ	DBK14-Bipolar Type J TC
Dbk14BiTypeK	DBK14-Bipolar Type K TC
Dbk14BiTypeT	DBK14-Bipolar Type T TC
Dbk14BiTypeE	DBK14-Bipolar Type E TC
Dbk14BiTypeN28	DBK14-Bipolar Type N28 TC
Dbk14BiTypeN14	DBK14-Bipolar Type N14 TC
Dbk14BiTypeS	DBK14-Bipolar Type S TC
Dbk14BiTypeR	DBK14-Bipolar Type R TC
Dbk14BiTypeB	DBK14-Bipolar Type B TC

DBK14 Unipolar	
Definition	**Description**
Dbk14UniCJC	DBK14-Unipolar CJC
Dbk14UniTypeJ	DBK14-Unipolar Type J TC
Dbk14UniTypeK	DBK14-Unipolar Type K TC
Dbk14UniTypeT	DBK14-Unipolar Type T TC
Dbk14UniTypeE	DBK14-Unipolar Type E TC
Dbk14UniTypeN28	DBK14-Unipolar Type N28 TC
Dbk14UniTypeN14	DBK14-Unipolar Type N14 TC
Dbk14UniTypeS	DBK14-Unipolar Type S TC
Dbk14UniTypeR	DBK14-Unipolar Type R TC
Dbk14UniTypeB	DBK14-Unipolar Type B TC

DBK15 Bipolar	
Definition	**Description**
Dbk15BiX1	DBK15-Bipolar, signal gain x 1
Dbk15BiX2	DBK15-Bipoalr, signal gain x 1

DBK15 Unipolar	
Dbk15UniX1	DBK15-Unipolar, signal gain x 1
Dbk15UniX2	DBK15-Unipolar, signal gain x 1

DBK16	
Definition	**Description**
Dbk16ReadBridge	DBK16-Read String Gage
Dbk16SetOffset	DBK16-Read Offset Trimpot
Dbk16SetScalingGain	DBK16-Read Scaling Trimpot
Dbk16SetInputGain	DBK16-Read Input Trimpot

DBK18	
Definition	**Description**
Dbk18X1	DBK18-signal gain x 1

DBK19 Bipolar[1]	
Definition	**Description**
Dbk19BiCJC	DBK19-Bipolar CJC
Dbk19BiTypeJ	DBK19-Bipolar Type J TC
Dbk19BiTypeK	DBK19-Bipolar Type K TC
Dbk19BiTypeT	DBK19-Bipolar Type T TC
Dbk19BiTypeE	DBK19-Bipolar Type E TC
Dbk19BiTypeN28	DBK19-Bipolar Type N28 TC
Dbk19BiTypeN14	DBK19-Bipolar Type N19 TC
Dbk19BiTypeS	DBK19-Bipolar Type S TC
Dbk19BiTypeR	DBK19-Bipolar Type R TC
Dbk19BiTypeB	DBK19-Bipolar Type B TC

DBK19 Unipolar	
Definition	**Description**
Dbk19UniCJC	DBK19-Unipolar CJC
Dbk19UniTypeJ	DBK19-Unipolar Type J TC
Dbk19UniTypeK	DBK19-Unipolar Type K TC
Dbk19UniTypeT	DBK19-Unipolar Type T TC
Dbk19UniTypeE	DBK19-Unipolar Type E TC
Dbk19UniTypeN28	DBK19-Unipolar Type N28 TC
Dbk19UniTypeN14	DBK19-Unipolar Type N19 TC
Dbk19UniTypeS	DBK19-Unipolar Type S TC
Dbk19UniTypeR	DBK19-Unipolar Type R TC
Dbk19UniTypeB	DBK19-Unipolar Type B TC

DBK42	
Definition	**Description**
Dbk42X1	DBK42-signal gain x 1

DBK43/43A	
Definition	**Description**
Dbk43ReadBridge	DBK43-Read String Gage
Dbk43SetOffset	DBK43-Read Offset Trimpot
Dbk43SetScalingGain	DBK43-Read Scaling Trimpot

DBK44	
Definition	**Description**
Dbk44X1	DBK44-signal gain x 1

DBK50	
Definition	**Description**
Dbk50Range0	DBK50-signal gain x 1
Dbk50Range10	DBK50-signal gain x 0.5
Dbk50Range100	DKB50-siganl gain x 0.05
Dbk50Range300	DBK50-signal gain x 0.06

DBK51	
Definition	**Description**
Dbk51Range0	DBK51-signal gain x 1
Dbk51Range100mV	DBK51-signal gain x 50
Dbk51Range1	DBK51-signal gain x 5
Dbk51Range10	DBK51-signal gain x 0.5

DBK52 Bipolar[1]	
Definition	**Description**
Dbk52BiCJC	DBK52-Bipolar CJC
Dbk52BiTypeJ	DBK52-Bipolar Type J TC
Dbk52BiTypeK	DBK52-Bipolar Type K TC
Dbk52BiTypeT	DBK52-Bipolar Type T TC
Dbk52BiTypeE	DBK52-Bipolar Type E TC
Dbk52BiTypeN28	DBK52-Bipolar Type N28 TC
Dbk52BiTypeN14	DBK52-Bipolar Type N52 TC
Dbk52BiTypeS	DBK52-Bipolar Type S TC
Dbk52BiTypeR	DBK52-Bipolar Type R TC
Dbk52BiTypeB	DBK52-Bipolar Type B TC

DBK52 Unipolar follows.

[1] When using the DBK19 and DBK52 with 10 V devices such as DaqBook/2000 Series, DaqBoard/2000 Series, DaqBoard/1000 Series, DaqScan/2000 Series, DaqLab/2000 Series, cPCI DaqBoard/2000c Series and Daq PC Cards, add four to the gain code.

DBK52 Unipolar	
Definition	**Description**
Dbk52UniCJC	DBK52-Unipolar CJC
Dbk52UniTypeT	DBK52-Unipolar Type T TC
Dbk52UniTypeE	DBK52-Unipolar Type E TC
Dbk52UniTypeN28	DBK52-Unipolar Type N28 TC
Dbk52UniTypeN14	DBK52-Unipolar Type N19 TC
Dbk52UniTypeS	DBK52-Unipolar Type S TC
Dbk52UniTypeR	DBK52-Unipolar Type R TC
Dbk52UniTypeB	DBK52-Unipolar Type B TC
DBK55	
Definition	**Description**
Dbk55X1	DBK55- signal gain x1
DBK65	
Definition	**Description**
Dbk65X1	DBK65 Bipolar X1 Gain
Dbk65X2	DBK65 Bipolar X2 Gain
Dbk65X4	DBK65 Bipolar X4 Gain
Dbk65X8	DBK65 Bipolar X8 Gain
Dbk65X16	DBK65 Bipolar X16 Gain
Dbk65X32	DBK65 Bipolar X32 Gain
Dbk65X64	DBK65 Bipolar X64 Gain
DBK80	
Definition	**Description**
Dbk80X1	DBK80 Bipolar X1 Gain
Dbk80X2	DBK80 Bipolar X2 Gain
Dbk80X4	DBK80 Bipolar X4 Gain
Dbk80X8	DBK80 Bipolar X8 Gain
Dbk80X16	DBK80 Bipolar X16 Gain
Dbk80X32	DBK80 Bipolar X32 Gain
Dbk80X64	DBK80 Bipolar X64 Gain
DBK85	
Definition	**Description**
Dbk85X1	DBK85 Bipolar X1 Gain
Dbk85X2	DBK85 Bipolar X2 Gain
Dbk85X4	DBK85 Bipolar X4 Gain
Dbk85X8	DBK85 Bipolar X8 Gain
Dbk85X16	DBK85 Bipolar X16 Gain
Dbk85X32	DBK85 Bipolar X32 Gain
DBK81-DBK84 Bipolar Only	
Definition	**Description**
Dbk81CJC	DBK81-84 Bipolar CJC
Dbk81TypeJ	DBK81-84 Bipolar Type J TC
Dbk81TypeK	DBK81-84 Bipolar Type K TC
Dbk81TypeT	DBK81-84 Bipolar Type T TC
Dbk81TypeE	DBK81-84 Bipolar Type E TC
Dbk81TypeN28	DBK81-84 Bipolar Type N28 TC
Dbk81TypeN14	DBK81-84 Bipolar Type N52 TC
Dbk81TypeS	DBK81-84 Bipolar Type S TC
Dbk81TypeR	DBK81-84 Bipolar Type R TC
Dbk81TypeB	DBK81-84 Bipolar Type B TC
DBK90 Bipolar Only	
Definition	**Description**
Dbk90CJC	DBK90 Bipolar CJC
Dbk90TypeJ	DBK90 Bipolar Type J TC
Dbk90TypeK	DBK90 Bipolar Type K TC
Dbk90TypeT	DBK90 Bipolar Type T TC
Dbk90TypeE	DBK90 Bipolar Type E TC
Dbk90TypeN28	DBK90 Bipolar Type N28 TC
Dbk90TypeN14	DBK90 Bipolar Type N52 TC
Dbk90TypeS	DBK90 Bipolar Type S TC
Dbk90TypeR	DBK90 Bipolar Type R TC
Dbk90TypeB	DBK90 Bipolar Type B TC

[1] When using the DBK19 and DBK52 with 10 V devices such as DaqBook/2000 Series, DaqBoard/2000 Series, DaqBoard/1000 Series, DaqScan/2000 Series, DaqLab/2000 Series, cPCI DaqBoard/2000c Series and Daq PC Cards, add four to the gain code.

Returns

`DerrNotCapable`	No digital or counter
`DerrInvGain`	Invalid gain
`DerrInvChan`	Invalid channel
`DerrNoError`	No error

For more details on error messages, please refer to the Daq Error Table.

This function may not return an error immediately.
`daqAdcArm` or `daqAdcSetRate` may return an error from the scan list.

Function Usage

Scan Sequence Configuration

The `daqAdcSetScan` function is the method by which the scan sequence is programmed. The channel scan is comprised of all channels configured for scanning. When a user application sets each of the values in the channels, gains and flags arrays and passes these array pointers to `daqAdcSetScan`, the driver interprets each array element as a configuration variable for the corresponding scan element. The following table demonstrates this relationship:

Array Location	Scan Location	Channels Array Elements	gains Array Elements	flags Array Elements
0	0	Channels(0)	gains(0)	flags(0)
1	1	Channels(1)	gains(1)	flags(1)
2	2	Channels(2)	gains(2)	flags(2)
3	3	Channels(3)	gains(3)	flags(3)
4	4	Channels(4)	gains(4)	flags(4)
chanCount+1	chanCount+1	channels(chanCount+1)	gains(chanCount+1)	flags(chanCount+1)

As many as 512 channel entries can be made in the acquisition scan group configuration. Any analog input channel can be included in the scan group configuration at any valid gain setting. Scan group configuration may be composed of local or expansion channels (and, for the DaqBook/DaqBoard, the high-speed digital I/O port). Each of the parameters that define the scan group are described in more detail below:

The `channels` parameter is a pointer to an array of up to 512 channel values. Each entry represents a channel number in the scan group configuration. Channels can be entered multiple times at the same or different gain and flags settings.

The `gains` parameter is a pointer to an array of up to 512 gain settings. Each entry in the gain array represents the gain to be used with the corresponding channel entry. Gain entry can be any valid gain setting for the corresponding channel.

The `flags` parameter is a pointer to an array of up to 512 channel flag settings. Each entry in the flag array represents a 4-byte-wide bit map of channel configuration settings for the corresponding channel entry. The flags can be used to set channel-specific configuration settings such as polarity [and channel type for DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series, DaqBoard/2000 Series, DaqBoard/1000 Series and cPCI DaqBoard/2000c Series boards]. The channel may require further configuration. If this is the case, then see the `daqSetOption` function for further channel configuration instructions.

The chanCount parameter is not a configuration parameter in the same sense as the channels, gains and flags parameters—it simply represents the total number of channels in the scan group configuration. This number also represents the number of entries in each of the channels, gains and flags arrays.

To illustrate how the scan group might be configured, suppose that we would like to configure a scan sequence in the following order:

Scan Location 0 --- an analog, bipolar, and single-ended channel with gain x 1, using channel 3
Scan Location 1 --- a 16-bit HS digital channel on P3
Scan Location 2 --- the lower 16-bits on counter 0 used (with scan location 3) as a cascaded 32-bit
    counter for totalizing
Scan Location 3 --- the upper 16-bits on counter 2 used (with scan location 2) as a cascaded 32-bit
    counter for totalizing

The following table shows how an array with the above specifications might be defined. Note that, in this example, there are 4 channel scan locations (chanCount=4) rather than 3 since the 32-bit cascaded counter occupies two scan locations.

Array Location	Scan Location	channels Array Elements	gains Array Elements	flags Array Elements
0	0	channels(0) = 3	gains(0) = DgainX1	flags(0) = DafAnalog + DafBipolar + DafSingleEnded
1	1	channels(1) = 0	gains(1) = N/A	flags(1) = DafP3Local16
2	2	channels(2) = 0	gains(2) = N/A	flags(2) = DafCtr32Low + DafCtrTotalize
3	3	channels(3) = 2	gains(3) = N/A	flags(3) = DafCtr32High + DafCtrTotalize

Typical `flags` Settings

Although the `flags` parameter may be constructed using any of the defined `flags` values, the following table illustrates how specific channel configurations are typically defined:

Desired Channel Configuration	Flag1	Flag2	Flag3	Notes
Analog, Bipolar, SE	DafAnalog (Default)	DafBipolar	DafSingleEnded (Default)	Configures P1 local and expansion analog input channel as bipolar, single ended
Analog, Bipolar, DE	DafAnalog (Default)	DafBipolar	DafDifferential	Configures P1 local and expansion analog input channel as bipolar, differential
Analog, Unipolar, SE	DafAnalog (Default)	DafUnipolar (Default)	DafSingleEnded (Default)	Configures P1 local and expansion analog input channel as unipolar, single ended
Analog, Unipolar, DE	DafAnalog (Default)	DafUnipolar (Default)	DafDifferential	Configures P1 local and expansion analog input channel as unipolar, differential
Digital (8 bit), Local P2	DafP2Local8	N/A	N/A	Configures P2 local (8255) digital input channel bank (8-bit)
Digital (8-bit), Expansion P2	DafP2Exp8	N/A	N/A	Configures P2 expansion (8255) digital input channel bank (8-bit)
Digital (16-bit), Local P3	DafP3Local16	N/A	N/A	Configures P3 local (HS Digital) digital input channel bank (16-bit)
Counter (16-bit), Local P3	DafCtr16	DafCtrPulse	DafCtrRisingEdge (Default)	Configures 16-bit P3 counter channel for pulse counting on rising edge of signal
Counter (16-bit), Local P3	DafCtr16	DafCtrTotalize	DafCtrRisingEdge (Default)	Configures 16-bit P3 counter channel for totalize counting on rising edge of signal
Counter (16-bit), Local P3	DafCtr16	DafCtrPulse	DafCtrFallingEdge	Configures 16-bit P3 counter channel for pulse counting on falling edge of signal
Counter (16-bit), Local P3	DafCtr16	DafCtrTotalize	DafCtrFallingEdge	Configures 16-bit P3 counter channel for totalize counting on falling edge of signal
Counter (32-bit-Low) Local P3	DafCtr32Low	DafCtrTotalize	DafCtrRisingEdge (Default)	Configures low word of 32-bit counter on P3 for totalizing on rising edge of signal. Must be paired with a 32-bit counter high word
Counter (32-bit-High) Local P3	DafCtr32High	DafCtrTotalize	DafCtrRisingEdge (Default)	Configures high word of 32-bit counter on P3 for totalizing on rising edge of signal. Must be paired with a 32-bit counter low word

Desired Channel Configuration	Flag1	Flag2	Flag3	Notes
Counter (32-bit-Low) Local P3	DafCtr32Low	DafCtrPulse	DafCtrRisingEdge (Default)	Configures low word of 32-bit counter on P3 for pulse counting on rising edge of signal. Must be paired with a 32-bit counter high word
Counter (32-bit-High) Local P3	DafCtr32High	DafCtrPulse	DafCtrRisingEdge (Default)	Configures high word of 32-bit counter on P3 for pulse counting on rising edge of signal. Must be paired with a 32-bit counter high word
Counter (32-bit-Low) Local P3	DafCtr32Low	DafCtrTotalize	DafCtrFallingEdge	Configures low word of 32-bit counter on P3 for totalizing on falling edge of signal. Must be paired with a 32-bit counter high word
Counter (32-bit-High) Local P3	DafCtr32High	DafCtrTotalize	DafCtrFallingEdge	Configures high word of 32-bit counter on P3 for totalizing on falling edge of signal. Must be paired with a 32-bit counter high word
Counter (32-bit-Low) Local P3	DafCtr32Low	DafCtrPulse	DafCtrFallingEdge	Configures low word of 32-bit counter on P3 for pulse counting on falling edge of signal. Must be paired with a 32-bit counter high word
Counter (32-bit-High) Local P3	DafCtr32High	DafCtrPulse	DafCtrFallingEdge	Configures high word of 32-bit counter on P3 for pulse counting on falling edge of signal. Must be paired with a 32-bit high

For digital and counter channel `flags` definitions, the corresponding element in the `gains` array will be ignored. Those flag settings who are marked as `default` will take on the default value if not explicitly set.

Other `flags` Settings

There are additional flags that can be added to any `flags` construct for more specific channel configurations:

Desired Channel Configuration	Flag	Notes
Unsigned Data Representation	DafUnsigned (Default)	Data returned for channel ranges from 0 to 65,535
Signed Data Representation	DafSigned	Data returned for channel ranges from –32,768 to +32,767
5 μs Sampling Interval *	DafSettle5us (Default)	Sets the sampling interval for the channel to 5 μs
10 μs Sampling Interval *	DafSettle10us	Sets the sampling interval for the channel to 10 μs
1000 μs (1 ms) Sampling Interval *	DafSettle1ms	Sets the sampling interval for the channel to 1000 μs, (1 ms)
Ignore least significant 4-bits	DafIgnoreLSNibble (Default)	The least significant 4-bits represents a channel tag on a 12 bit device and should be ignored (DaqBook/DaqBoard/1xx series products only)
Clear least significant 4-bits	DafClearLSNibble	The least significant 4-bits will be set to 0
Shift least significant 4-bits	DafShiftLSNibble	The least significant 4-bits will be shifted left by 4 places (making a 12-bit number into a 16-bit number) (DaqBook/DaqBoard/1xx series products only)

* Channel configurations for 5 μs, 10 μs and 1000 μs do not apply to DaqBoard/500 Series devices.

Prototypes

C/C++

```
daqAdcSetScan(DaqHandleT handle, PDWORD channels, DaqAdcGain *gains,
PDWORD flags, DWORD chanCount);
```

Visual BASIC

```
VBdaqAdcSetScan&(ByVal handle&, channels&, gains&, flags&, ByVal
chanCount&)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK53_54EX.CPP, DAQEX.FRM (VB)
```

daqAdcSetSetpoints

Format

```
daqAdcSetSetpoints(handle, LimitA, LimitB, reserved, setpointFlags,
      setpointOutput, output1, output2, outputMask1,  outputMask2,
      setpointCount)
```

Purpose

`daqAdcSetSetpoints` allows the setting of up to 16 setpoints for a device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	The handle to the device for which to set the option
limitA	PFLOAT	Limit A specifies a value against which to determine if setpoint criteria are met
limitB	PLFOAT	Limit B specifies a value against which to determine if setpoint criteria are met
reserved	PFLOAT	Reserved – reserved for future use
setpointFlags	DWORD	The value of the option to set
setpointOutput	DWORD	Output source or action: None, P2 Port C, Timer0-1, DAC0-3
output1	PFLOAT	The value to output when the setpoint criteria are met
output2	PFLOAT	The value to output when the setpoint criteria are NOT met
outputMask1	PFLOAT	Output mask for output1 – for P2 Port C only
outputMask2	PFLOAT	Output mask for output2 – for P2 Port C only
setpointCount	DWORD	Number of setpoints to configure (16 max)

Parameter Values

`handle:` obtained from the `daqOpen` function

`limitA:` any valid value for the input channel used for the setpoint

`limitB:` any valid value for the input channel used for the setpoint

`reserved:` not specified

`setpointFlags:` see following table

`setpointOutput:` see following table

`output1:` any valid value for the output channel used for the setpoint

`output2:` any valid value for the output channel used for the setpoint

`setpointCount:` from 0 to 16

Parameter Type Definitions

setpointflags - DaqSetpointFlags	
Definition	**Description**
DsfEqualLimitA	Setpoint criteria: Input Channel = limit A
DsfLessThanLimitA	Setpoint criteria: Input Channel < limit A
DsfGreaterThanLimitB	Setpoint criteria: Input Channel > limit B
DsfOutsideLimits	Setpoint criteria: Input Channel > limit A OR Input Channel < limit B
DsfInsideLimits	Setpoint criteria: Input Channel < limit A AND Input Channel > limit B
DsfHysteresis	Setpoint criteria: Input Channel > limit A then output1 Input Channel < limit B then output2
DsfUpdateOnTrueOnly	If criteria met then output1
DsfUpdateOnTrueAndFalse	If criteria met then output1 else output2

For outside or inside window limits (`DsfOutsideLimits, DsfInsideLimits`) and for hysteresis (`DsfHysteresis`) <u>Limit A must be greater than Limit B</u>. If this criteria is not met `DerrSetpointLimits` error (error code 164) will be returned.

setpointOutput - DaqSetpointOutputT	
Definition	**Description**
DsoNone	Perform no outputs
DsoP2PortC	Output to P2 Port C when criteria met
DsoDAC0	Output to DAC0 when criteria met (must have device with DAC0)
DsoDAC1	Output to DAC1 when criteria met (must have device with DAC1)
DsoDAC2	Output to DAC2 when criteria met (must have device with DAC2)
DsoDAC3	Output to DAC3 when criteria met (must have device with DAC3)
DsoTMR0	Output to Timer0 when criteria met
DsoTMR1	Output to Timer1 when criteria met

Function Usage

The daqAdcSetSetpoints function may be used to setpoints for those devices which support setpoint controlled outputs such as the DaqBoard/3000 and PersonalDaq/3000 Series devices. Up to 16 separate setpoints are supported and an input channel can have more than one setpoint associated with it. Any input channel can be configured to use a setpoint including analog, digital or counter channels. However, if the input channel is a TC channel, there will be no compensation or linearization of the channel applied when evaluating the setpoint and the setpoint input channel must be specified in voltage. The daqAdcSetSetpoints function only configures the setpoints. See daqAdcSetScan function for information regarding how to associate an input channel to a setpoint.

Each setpoint requires an input channel, a detection criteria and and output or action to be performed when the detection critera are met. As mentioned before, the input channel can be an analog voltage, digital or counter channel. If a counter channel is used and the counter channel is a 32-bit counter the setpoint can only be evaluated against either the high or low order word (or 16-bits) of the counter. See the See daqAdcSetScan function for information regarding how to configure the high and low order words in the scan list.

While data is being collected by the device the input signal is compared to the detection criteria mentioned above. When given criteria are met the result is a condition which is driven High(True) or Low(False) depending upon current state of the signal. The rate at which the condition is updated depends upon the overall scan rate selected for the acquisition. See See daqAdcSetRate function for more details on setting up the scan rate. So the detection of change of input state against the criteria can be no faster than the overall acquisition scan rate.

The output target or channel can be the P2 Port C, Timer0, Timer1 or any of the onboard DAC. In addition, the user may select no outputs to be peformed. When the condition is driven High(True) or Low(False) the UpdateOnTrueOnly and the UpdateOnTrueAndFalse flags determine which output value will be used to update the output channel.

If Timer0 or Timer1 (TMR0 or TMR1) is the output channel , then the value you enter is that of the 1 MHz clock divisor. The following table shows 5 examples of divisor value and associated Timer Output Frequency.

Divisor	Timer Output Frequency
1	1 MHz
100	10 kHz
1000	1 kHz
5000	200 Hz
65535	15.259 Hz

If using UpdateOnTrueOnly, then output1 will be updated to the output channel each time the criteria is met. If using UpdateOnTrueAndFalse, then output1 will be updated to the output channel each time the criteria is met. In addition output2 will be updated to the output channel each time the criteria is NOT met.

See your device hardware manual for a more complete description and theory of operations for the setpoint feature.

Returns

> **Note!** For outside or inside window limits (`DsfOutsideLimits, DsfInsideLimits`) and for hysteresis (`DsfHysteresis`) <u>Limit A must be greater than Limit B</u>. If this criteria is not met `DerrSetpointLimits` error (error code 164) will be returned.

```
DerrSetpointLimits    Setpoint comparison, limit B is greater than Limit A.

DerrNoError           No error
```

Prototypes

C/C++

```
daqAdcSetSetpoints(DaqHandleT handle, PFLOAT LimitA, PFLOAT LimitB,
PFLOAT reserved,PDWORD setpointFlags,DaqSetpointOutputT *setpointOutput,
PFLOAT output1,PFLOAT output2,PFLOAT outputMask1,PFLOAT outputMask2, DWORD
setpointCount);
```

Visual BASIC

```
VBdaqAdcSetSetpoints&(ByVal handle&, ByRef LimitA!, ByRef LimitB!, ByRef
Reserved!, ByRef SetpointFlags&, ByRef SetpointOutput&, ByRef Output1!,
ByRef Output2!, ByRef OutputMask1!, ByRef OutputMask2!, ByVal
SetpointCount&)
```

Program References

```
Daq3KEnhancedCounters.cpp
Daq3KEncoderExample.cpp
Daq3KMultiChannelSetpointTrig.cpp
Daq3KSetpointExample.cpp
```

daqAdcSetTrig

Format

daqAdcSetTrig(handle, triggerSource, rising, level, hysteresis, channel)

Purpose

daqAdcSetTrig configures the device for enhanced triggering.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the ADC acquisition trigger will be configured
triggerSource	DaqAdcTriggerSource	Sets the trigger source
rising	BOOL	Boolean flag to indicate the rising or falling edge of the trigger source
level	WORD	The trigger level (in A/D counts) for an analog level trigger
hysteresis	WORD	Hysteresis value for an analog level trigger (if selected)
channel	DWORD	Channel for which the analog level trigger will be detected (if selected)

Parameter Values

handle: obtianed from the daqOpen function

triggerSource: see table below

rising: valid values are either true ($\neq 0$) or false ($= 0$)

level: valid values range from 0 to 65,535

hysteresis: valid values range from 0 to 65,535

channel: valid values range from 0 to 271

Parameter Type Definitions

trigSource-(DaqAdcTriggerSource)		
Definition	**Devices**	**Description**
DatsImmediate	All	Post-trigger data acquisition begins immediately upon invocation of the daqAdcArm command (no pre-trigger data acquisition is possible with this trigger source)
DatsSoftware	All	Post-trigger data acquisition begins upon a software command issued by the calling application (see daqAdcSoftTrig)
DatsAdcClock	All	Post-trigger data acquisition begins immediately upon invocation detection of the Adc Clock pulse being driven.
DatsGatedAdcClock	DaqBooks DaqBoard(ISA)	Post-trigger data acquisition begins immediately upon invocation detection of the Adc Clock pulse being driven.
DatsExternalTTL	All	Post-trigger data acquisition begins on the selectable edge of an external TTL signal. Refer to the applicable user's manual for pinouts showing the location of the external TTL signal connection.
DatsHardwareAnalog	All	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)
DatsSoftwareAnalog	All	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)

Returns

DaqError   See Daq Error Table.

Function Usage

This function has been obsoleted by the **daqSetTriggerEvent** function, and his presented here only as a reference. See **daqSetTriggerEvent** for more details.

The `daqAdcSetTrig` function sets and arms the trigger of the A/D converter. Several trigger sources and several mode flags can be used for a variety of acquisitions. `daqAdcSetTrig` will stop current acquisitions, empty acquired data, and arm the device using the specified trigger. All of this can be accomplished using the `daqSetTriggerEvent` function.

Prototypes

C/C++

```
daqAdcSetTrig(DaqHandleT handle, DaqAdcTriggerSource triggerSource, BOOL
rising, WORD level, WORD hysteresis, DWORD channel);
```

Visual BASIC

```
VBdaqAdcSetTrig&(ByVal handle&, ByRef triggerSource&, ByVal rising&, ByVal
level%, ByVal hysteresis%, ByVal channel&)
```

Program References

```
None
```

daqAdcSetTrigEnhanced

Also See: **daqAdcSetAcq, daqAdcSetScan,daqAdcSetTrigEnhanced,daqSetTriggerEvent,
daqAdcSoftTrig**

Format

```
daqAdcSetTrigEnhanced(handle, trigSources, gains, adcRanges,
trigSensitivity, level, hysteresis, channels, chanCount, opStr)
```

Purpose

`daqAdcSetTrigEnhanced` configures the device for enhanced triggering.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the ADC acquisition trigger will be configured
trigSources	DaqAdcTriggerSource	A pointer to an array of trigger sources for each defined trigger channel
gains	DaqAdcGain	A pointer to an array of gains for each defined trigger channel
adcRanges	DaqAdcRangeT	A pointer to an array of polarity flag definitions for each defined channel
trigSensitivity	DaqEnhTrigSenseT	A pointer to an array of trigger sensitivity flags for each defined channel trigger source
level	PFLOAT	A pointer to an array of analog trigger levels for each defined trigger channel
hysteresis	PFLOAT	A pointer to an array of hysteresis values for each defined trigger channel
channels	PDWORD	A pointer to an array of trigger channels representing the actual trigger channels to trigger on
chanCount	DWORD	Number of trigger channels
opStr	char	A pointer to an array of characters which determines the interaction of the trigger channels

Parameter Values

`handle:` obtianed from the `daqOpen` function

`trigSources:` see table below

`gains:` see the ADC Gains Table

`adcRanges:` see table below

`trigSensitivity:` see table below

`level:` a pointer to an array of values ranging from –10.0 to +10.0 Volts

`hysteresis:` a pointer to an array of values ranging from –10.0 to +10.0 Volts

`channels:` a pointer to an array of values ranging from 0 to 71

`chanCount:` valid values range from 1 to 512

`opStr:` a pointer to an array characters which can be "+" or "-"

Parameter Type Definitions

trigSource-(DaqAdcTriggerSource)		
Definition	**Devices**	**Description**
DatsImmediate	All	Post-trigger data acquisition begins immediately upon invocation of the daqAdcArm command (no pre-trigger data acquisition is possible with this trigger source)
DatsSoftware	All	Post-trigger data acquisition begins upon a software command issued by the calling application (see daqAdcSoftTrig)
DatsExternalTTL	All	Post-trigger data acquisition begins on the selectable edge of an external TTL signal. Refer to the applicable user's manual for pinouts showing the location of the external TTL signal connection.
DatsHardwareAnalog	All	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)
DatsEnhancedTrig	WaveBooks	Post-trigger data acquisition begins upon the detection of the multiple channel trigger event defined by daqAdcSetTrigEnhanced
DatsSoftwareAnalog	All	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)
DatsDigPattern	Supported within most product series. *Refer to your product-specific user manual regarding applicability.*	Post-trigger data acquisition beings upon receiving a specified byte length digital pattern on a P2 digital port.
DatsPulse	WaveBook/516	Post-trigger data acquisition begins upon detection of a detection of a pulse of specified duration and magnitude on an analog input channel.

adcRanges-(DaqAdcRangeT)	
Definition	**Description**
DarUni0to10V	Sets acquisition range as unipolar from 0 to +10 Volt range
DarBiMinus5to5V	Sets acquisition range as bipolar from -5 to +5 Volt range
DarUniPolarDE	Sets acquisition range as unipolar differential
DarBiPolarDE	Sets acquisition range as bipolar differential
DarUniPolarSE	Sets acquisition range as unipolar single-ended
DarBiPolarSE	Sets acquisition range as bipolar single-ended

trigSensitivity-(DaqEnhTrigSenseT)	
Definition	**Description**
DatdRisingEdge	Trigger the channel on the rising edge of the signal at the specified level
DatdFallingEdge	Trigger the channel on the falling edge of the signal at the specified level
DatdAboveLevel	Trigger the channel when the signal is above the specified level
DatdBelowLowel	Trigger the channel when the signal is below the specified level
DatdRisingEdgeLatched	Trigger the channel on the rising edge of the signal at the specified level and latch the channel trigger event
DatdFallingEdgeLatched	Trigger the channel on the falling edge of the signal at the specified level and latch the channel trigger event
DatdAboveLevelLatched	Trigger the channel when the signal is above at the specified level and latch the channel trigger event
DatdBelowLevelLatched	Trigger the channel when the signal is below at the specified level and latch the channel trigger event

The `Latched` values are trigger sensitivities which indicate that the device will maintain the trigger event for the given channel, regardless of subsequent states of the input signal. After the channel has triggered, it will remain in a triggered state while the current acquisition is active. The non-latched trigger sensitivities will only indicate a channel trigger event while the input signal for the given channel is in the triggered state.

Returns

DaqError   See Daq Error Table.

Function Usage

Enhanced trigger configuration available through the `daqAdcSetTrigEnhanced` command allows the device to detect a trigger start event formed with multiple acquisition channels. The enhanced trigger start event may be defined as a combination of multiple acquisition channel event conditions that are connected logically by AND or OR.

To use multi-channel analog triggering, the `trigSource` parameter needs to be set to `DatsEnhTrigger`. Other trigger sources may used with the WaveBook as well --however, they can only act upon the first channel defined in the trigger channels array.

The `daqAdcSetTrigEnhanced` command sets the trigger start event only. The start event defines the method by which the acquisition is to begin collecting data. When the trigger event is satisfied, the device will begin to collect post-trigger data. The acquisition will terminate based on the acquisition `mode` parameter set in the `daqAdcSetAcq` command.

The trigger event is based on the channel trigger event for each channel in the trigger sequence. The total number of trigger channels is defined by the `chanCount` parameter. Each channel trigger configuration parameter is a pointer to an array of `chanCount` length and is defined as follows:

`channels` – a pointer to an array of actual scan channel numbers; these channels will be assigned corresponding trigger events.

`trigSources` – a pointer to an array of trigger sources for which the corresponding A/D trigger events will be configured on the corresponding channel as defined in the channels array.

`gains` – a pointer to an array of gains corresponding to the actual A/D channels with the corresponding A/D channel number as defined in the channels array. This setting applies only for analog trigger channels.

`adcRanges` – a pointer to an array of A/D ranges for the A/D channels as defined in the corresponding channels array. This setting applies only for analog trigger channels.

`hysteresis` – a pointer to an array of hysteresis values for each corresponding A/D channel as defined in the channels array.

`level`– a pointer to an array of level values for which, when satisfied, will set the trigger event for the corresponding channel as defined in the channels array.

`opStr`– a string that establishes the logical relationship between the individual channel trigger events and the global acquisition trigger condition. Currently, the string can be defined as "*" to perform an AND operation, or "+" to perform an OR operation on the individual channel trigger events. These two logical connections can be used to formulate global A/D trigger conditions. If the AND operation is selected, then *all* trigger channels must be in the triggered condition for the trigger event to occur. If the OR operation is selected, then *any* of the trigger channels can be in the triggered condition for the trigger event to occur.

`trigSensitivity` – an array of trigger sensitivity definitions for a specified trigger event on the corresponding channel as defined in the channels array.

Prototypes

C/C++

```
daqAdcSetTrigEnhanced(DaqHandleT handle, DaqAdcTriggerSource *trigSources,
DaqAdcGain *gains, DaqAdcRangeT *adcRanges, DaqEnhTrigSensT
*trigSensitivity, PFLOAT level, PFLOAT hysteresis, PDWORD channels, DWORD
chanCount, char *opStr);
```

Visual BASIC

```
VBdaqAdcSetTrigEnhanced&(ByVal handle&, ByRef trigSources&, ByRef gains&,
ByRef adcRanges&, ByRef trigSens&, ByRef levels!, ByRef hysteresis!, ByRef
chan&, ByVal CHANCOUNT&, ByRef opStr&)
```

Program References
```
None
```

daqAdcSoftTrig

Format

daqAdcSoftTrig(handle)

Purpose

daqAdcSoftTrig is used to send a software trigger command to the device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to which the ADC software trigger is to be applied

Parameter Values:

handle: obtained from the daqOpen function

Returns

DerrNoError   No error

Function Usage

The daqAdcSoftTrig function is a software trigger that can be used to initiate either a scan or an acquisition from a program after configuring the software trigger as the trigger source. This function may only be used if the trigger source for the acquisition has been set to DatsSoftware with the daqAdcSetTrig function.

daqAdcSoftTrig will override the DatsSoftwareAnalog, DatsDigPattern, and DatsCounter triggers as set by the daqSetTriggerEvent function on some devices. See daqSetTriggerEvent for more details.

Prototypes

C/C++

daqAdcSoftTrig(DaqHandleT handle);

Visual BASIC

VBdaqAdcSoftTrig&(ByVal handle&)

Program References

DQADCEX02.CPP, DQADCEX07.CPP

daqAdcTransferBufData

Format

```
daqAdcTransferBufData(handle, buf, scanCount, bufMask, retCount)
```

Purpose

`daqAdcTransferBufData` requests a transfer of `scanCount` scans from the driver allocated acquisition buffer (driver buffer) to the specified linear data retrieval buffer (`buf`). The driver buffer is configured with the `daqAdcTransferSetBuffer` function.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which the ADC buffer should be retrieved
buf	PWORD	Pointer to an application-supplied buffer in which to place the buffered data
scanCount	DWORD	Number of scans to retrieve from the acquisition buffer
bufMask	DaqAdcBufferXferMask	A mask-defining operation depending on the current state of the acquisition buffer
retCount	PDWORD	A pointer to the total number of scans returned, if any

Parameter Values

handle: obtained from the `daqOpen` function

buf: must be a valid pointer to memory whose size is at least equal to:

[(scanCount) * (the channel count) * (the sample size, normally 2 bytes)]

scanCount: valid values are 1 to length of the driver buffer (see `daqAdcTransferSetBuffer` function)

bufMask: see table below

retCount: a valid pointer to a long integer variable (4 bytes) in which the total number of scans returned (from 0 to `scanCount`) will be stored upon return of this function

Parameter Type Definitions:

bufMask-(DaqAdcBufferXferMask)	
Definition	**Description**
DabtmOldest	Retrieve oldest unread scans from driver buffer
DabtmNewest	Retrieve newest unread scans from driver buffer
DabtmWait	Wait for requested amount of data to become available
DabtmRetAvail	Return immediately with whatever data is available
DabtmNoWait	Return immediately, only retrieve if requested data is available
DabtmRetNotDone	Return immediately if still active

Function Usage

Data Retrieval Buffer

Upon completion of this function, the data retrieval buffer (`buf`) contains the requested data from the driver buffer (if the data was retrieved – see the "Data Retrieval Modes" section below). The buffer must be allocated by the application prior to calling this function. The diagram below illustrates the path of data transfer:
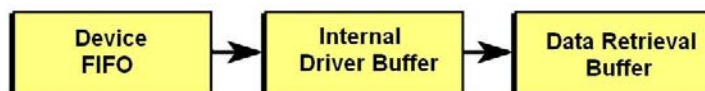
The `scanCount` parameter defines the number of scans to be retrieved (or the length of the data retrieval buffer in scans). The size of a scan is determined by the total number of channels in the scan group configuration (see the `daqAdcSetScan` function for further information on scan group configuration). Therefore the size of the data retrieval buffer to be allocated (in bytes) should be no smaller than:

$$scanCount \ * \ \text{scan size (number of channels)} \ * \ \text{sample size (normally 2 bytes)}$$

Note! If using packed mode with a WaveBook/512, the above calculation should be multiplied by ¾ to determine actual buffer allocation size required.

Data Retrieval Modes

The following values for the `bufMask` parameter set how recent the transferred data is and whether or not it remains in the buffer after transfer:

DabtmOldest – This value specifies that the specified number of the earliest scans be retrieved from the driver buffer. After they have been transferred, they will be removed from the buffer.

DabtmNewest – This value specifies that the specified number of the most recent scans be retrieved from the driver buffer. These scans will remain in the buffer after transfer.

The following values can be set concurrently with the values above. They specify the timing of the retrieval and the amount of data to transfer from the driver buffer:

DabtmWait – Instructs the function to wait until the requested number of scans (`scanCount`) are available in the driver-allocated acquisition buffer. When the requested number of scans are available, the function will return with `retCount` set to `scanCount` (the number of scans requested). Retrieved data will be returned in the memory referred to by the `buf` parameter. Returns if the `daqSetTimeout` value is exceeded.

DabtmNoWait – Instructs the function to return immediately if the specified number of scans (`scanCount`) are not available when the function is called. If the entire amount requested is not available, the function will return with no data and `retCount` will be set to 0. If the requested number of scans are available in driver buffer, the function will return with `retCount` set to `scanCount` (the number of scans requested). Retrieved data will be returned in the memory referred to by the `buf` parameter.

DabtmRetAvail – Instructs the function to return immediately, with any scans that are available in the driver-allocated acquisition buffer. The `retCount` parameter will return the total number of scans retrieved. **retCount** can return anything from 0 to `scanCount` (the number of scans requested). The retrieved data will be returned in the memory referred to by the `buf` parameter.

DabtmRetNotDone – Instructs the function to return immediately if the acquisition is still active without retrieving any data. If the acquisition is still active, the `retCount` parameter will return 0. If the acquisition is complete, then `retCount` can return anything from 0 to `scanCount` (the number of scans requested). The retrieved data will be returned in the memory referred to by the `buf` parameter.

Returns

`DerrNoError`   No error

Prototypes

C/C++

```
daqAdcTransferBufData(DaqHandleT handle, PWORD buf, DWORD scanCount,
DaqAdcBufferXferMask bufMask, PDWORD retCount);
```

Visual BASIC

```
VBdaqAdcTransferBufData(ByVal handle, buf%, ByVal scanCount&, ByVal
bufMask&, retCount&);
```

Program References

None

daqAdcTransferGetStat

Format

```
daqAdcTransferGetStat(handle, active, retCount)
```

Purpose

`daqAdcTransferGetStat` retrieves the current state of an acquisition transfer, and can be used to initiate transfers to the disk.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status is to be retrieved
active	PDWORD	A pointer to the transfer-state flags in the form of a bit mask
retCount	PDWORD	A pointer to the total number of ADC scans acquired (or available) in the current transfer

Parameter Values

handle: obtained from the `daqOpen` function

active: a valid pointer to a double word variable (4 bytes) in which the acqusition and transfer status flags will be stored upon return of this function; see table below for flag values

retCount: a valid pointer to a long integer variable (4 bytes) in which the total number of scans transferred will be stored (0 to `scanCount`) upon return of this function

Parameter Type Definitions

active	
Definition	**Description**
DaafAcqActive	An acquisition is active; the trigger may or may not yet have occurred but the acquisition has at least been armed
DaafAcqTriggered	The acquisition has been triggered, and post-trigger data is now being collected
DaafTransferActive	A buffer transfer is active; an acquisition may or may not be active but a buffer transfer has been enabled

Returns

DerrNoError   No error

Function Usage

Data Transfer Progress

The value that the `retCount` parameter returns depends upon the buffer management mode selected (see `daqAdcTransferSetBuffer` function for more information on buffer allocation modes):

User Buffer Mode (`DatmUserBuf`) – The `retCount` parameter will return the total amount of data written (in scans) to the user allocated buffer(s) during the duration of the acquisition. The following diagram illustrates how `retCount` is determined:
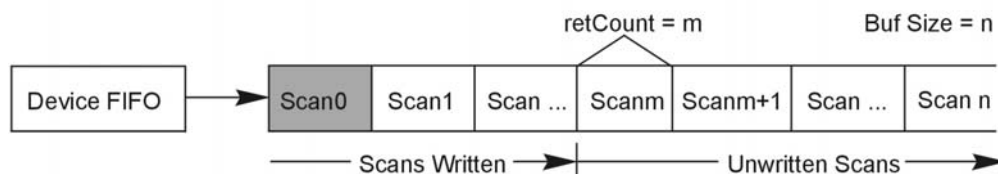
After the device's FIFO has written `m` scans to a linear user allocated buffer, `retCount` equals `m`. The current data write position (in scans) within the buffer is:

$$retCount \ mod \ scanCount$$

where the operation *mod* is defined as the integer remainder of performing an integer divide of `retCount` by `scanCount`. The `scanCount` parameter is set by the `daqAdcTransferSetBuffer` function.

> **If pre-trigger scan data has been configured (see `daqAdcSetAcq`), `retCount` will not return available pre-trigger scans until the acquisition has been triggered.**

Driver Buffer Mode (`DatmDriverBuf`) – The `retCount` parameter will return the total number of unread scans in the driver allocated circular buffer. In this case, the `retCount` parameter represents the amount of data that is currently available to be retrieved from the driver buffer. See `daqAdcTransferBufData` for more info on retrieving data from the driver buffer.

Example Acquisition States

The following table shows a number of different acquisition state combinations. These scenarios are representative of some typical acquisition states but do not necessarily represent all of the possible states. The table assumes a user-buffer mode with total scan count of 100000 scans.

States `(active)`	Progress `(retCount)`	Description

`DaafAcqActive +`
`DaafAcqTriggered +`
`DaafTransferActive`
	10,112	The acquisition is active and has been triggered and is currently collecting post-trigger data. A transfer to buffer is also active and a total of 10112 scans have been collected so far.

`DaafAcqActive +` `DaafTransferActive`	0	An acquisition has been armed but has not yet been triggered. If pre-trigger data collection has been configured (see `daqAdcSetAcq`), then pre-trigger data may be being collected (`retCount` will not return available pre-trigger scan counts until the trigger has occurred).
`DaafTransferActive`	0	A transfer has been configured and started (with the functions `daqAdcTransferSetBuffer` and `daqAdcTransferStart`), but the acquisition has not yet been armed. *The acquisition needs to be armed using* `daqAdcArm`.
`DaafAcqActive +` `DaafAcqTriggered`	10,112	The acquisition is active and has been triggered, but no transfer is currently active. A total of 10112 scans have been collected so far. This condition will normally only occur if transfers are performed with linear buffers while an acquisition is ongoing. *This state represents a possible loss of data if a transfer is not activated before the device overruns its internal buffering.*
`DaafAcqTriggered +` `DaafTransferActive`	98,000	An acquisition has been triggered, but is no longer active. A transfer is currently active. This sometimes will occur at the end of an acquisition if the acquisition is complete but data is still being transferred from the device.
`DaafAcqArmed`	0	The acquisition has been armed, but no transfer is currently active. This situation may result in loss of data unless the application initiates a transfer before the trigger occurs and the device overruns its internal buffering.
`DaafAcqTriggered`	100,000	The acquisition has triggered and has completed. The transfer is not active and 10000 scans have been collected. So, both the acquisition and the transfer are complete.

Prototypes

C/C++

```
daqAdcTransferGetStat(DaqHandleT handle, PDWORD active, PDWORD retCount);
```

Visual BASIC

```
VBdaqAdcTransferGetStat&(ByVal handle&, active&, retCount&)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK53_54EX.CPP, DAQEX.FRM (VB)
```

daqAdcTransferSetBuffer

Format

daqAdcTransferSetBuffer(handle, buf, scanCount, transferMask)

Purpose

daqAdcTransferSetBuffer configures transfer buffers for acquired data, and can also be used to configure the specified user- or driver-allocated buffers for subsequent acquisition transfers.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which an ADC transfer is to be performed
buf	PDWORD	Pointer to the buffer for which the acquired data is to be placed
scanCount	DWORD	The total length of the buffer (in scans)
transferMask	DWORD	Configures the buffer transfer mode

Parameter Values

handle: obtained from the daqOpen function

buf: must be a valid pointer to memory whose size is at least:

(scanCount) * scan size * 2 (the sample size, normally 2 bytes)

scanCount: valid values for buffer length are 1 to 4,294,967,295 scans, however, memory limitaions apply

transferMask: see table below

Parameter Type Definitions

transferMask	
Definition	**Description**
DatmCycleOn	Defines the buffer as a circular buffer
DatmCycleOff	Defines the buffer as a linear buffer
DatmUpdateSingle	Defines the update mode as single sample
DatmUpdateBlock	Defines the update mode as block update
DatmDriverBuf	Causes the driver to allocate the acquisition transfer buffer as a circular buffer
DatmUserBuf	Specifies a user-allocated and managed acquisition transfer buffer
DatmIgnoreOverruns	Ignores buffer overruns. Upon a buffer wrap, no error will be returned.

Returns

DerrNoError   No error

Function Usage

Transfer Buffer Location

The `buf` parameter is the address of the acquisition transfer buffer allocated by the application. If the application is supplying the buffer then this value must be an address to an adequately allocated buffer.

Transfer Buffer Length

The `scanCount` parameter is the total length of the transfer buffer in scans. The size of a scan is determined by the total number of channels in the scan group configuration (see `daqAdcSetScan` and `daqAdcSetMux` for further information on scan group configuration). Therefore, the buffer size to be allocated (in bytes):

$$scanCount * \text{scan size (number of channels)} * \text{sample size (normally 2 bytes)}$$

Note! **If using packed mode with a WaveBook/512, the above calculation should be multiplied by ¾ to determine actual buffer allocation size required.**

Transfer Buffer Settings

The character of the transfer buffer can be configured via the `transferMask` parameter. This parameter is a bit mask parameter--it can take on a number of settings, each dependant on how the values are joined logically with either OR or AND (see the "Mask and Flag Definitions" section). Among other things, the `transferMask` parameter specifies the update, layout/usage, and allocation modes of the buffer. The parameter's possible values are defined as follows:

DatmCycleOn – This parameter value defines the buffer as a circular buffer in buffer-cycle mode. This allows the transfer to continue when the end of the transfer buffer is reached by wrapping the transfer of acquisition data back to the beginning of the buffer, as shown in the diagram below.
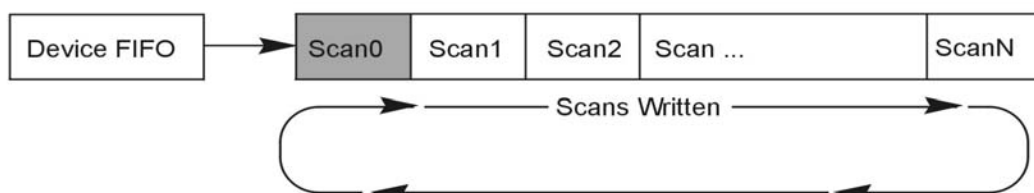
In the circular buffer mode, the acquisition transfer buffer will continue to be wrapped until the post-trigger count has been reached (specified by `daqAdcSetAcq`) or the transfer/acquisition is halted by the application (with the functions `daqAdcTransferStop` and `daqAdcDisarm`).

DatmCycleOff(`default`) – This parameter value defines the buffer as a linear buffer. This causes the transfer to continue to the end of the transfer buffer, at which point it will terminate.
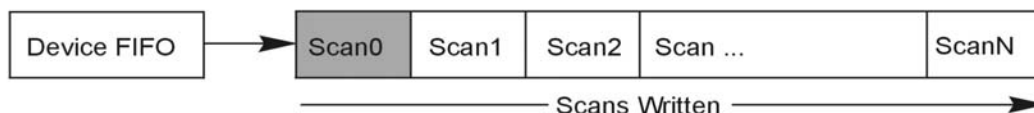
In the linear buffer mode, no more data will be transferred once the end of the buffer has been reached, regardless of whether an acquisition is still active. If using this mode, the application needs to ensure that another buffer is allocated if the acquisition is going to continue beyond the end of the transfer buffer.

DatmUpdateSingle – This value specifies the update mode as a single sample. This setting allows the acquisition transfer buffer to be updated for each sample collected during the acquisition. Compared to the block mode, this setting provides a higher degree of real-time transfer buffer updating at the expense of slower aggregate data throughput rates.

DatmUpdateBlock(default) – This value specifies the update mode as block. This mode allows the acquisition transfer buffer to be updated in blocks of acquired data. The size of the data block depends upon the product in use (see the table below). The block update setting allows faster transfer rates than DatmUpdateSingle mode and therefore should be used when aggregate throughput performance is paramount.

Product	Block Size
DaqBooks	2048 Samples (older versions of these products had a block size of 256)
TempBooks	2048 Samples (older versions of these products had a block size of 256)
DaqBoard (ISA)	2048 Samples (older versions of these products had a block size of 256)
Daq PC Cards	256 Samples
WaveBooks	2048 Samples
DaqBoard/500 Series DaqBoard/1000 Series DaqBook/2000 Series DaqLab/2000 Series DaqScan/2000 Series DaqBoard/2000 Series cPCI DaqBoard/2000c Series	Variable

DatmDriverBuf – This parameter value specifies that the driver allocate and manage the acquisition transfer buffer as a circular buffer whose length is determined by the scanCount parameter (in the current scan group configuration). In this case, the driver will allocate the buffer as a circular buffer so that the application need not set the DatmCycleOff option. Also, since the driver is allocating the buffer, the buf parameter will be ignored with this setting. This option allows the driver to allocate and manage the circular acquisition transfer buffer rather than placing the burden of buffer management on the application. This mode requires the use of the daqAdcTransferBufData function to access and retrieve the acquired data from the driver-allocated acquisition transfer buffer. The scanCount parameter should set to a large enough value to prevent the driver-allocated transfer buffer from over-running. The appropriate setting for scanCount parameter is highly dependent upon the nature of the application and the frequency at which the data will be retrieved (using daqAdcTransferBufData) from the driver allocated transfer buffer. If the interval between data retrievals is large, the scanCount parameter may need to be increased accordingly. If a buffer overrun condition occurs, the error will be posted and returned by the next invocation of the daqAdcTransferBufData function.

DatmUserBuf(default) - The DatmUserBuf option specifies a user-allocated and managed acquisition transfer buffer. Here, the driver will write acquired data to the user-allocated buffer, but the management of the buffer must be performed by the application. If using this mode, there are a few things to keep in mind:

The specified buf parameter must point to memory which has been allocated by the application prior to calling this function.
The allocated buffer must be large enough to hold the number of acquisition scans as determined by the current scan group configuration (as described earlier).
The total amount of data (in scans) written to the user-allocated transfer buffer during the acquisition is determined by the retCount parameter (which is returned by the daqAdcTransferGetStat function). The current data write position within the buffer is:

retCount mod scanCount

where mod is defined as the integer remainder of performing an integer divide of `retCount` by `scanCount`. Detection of a buffer overrun condition is the responsibility of the application.

If using a linear buffer (`DatmCycleOff`) and the acquisition is expected to continue past the end of the allocated buffer size, a new buffer must be allocated and configured using this function in a timely manner to avoid loss of data (as would be the case in an alternating or "ping-pong" buffer approach).

Typical Transfer Buffer Settings

The following table shows a set of are typical transfer buffer settings; it assumes scan group size of 4 channels and sample size of 2 bytes.

Desired Buffer	buf	scanCount	transferMask
Application allocated linear buffer of 50,000 scans updated on every sample	App pointer to memory (size = 50,000 x 4 x 2 = 400,000 bytes)	50,000	DatmCycleOff + DatmUpdateSingle + DatmUserBuf
Application allocated linear buffer of 100,000 scans updated by block	App pointer to memory (size = 100,000 x 4 x 2 = 800,000 bytes)	100,000	DatmCycleOff + DatmUpdateBlock + DatmUserBuf
Application allocated circular buffer of 40,000 scans updated on every sample	App pointer to memory (size = 40000 x 4 x 2 = 320,000 bytes)	40,000	DatmCycleOn + DatmUpdateSingle + DatmUserBuf
Application allocated circular buffer of 200,000 scans updated by block	App pointer to memory (size = 200,000 x 4 x 2 = 1,600,000 bytes)	200,000	DatmCycleOn + DatmUpdateBlock + DatmUserBuf
Driver allocated circular buffer of 40,000 scans updated on every sample	NULL (Driver allocates = 40,000 x 4 x 2 = 320,000 bytes)	40,000	DatmUpdateSingle + DatmDriverBuf
Driver allocated circular buffer of 200,000 scans updated by block	NULL (Driver allocates = 200,000 x 4 x 2 = 1,600,000 bytes)	200,000	DatmUpdateBlock + DatmDriverBuf

Prototypes

C/C++

```
daqAdcTransferSetBuffer(DaqHandleT handle, PWORD buf, DWORD scanCount,
DWORD transferMask);
```

Visual BASIC

```
VBdaqAdcTransferSetBuffer&(ByVal handle&, buf%, ByVal scanCount&, ByVal
transferMask&)
```

Program References

```
DAQADCEXO1.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX05.CPP,
DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP, DBK08EX.CPP, DBK09EX.CPP,
DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP, DBK17EX.CPP, DBK18EX.CPP,
DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP, DBK44EX.CPP, DBK45EX.CPP,
DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP, DBK53_54EX.CPP, DAQEX.FRM (VB)
```

daqAdcTransferStart

Format

```
daqAdcTransferStart(handle)
```

Purpose

`daqAdcTransferStart` initiates an ADC acquisition transfer.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which to initiate an ADC transfer

Parameter Values

handle: obtained from the `daqOpen` function

Returns

DerrNoError   No error

Function Usage

The transfer will be performed under the current active acquisition. If no acquisition is currently active, the transfer will not initiate until an acquisition becomes active (via the `daqAdcArm` function). The transfer will be characterized by the current settings for the transfer buffer. The transfer buffer is configured via the `daqAdcTransferSetBuffer` function.--This buffer configuration must be done before calling the `daqAdcTransferStart` function.

Prototypes

C/C++

```
daqAdcTransferStart(DaqHandleT handle);
```

Visual BASIC

```
VBdaqAdcTransferStart&(ByVal handle&)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX04.CPP,
DAQADCEX05.CPP, DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP,
DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP,
DBK17EX.CPP, DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP,
DBK44EX.CPP, DBK45EX.CPP, DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP,
DBK53_54EX.CPP, DAQEX.FRM (VB)
```

daqAdcTransferStop

Format

```
daqAdcTransferStop(handle)
```

Purpose

`daqAdcTransferStop` stops a current ADC buffer transfer, if one is active.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the ADC transfer will be stopped

Parameter Values

`handle:` obtained from the `daqOpen` function

Returns

DerrNoError   No error

Function Usage

The current transfer will be halted and no more data will transfer into the transfer buffer. Though the transfer is stopped, the acquisition will remain active. Transfers can be re-initiated with `daqAdcTransferStart` after the stop, as long as the current acquisition remains active. The acquisition can be halted by calling the `daqAdcDisarm` function.

Prototypes

C/C++

```
daqAdcTransferStop(DaqHandleT handle);
```

Visual BASIC

```
VBdaqAdcTransferStop&(ByVal handle&)
```

Program References

```
None
```

daqAutoZeroCompensate

Format

daqAutoZeroCompensate (zero)

Purpose

daqAutoZeroCompensate will configure the thermocouple linearization functions to automatically perform zero compensation.

Parameter Summary

Parameter	Type	Description
zero	DaqAutoZeroCompT	If set to DazcAutoZero, will enable auto zero compensation in the daqCvtTC... functions

Parameter Values

zero: see table below

Parameter Type Definitions

zero-(DaqAutoZeroCompT)	
Definition	**Description**
DazcNone	Do not include auto-zero compensation in TC conversion.
DazcAutoZero	Apply auto-zero compensation in TC conversion.

Returns

DerrZCInvParam	Invalid parameter value
DerrNoError	No error

Function Usage

> **Only DaqBook/100 Series, DaqBook/200 Series, Daq PC Cards, DaqBoard (ISA), DaqBook/2000 Series, DaqScan/2000 Series,DaqLab/2000 Series, DaqBoard/2000 series, DaqBoard/1000 Series and cPCI DaqBoard/2000c Series devices connected to a DBK19,DBK52,DBK81-84 or DBK90 expansion card can use the auto-zeroing functions.**

Using the daqAutoZeroCompensate function is the easiest way to use zero compensation with the DBK19 or DBK 52. When enabled, the thermocouple conversion functions will require a CJC zero reading and a TC zero reading to precede the actual CJC and TC reading. This can easily be done by configuring the scan group to read:

- channel 17 using the DBK19/DBK52/DBK81-84/DBK90 CJC gain code (CJC zero)

- channel 17 using the gain code of the connected TC (TC zero)

- channel 16 using the DBK19/DBK52/DBK81-84/DBK90 CJC gain code (CJC)

- the thermocouple channels (channels 18 and above) using the gain code of the connected thermocouples.

> **The real CJC value should be specified (not the offset of the CJC zero) when calling the thermocouple linearization setup functions.**

Prototypes

C/C++

```
daqAutoZeroCompensate(DaqAutoZeroCompT zero);
```

Visual BASIC

```
VBdaqAutoZeroCompensate&(ByVal zero&)
```

Program References

```
DBK19EX.CPP, DBK52EX.CPP
```

This page is intentionally blank.

daqCalClearCalTable

Format

daqCalClearCalTable(handle,tableType,factoryKey)

Purpose

daqCalClearCalTable clears the factory calibration table

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved
tableType	DaqCalTableTypeT	Calibration table type used
factoryKey	DWORD	Key to the factory calibration table; manufacturer use.

Parameter Values

handle: obtained from the daqOpen function

tableType: see tableType table below

factoryKey: reserved by manufacturer

Parameter Type Definitions

tableType - DaqCalTableTypeT	
Definition	Description
DcttFactory	Selects the factory calibration table. The factory calibration table reflects factory calibration constants for the selected device. This is the default setting.
DcttUser	Selects the user-calibration table. The user-calibration table reflects calibration constants defined by the user or the device's user-calibration application. Refer to the device's calibration documentation for specific settings.
DcttSelfCal	Self-calibration constants.

Function Usage

Typically reserved for factory use.

Prototypes

C/C++

daqCalClearCalTable(DaqHandleT handle, DaqCalTableTypeT tableType, DWORD factoryKey);

Visual BASIC

VBdaqCalClearCalTable&(ByVal handle&, ByVal tableType&, ByVal factoryKey&)

Returns

DerrNoError   No error

daqCalConvert

Format

```
daqCalConvert(handle, counts, scans)
```

Purpose

daqCalConvert performs the calibration of one or more scans according to the previously called daqCalSetup function.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to be calibrated
counts	PWORD	Raw data from one or more scans to be calibrated
scans	DWORD	Number of scans of raw data in the counts array

Parameter Values

handle: obtained from the daqOpen function

counts: a valid pointer to an array of uncalibrated scan data whose size must be at least equal to:
   [(scans)* scan size * (the sample size, normally 2 bytes)]

scans: valid values range from 1 to 4,294,967,295; however, memory limitations may apply

Returns

DerrZCInvParam	Invalid parameter value
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The daqCalConvert function will modify the array of data passed to it. The counts parameter specifies a pointer to an array of the raw A/D counts retrieved during an acquisition. Upon return, the counts array will hold calibrated data. The scans parameter indicates the number of scans (as defined by the current scan group configuration) in the acquisition.

This function should be preceded by the daqCalSetup function.

Prototypes

C/C++

```
daqCalConvert(DaqHandleT handle, PWORD counts, DWORD scans);
```

Visual BASIC

```
VBdaqCalConvert&(ByVal handle&, counts%, ByVal scans&)
```

Program References

None

---

daqCalGetCalEquation

Format

    daqCalGetCalEquation(handle, chanConfig, tableType, calEquation)

Purpose

daqCalGetCalEquation retrieves the calibration equation from the currently selected calibration table chosen by the daqCalSelectCalTable function.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved
chanConfig	DaqCalChanConfigT	Configuration info which includes channel and gain.
tableType	DaqCalTableTypeT	Calibration table type to use
calEquation	DaqCalEquationType	Defines the equation type to use

Parameter Values

handle: obtained from the daqOpen function

chanConfig: channel configuration information (see chanConfig table, below)

tableType: see tableType table below

calEquation: see DaqCalEquationType below

Parameter Type Definitions

chanConfig - DaqCalChanConfigT	
Definition	**Description**
DaqCalType	dctAdc = 0; dctDac = 1; dctTrigger = 2; dctVref = 3; dctTime = 4 (for time stamp)
channel	Channel number to which the equation will be applied
gain	Channel gain
flags	Channel flags, as applicable
extendedFlags	The Channel's extended flags, as applicable.

tableType - DaqCalTableTypeT	
Definition	**Description**
DcttFactory	Selects the factory calibration table. The factory calibration table reflects factory calibration constants for the selected device. This is the default setting.
DcttUser	Selects the user-calibration table. The user-calibration table reflects calibration constants defined by the user or the device's user-calibration application. Refer to the device's calibration documentation for specific settings.
DcttSelfCal	Self-calibration constants.

calEquation - DaqCalEquationType	
Definition	**Description**
dcetConstant	= 0; used to select constant
dcetLinear	= 1; used to select linear slope and offset
dcetSplitLinear	= 2; slope positive, slope negative; offset
dcetNonZeroSplitLinear	= 3; slope positive, slope negative, offset positive, offset negative; split
dcetTime	= 4; time stamp: year, month, day, hour, minute, second, milli-second

Function Usage

To retrieve an existing calibration equation.

Prototypes

C/C++

```
daqCalGetCalEquation(DaqHandleT handle,DaqCalChanConfigT pChanConfig,
DaqCalTableTypeT tableType, DaqCalEquationT CalEquation);
```

Visual BASIC

```
VBdaqCalGetCalEquation& (ByVal handle&, chanConfig As DaqCalChanConfigT,
ByVal tableType&, calEquation As DaqCalEquationT)
```

Returns

DerrNoError   No error

daqCalGetConstants

Format

```
daqCalGetConstants(handle, channel, gain, range, gainConstant, offsetConstant)
```

Purpose

`daqCalGetConstants` retrieves the calibration constants from the currently selected calibration table chosen by the `daqCalSelectCalTable` function.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved
channel	DWORD	Channel number to which the calibration settings will be applied
gain	DaqAdcGain	Gain range over which the calibration settings will be applied
range	DaqAdcRangeT	A/D input range over which the calibration settings will be applied
gainConstant	PWORD	Pointer to the gain value for the current entry
offsetConstant	PSHORT	Pointer to the offset value for the current entry

Parameter Values

handle: obtained from the `daqOpen` function

channel: a valid channel for the device

gain: see ADC Gain Definition table for `gain` parameter definitions

range: see table below

gainConstant: valid pointer to a word variable (2 bytes) in which the gain constant will be stored upon return from this function (valid gain constant range is from 0 to 65,535)

offsetConstant: valid pointer to a short integer variable (2 bytes) in which the offset constant will be stored upon return from this function (valid offset constant range is from –32,768 to 32,767)

Parameter Type Definitions

adcRanges-(DaqAdcRangeT)	
Definition	**Description**
DarUni0to10V	Sets acquisition range as unipolar from 0 to +10 Volt range
DarBiMinus5to5V	Sets acquisition range as bipolar from -5 to +5 Volt range
DarUniPolarDE	Sets acquisition range as unipolar differential
DarBiPolarDE	Sets acquisition range as bipolar differential
DarUniPolarSE	Sets acquisition range as unipolar single-ended
DarBiPolarSE	Sets acquisition range as bipolar single-ended

Returns

DerrNoError   No error

Function Usage

Selecting the Calibration Constants to Retrieve

Before calling this function, the `daqOpen` function should be used to open the device from which the calibration constants will be retrieved, and the `daqCalSelectCalTable` function should be used to select the calibration table from which the calibration constants will be retrieved. The internal tables are organized into gain and calibration entries according to the channel, gain range and A/D range capabilities of the device. The following need to be selected for each calibration gain and offset constant entries to be retrieved:

- The `channel` parameter should be set to the desired channel.
- The `gain` parameter should be set to the desired gain setting for the selected channel.
- The `range` parameter should be set to the desired range for the gain and channel selection.

Retrieving the Calibration Constants

The `gains` and `offsets` constants for the `handle`, `channel`, `gain` and `range` entries selected are set for the device and stored in the memory pointed to by the corresponding parameters `gainConstant` and `offsetConstant` parameters respectively. The values returned are stored in a 16-bit data word which applies to the current settings for the selected table entry. These values need to be calculated and set by the calibration application according to the methods described below.

Interpreting Calibration Constant Values

These calibration constants are gains and offsets that are applied to the input data. After the data comes in, it is multiplied by the gain, and then the offset is added to it. The resulting data has been converted from raw A/D data to calibrated data. Each channel, gain, and bipolar/unipolar range setting has a different pair of gain and offset values.

As mentioned above, the first three parameters of the `daqCalSetConstants` function specify which set of constants are to be changed. The last two parameters are the actual constants. These constants are in a particular binary format. The gain constant is 32,768 times the gain. For a gain of x1, the gain constant is 32,768 or 0x8000. The maximum gain is approximately x2 (65,535 / 32,768), and the minimum gain is x 0 (0 / 32,768). The offset (a left-justified signed 12-bit number) is added to the final result. A single least-significant bit has an integer value of 16 or 0x0010. Setting the calibration constants affects subsequent acquisitions until another `daqOpen` is performed. After `daqOpen`, the original calibration constants are re-read from the NVRAM in the WaveBook and expansion chassis; then, the working copy as set by `daqCalSetConstants` is overwritten.

Prototypes

C/C++

```
daqCalGetConstants(DaqHandleT handle, DWORD channel, DaqAdcGain gain,
DaqAdcRangeT range, PWORD gainConstant, PSHORT offsetConstant);
```

Visual BASIC

```
VBdaqCalGetConstants(ByVal handle&, ByVal channel, ByVal gain, ByVal
range, ByRef gainConstant, ByRef offsetConstant);
```

Program References

```
None
```

daqCalPerformSelfCal

Format

```
daqCalPerformSelfCal(handle,chanConfig,calFlags)
```

Purpose

daqCalPerformSelfCal selects a channel for self-calibration.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved
chanConfig	DaqCalChanConfigT	Configuration info which includes channel and gain.
calFlags	DaqCalSelfCalFlags	Calibrates all available options.

Parameter Values

handle: obtained from the daqOpen function

chanConfig: channel configuration information (see chanConfig table, below)

calFlags: DcsfCalibrateAll = 0x01, self-calibration of all available options

Parameter Type Definitions

chanConfig - DaqCalChanConfigT	
Definition	Description
DaqCalType	dctAdc = 0; dctDac = 1; dctTrigger = 2; dctVref = 3; dctTime = 4 (for time stamp)
channel	Channel number to which the equation will be applied
gain	Channel gain
flags	Channel flags, as applicable
extendedFlags	The Channel's extended flags, as applicable.

Function Usage

Used to perform a self-calibration of all available options via the DcsfCalibrateAll flag.

Prototypes

C/C++

```
daqCalPerformSelfCal(DaqHandleT handle, DaqCalChanConfigT chanConfig,
DaqCalSelfCalFlags calFlags)
```

Visual BASIC

```
VBdaqCalPerformSelfCal(ByVal handle&, ByRef chanConfig As
DaqCalChanConfigT, ByVal calFlags As DaqCalSelfCalFlags)
```

Returns

DerrNoError   No error

daqCalSaveCalTable

Format

daqCalSaveCalTable(handle,tableType,factoryKey)

Purpose

daqCalSaveCalTable saves the calibration table

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved
tableType	DaqCalTableTypeT	Calibration table type
factoryKey	DWORD	Key to the factory calibration table. Manufacturer's use.

Parameter Values

handle: obtained from the daqOpen function

tableType: see tableType table below

factoryKey: reserved for manufacturer

Parameter Type Definitions

tableType - DaqCalTableTypeT	
Definition	Description
DcttFactory	Selects the factory calibration table. The factory calibration table reflects factory calibration constants for the selected device. This is the default setting.
DcttUser	Selects the user-calibration table. The user-calibration table reflects calibration constants defined by the user or the device's user-calibration application. Refer to the device's calibration documentation for specific settings.
DcttSelfCal	Self-calibration constants.

Function Usage

Used to save the calibration table.

Prototypes

C/C++

daqCalSaveCalTable(DaqHandleT handle, DaqCalTableTypeT tableType, DWORD factoryKey);

Visual BASIC

VBdaqCalSaveCalTable&(ByVal handle&, ByVal tableType&, ByVal factoryKey&)

Returns

DerrNoError   No error

daqCalSaveConstants

Format

```
daqCalSaveConstants(handle, channel)
```

Purpose

daqCalSaveConstants saves the current calibration table selected by the daqCalSelectCalTable function.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the calibration constants will be saved
channel	DWORD	Channel whose current calibration settings will be saved

Parameter Values

handle: obtained from the daqOpen function

channel: a valid channel for the device

Returns

DerrNoError   No error

Function Usage

Saving the Current Calibration Table

Current calibration constants can be updated or modified with the daqCalSetConstants function. The working calibration table should only be saved after all desired calibration constants have been updated for the device.

daqCalGetConstants retrieves the calibration constants from the currently selected calibration table chosen by the daqCalSelectCalTable function for the device.

Before calling this function, the daqOpen function should be used to open the device whose calibration constants are being saved, and the daqCalSelectCalTable function should be used to select the calibration table from which the calibration constants are being saved. The internal tables are organized into gain and calibration entries according to the channel, gain range and A/D range capabilities of the device.

Prototypes

C/C++

```
daqCalSaveConstants(DaqHandleT handle, DWORD channel);
```

Visual BASIC

```
VBdaqCalSaveConstants(ByVal handle&, ByVal channel)
```

Program References

None

daqCalSelectCalTable

Format

daqCalSelectCalTable(handle, tableType)

Purpose

daqCalSelectCalTable selects the calibration table source for the device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which ADC transfer status is to be retrieved
tableType	DaqCalTableTypeT	Calibration table type to use

Parameter Values

handle: obtained from the daqOpen function
tableType: see table below

Parameter Type Definitions

tableType - DaqCalTableTypeT	
Definition	Description
DcttFactory	Selects the factory calibration table. The factory calibration table reflects factory calibration constants for the selected device. This is the default setting.
DcttUser	Selects the user-calibration table. The user-calibration table reflects calibration constants defined by the user or the device's user-calibration application. Refer to the device's calibration documentation for specific settings.
DcttSelfCal	Self-calibration constants.

Returns

DerrNoError   No error

Function Usage

Selecting the Calibration Table

The daqCalSelectCalTable function should be used to set the current calibration table for the device. Currently, there are two valid calibration table types which can be selected with the tableType parameter-- either the factory calibration table can be selected with the DcttFactory value, or a user-defined calibration table can be selected with the DcttUser value. The current calibration table at any time will be set to the calibration table last selected during the current device session.

Prototypes

C/C++

daqCalSelectCalTable(DaqHandleT handle, DaqCalTableTypeT tableType);

Visual BASIC

VBdaqCalSelectCalTable(ByVal handle&, ByVal tableType)

Program References

None

daqCalSelectInputSignal

Format

```
daqCalSelectInputSignal(handle, input)
```

Purpose

`daqCalSelectInputSignal` selects of the input signal source for user calibration.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which the ADC transfer status will be retrieved
input	DaqCalInputT	Signal source to use for calibration input

Parameter Values

handle: obtained from the `daqOpen` function
input: see table below

Parameter Type Definitions

input–(`DaqCalInputT`)	
Definition	Description
DciNormal	External signal from device input connector(s)
DciCalGround	Internal calibration ground signal
DciCal5V	Internal 5 V calibration signal
DciCal500mV	Internal 500 mV calibration signal

Returns

DerrNoError   No error

Function Usage

Selecting the Calibration Input Signal Source

The input signal source is specified by the `input` parameter. The input signal selection allows the calibration application to select the signal source from which the channels can be calibrated for a given calibration session. Prior to calling this function, the `daqOpen` function should be used to open the device to be calibrated, and the `daqCalSelectCalTable` function should specify which calibration table will be used to perform the calibration .

Prototypes

C/C++

```
daqCalSelectInputSignal(DaqHandleT handle, DaqCalInputT input);
```

Visual BASIC

```
VBdaqCalSelectInputSignal(ByVal handle&, inputSignal&)
```

Program References

None

---

daqCalSetCalEquation

Format

daqCalSetCalEquation(handle,chanConfig,tableType, calEquation)

Purpose

daqCalSetCalEquation defines and sets the calibration equation

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved.
chanConfig	DaqCalChanConfigT	Configuration info which includes channel and gain.
tableType	DaqCalTableTypeT	Calibration table type to use.
pCalEquation	DaqCalEquationType	Defines the equation type to use

Parameter Values

handle: obtained from the daqOpen function

chanConfig: channel configuration information (see chanConfig table, below)

tableType: see tableType table below

calEquation: see DaqCalEquationType below

Parameter Type Definitions

chanConfig - DaqCalChanConfigT	
Definition	Description
DaqCalType	dctAdc = 0; dctDac = 1; dctTrigger = 2; dctVref = 3; dctTime = 4 (for time stamp)
channel	Channel number to which the equation will be applied
gain	Channel gain
flags	Channel flags, as applicable
extendedFlags	The Channel's extended flags, as applicable.

tableType - DaqCalTableTypeT	
Definition	Description
DcttFactory	Selects the factory calibration table. The factory calibration table reflects factory calibration constants for the selected device. This is the default setting.
DcttUser	Selects the user-calibration table. The user-calibration table reflects calibration constants defined by the user or the device's user-calibration application. Refer to the device's calibration documentation for specific settings.
DcttSelfCal	Self-calibration constants.

calEquation - DaqCalEquationType	
Definition	Description
dcetConstant	= 0; used to select constant
dcetLinear	= 1; used to select linear slope and offset
dcetSplitLinear	= 2; slope positive, slope negative; offset
dcetNonZeroSplitLinear	= 3; slope positive, slope negative, offset positive, offset negative; split
dcetTime	= 4; time stamp: year, month, day, hour, minute, second, milli-second

Function Usage

Used to set calibration constants.

Prototypes

C/C++

```
daqCalSetCalEquation(DaqHandleT handle,DaqCalChanConfigT chanConfig,
DaqCalTableTypeT tableType,DaqCalEquationT calEquation);
```

Visual BASIC

```
VBdaqCalSetCalEquation& (ByVal handle&, chanConfig As DaqCalChanConfigT,
ByVal tableType&, calEquation As DaqCalEquationT)
```

Returns

DerrNoError   No error

daqCalSetCalPoints

Format

```
daqCalSetCalPoints(handle,chanConfig,calData,tableType)
```

Purpose

`daqCalSetCalPoints` sets calibration constants automaticllay after compare true (expected) values to observed values.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved.
chanConfig	DaqCalChanConfigT	Configuration info which includes channel and gain.
calData	DaqCalUserCalDataT	Calibration points (0 to 63)
tableType	DaqCalTableTypeT	Calibration table type to use.

Parameter Values

handle: obtained from the `daqOpen` function

chanConfig: channel configuration information (see `chanConfig` table, below)

calData: see `calDataType` table

tableType: see tableType table below

Parameter Type Definitions

chanConfig - DaqCalChanConfigT	
Definition	**Description**
DaqCalType	dctAdc = 0; dctDac = 1; dctTrigger = 2; dctVref = 3; dctTime = 4 (for time stamp)
channel	Channel number to which the equation will be applied
gain	Channel gain
flags	Channel flags, as applicable
extendedFlags	The Channel's extended flags, as applicable.

calDataType	
Definition	**Description**
dcdtConstant	= 0 , global constant
dcdtLinear	= 1, global linear constant
dcdtSplitLinear	= 2, global split linear constant
DaqCalUserCalDataT	
calDataType	As Long
numCalPoints	As Long
calPoints	0 to 63 as DaqCalPointT

tableType - DaqCalTableTypeT	
Definition	**Description**
DcttFactory	Selects the factory calibration table. The factory calibration table reflects factory calibration constants for the selected device. This is the default setting.
DcttUser	Selects the user-calibration table. The user-calibration table reflects calibration constants defined by the user or the device's user-calibration application. Refer to the device's calibration documentation for specific settings.
DcttSelfCal	Self-calibration constants.

Function Usage

Calculates and stores constants to calibraration table after comparing true (expected) vlalues to the corresponding measured values.

Prototypes

C/C++

```
daqCalSetCalPoints(DaqHandleT handle, DaqCalChanConfigT chanConfig,
DaqCalUserCalDataT calData, DaqCalTableTypeT tableType);
```

Visual BASIC

```
VBdaqCalSetCalPoints&((ByVal handle&, chanConfig As DaqCalChanConfigT,
calData As DaqCalUserCalDataT, ByVal tableType&)
```

Returns

DerrNoError   No error

daqCalSetConstants

Format

```
daqCalSetConstants(handle, channel, gain, range, gainConstant,
offsetConstant)
```

Purpose

daqCalSetConstants sets the user-accessible calibration constants.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which ADC transfer status will be retrieved
channel	DWORD	Channel number for which to apply the calibration settings
gain	DaqAdcGain	Gain range for which to apply the calibration settings
range	DaqAdcRangeT	A/D input range for which to apply the calibration settings
gainConstant	WORD	Gain value to apply
offsetConstant	SHORT	Offset value to apply

Parameter Values

handle: obtained from the daqOpen function

channel: a valid channel for the device

gain: see ADC Gain Definition table for gain parameter definitions

range: see table below

gainConstant: valid values range from 0 to 65,535

offsetConstant: valid values range from –32,768 to 32,767

Parameter Type Definitions

range-(DaqAdcRangeT)	
Definition	**Description**
DarUni0to10V	Sets acquisition range as unipolar from 0 to +10 Volt range
DarBiMinus5to5V	Sets acquisition range as bipolar from -5 to +5 Volt range
DarUniPolarDE	Sets acquisition range as unipolar differential
DarBiPolarDE	Sets acquisition range as bipolar differential
DarUniPolarSE	Sets acquisition range as unipolar single-ended
DarBiPolarSE	Sets acquisition range as bipolar single-ended

Returns

DerrNoError   No error

Function Usage

Selecting the Calibration Constants to Retrieve

Before calling this function, the daqOpen function should be used to open the device from which the calibration constants will be retrieved, and the daqCalSelectCalTable functions should be used to select the calibration table from which the calibration constants will be retrieved. The internal tables are organized into gain and calibration entries according to the channel, gain range and A/D range capabilities of the device. The following need to be selected for each calibration gain and offset constant entries to be retrieved:

- The channel parameter should be set to the desired channel.
- The gain parameter should be set to the desired gain setting for the selected channel.
- The range parameter should be set to the desired range for the gain and channel selection.

Retrieving the Calibration Constants

The `gains` and `offsets` constants for the `handle`, `channel`, `gain` and `range` entries selected are set for the device and stored in the memory pointed to by the corresponding parameters `gainConstant` and `offsetConstant` parameters respectively. The values returned are stored in a 16-bit data word which applies to the current settings for the selected table entry. These values need to be calculated and set by the calibration application according to the methods described below.

Interpreting Calibration Constant Values

These calibration constants are gains and offsets that are applied to the input data. After the data comes in, it is multiplied by the gain, and then the offset is added to it. The resulting data has been converted from raw A/D data to calibrated analog data. Each channel, gain, and bipolar/unipolar range setting has a different pair of gain and offset values.

As mentioned above, the first three parameters of the `daqCalSetConstants` function specify which set of constants are to be changed. The last two parameters are the actual constants. These constants are in a particular binary format. The gain constant is 32,768 times the gain. For a gain of x1, the gain constant is 32,768 or 0x8000. The maximum gain is approximately x2 (65,535 / 32,768), and the minimum gain is x 0 (0 / 32,768). The offset (a left-justified signed 12-bit number) is added to the final result. A single least-significant bit has an integer value of 16 or 0x0010. Setting the calibration constants affects subsequent acquisitions until another `daqOpen` is performed. After `daqOpen`, the original calibration constants are re-read from the NVRAM in the WaveBook and expansion chassis; then, the working copy as set by `daqCalSetConstants` is overwritten.

Prototypes

C/C++

```
daqCalSetConstants(DaqHandleT handle, DWORD channel, DaqAdcGain gain,
DaqAdcRangeT range, WORD gainConstant, SHORT offsetConstant);
```

Visual BASIC

```
VBdaqCalSetConstants(ByVal handle&, ByVal channel, ByVal gain, ByVal
range, ByVal gainConstant, ByVal offsetConstant);
```

Program References

```
None
```

daqCalSetup

Also See: **daqReadCalFile, daqCalConvert, daqCalSetupConvert**

Format

```
daqCalSetup(handle, nscan, readingsPos, nReadings, chanType, chanGain,
startChan, bipolar, noOffset)
```

Purpose

daqCalSetup configures the order and type of data to be calibrated.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to be calibrated
nscan	DWORD	Number of channels in a single scan
readingsPos	DWORD	Position of the readings to be calibrated within the scan
nReadings	DWORD	Number of readings to calibrate
chanType	DcalType	Type of channel/board from which the readings to be calibrated are read
chanGain	DaqAdcGain	Gain setting of the channels to be calibrated
startChan	DWORD	Channel number of the first channel to be converted
bipolar	BOOL	If true, the DaqBook/DaqBoard is configured for bipolar readings; if false, it is configured for unipolar readings
noOffset	BOOL	If true, the offset calibration constant will not be used to calibrate the readings; if false, the offset calibration will be used

Parameter Values

handle: obtained from the daqOpen function

nscan: the number of channels in the scan group configuration (see daqAdcSetScan)

readingsPos: the position of the channels to calibrate within the scan (see daqAdcSetScan)

nReadings: the number of readings to calibrate from 1 to 4,294,967,295; however, memory limitations may apply

chanType: see table below

chanGain: see ADC Gain Definition table for chanGain parameter definitions

startChan: must be a valid device channel within the scan group definition (see daqAdcSetScan)

bipolar: valid values are either true ($\neq 0$) or false ($= 0$)

noOffset: valid values are either true ($\neq 0$) or false ($= 0$)

Parameter Type Definitions

chanType-(DcalType)		
Definitions	Channel Configuration	Description
DcalTypeDefault	Any voltage channel	Defines a calibration of main unit or expansion channel voltage
DcalTypeCJC	Cold Junction Compensation Channel	Defines a calibration of expansion CJC channel for a thermocouple module or card
DcalTypeBypass	Filter Bypass mode for DBK4	Defines a calibration for DBK4 channel in filter bypass mode
DcalTypeFilter	Filter Cutoff mode for DBK4	Defines a calibration for DBK4 channel in filter cutoff mode

The value for the chantype parameter should be set to DcalTypeCJC (1) when calibrating a CJC channel of a thermocouple module or card, and DcalTypeDefault (0) when reading any other channel.

Returns

`DerrZCInvParam`	Invalid parameter value
`DerrNoError`	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The `daqCalSetup` function configures the calibration of acquisition data. It requires that all data to be calibrated comes from consecutive channels, each configured for the same gain, polarity, and channel type. The calibration itself can be configured to use only the gain calibration constant and not the offset constant--this allows the offset to be removed at runtime using the zero compensation functions. The following list describes how `daqCalSetup's` parameters configure a data calibration in more detail:

The `nscan` parameter indicates the number of channels in the scan. (See `daqAdcSetScan` function for complete details on channel scan group configuration settings).

The `readingPos` parameter indicates the *position* within the channel scan to begin calibrating--this is different that the `startChan` parameter, which indicates the beginning *channel* within the channel scan group (see the `daqAdcSetScan` function for complete details on channel scan group configuration settings).

The `nReadings` parameter indicates the number of readings to calibrate in the channel scan, starting from the `readingPos` position.

The `chanType` parameter is a pointer to an array of length `nscan`; the value of each element in the array is the channel type for the corresponding channel in the scan configuration.

The `chanGain` parameter is a pointer to an array of length equal to `nscan`; the value of each element in the array is the channel gain for the corresponding channel in the scan configuration.

The `startChan` parameter indicates the *channel number* within the channel scan to begin calibrating—this is different that the `readingPos` parameter, which indicates the beginning *position* within the channel scan group (see the `daqAdcSetScan` function for complete details on channel scan group configuration settings).

The `bipolar` parameter should be `true` if the device whose data is being calibrated was set to bipolar mode during the acquisition. If the device was set to unipolar mode when the data being calibrated was acquired, this value should be `false`.

The `noOffset` parameter should be `true` if it is desirable to perform a gain constant calibration only. If this value is `true`, the offset calibration constant will not be used and the calibration will not attempt to adjust for offset errors. If this value is `false`, then the offset constant calibration will be performed and the calibration process will attempt to adjust the data to reduce offset error.

The `daqCalSetup` function only configures a data calibration—the actual calibration is performed by the `daqCalConvert` function. For convenience, `daqCalSetupConvert` can perform both setup and conversion.

Prototypes

C/C++

```
daqCalSetup(DaqHandleT handle,DWORD nscan, DWORD readingsPos, DWORD
nReadings, DcalType chanType, DaqAdcGain chanGain, DWORD startChan, BOOL
bipolar, BOOL noOffset);
```

Visual BASIC

```
VbdaqCalSetup&(ByVal handle&, ByVal nscan&, ByVal readingsPos&, ByVal
nReadings&, ByVal chanType&, ByVal chanGain&, ByVal startChan&, ByVal
bipolar&, ByVal noOffset&)
```

Program References

```
None
```

daqCalSetupConvert

Format

```
daqCalSetupConvert(handle, nscan, readingsPos, nReadings, chanType,
chanGain, startChan, bipolar, noOffset, counts, scans)
```

Purpose

`daqCalSetupConvert` both configures and performs the calibration of the specified data.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to be calibrated
nscan	DWORD	Number of channels in a single scan
readingsPos	DWORD	Position of the readings to be calibrated within the scan
nReadings	DWORD	Number of readings to calibrate
chanType	DcalType	Type of channel/board from which the readings to be calibrated are read
chanGain	DaqAdcGain	Gain setting of the channels to be calibrated
startChan	DWORD	Channel number of the first channel to be converted
bipolar	BOOL	If true, the DaqBook/DaqBoard is configured for bipolar readings; if false, it is configured for unipolar readings
noOffset	BOOL	If true, the offset calibration constant will not be used to calibrate the readings; if false, the offset calibration will be used
counts	PWORD	Raw data from one or more scans
scans	DWORD	Number of scans of raw data in the counts array

Parameter Values

handle: obtained from the daqOpen function

nscan: the number of channels in the scan group configuration (see daqAdcSetScan)

readingsPos: the position of the channels to calibrate within the scan (see daqAdcSetScan)

nReadings: the number of readings to calibrate from 1 to 4,294,967,295; however, memory limitations
       may apply

chanType: see table below

chanGain: see ADC Gain Definition table for chanGain parameter definitions

startChan: must be a valid device channel within the scan group definition (see daqAdcSetScan)

bipolar: valid values are either true ($\neq 0$) or false ($= 0$)

noOffset: valid values are either true ($\neq 0$) or false ($= 0$)

counts: a valid pointer to an array of uncalibrated scan data whose size must be at least equal to
       (scans * scan size * 2)

scans: the number of scans to be calibrated from 1 to 4,294,967,295; however, memory limitations may
       apply

Parameter Type Definitions

chanType - DcalType		
Definitions	Channel Configuration	Description
DcalTypeDefault	Any voltage channel	Defines a calibration of main unit or expansion channel voltage
DcalTypeCJC	Cold Junction Compensation Channel	Defines a calibration of expansion CJC channel for a thermocouple module or card.
DcalTypeBypass	Filter Bypass mode for DBK4	Defines a calibration for DBK4 channel in filter bypass mode
DcalTypeFilter	Filter Cutoff mode for DBK4	Defines a calibration for DBK4 channel in filter cutoff mode

The value for the `chantype` parameter should be set to `DcalTypeCJC` (1) when calibrating a CJC channel of a thermocouple module or card, and `DcalTypeDefault` (0) when reading any other channel.

Returns

DerrZCInvParam   Invalid parameter value
DerrNoError   No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The `daqCalSetupConvert` function will setup the calibration in the same manner as the `daqCalSetup` function and will perform the calibration in the same manner as the `daqCalConvert` function:

Like the `daqCalSetup` function, this function requires that all data to be calibrated comes from consecutive channels, each configured for the same gain, polarity, and channel type. The calibration can be configured to use only the gain calibration constant and not the offset constant--this allows the offset to be removed at runtime using the zero compensation functions.

Refer to the `daqCalSetup` function for a description of the `counts, nscan, readingPos, nreadings, chanType, chanGain, startChan, bipolar` and `noOffset` parameters.

Like the `daqCalConvert` function, this function will modify the array of data passed to it. The `counts` parameter specifies a pointer to an array of the raw A/D counts retrieved during an acquisition. Upon return, the `counts` array will hold calibrated data. The `scans` parameter indicates the number of scans (as defined by the current scan group configuration) in the acquisition.

Prototypes

C/C++

```
daqCalSetupConvert(DaqHandleT handle,DWORD nscan, DWORD readingsPos, DWORD
nReadings, DcalType chanType, DaqAdcGain chanGain, DWORD startChan, BOOL
bipolar, BOOL noOffset, PWORD counts, DWORD scans);
```

Visual BASIC

```
VBdaqCalSetupConvert&(ByVal handle&, ByVal nscan&, ByVal readingsPos&,
ByVal nReadings&, ByVal chanType&, ByVal chanGain&, ByVal startChan&,
ByVal bipolar&, ByVal noOffset&, counts%, ByVal scans&)
```

Program References

DBK81EX.CPP, DBK82_83EX.CPP, DBK90EX.CPP

daqClose

Format

```
daqClose(handle)
```

Purpose

`daqClose` is used to close a device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to be closed

Parameter Values

`handle:` obtained from the `daqOpen` function

Returns

`DerrNoError`   No error

Function Usage

Once the specified device has been closed, no subsequent communication with the device can be performed. In order to re-establish communications with a closed device, the device must be re-opened with the `daqOpen` function.

Prototypes

C/C++

```
daqClose(DaqHandleT handle);
```

Visual BASIC

```
VBdaqClose&(ByVal handle&)
```

Program References

```
DAQADCEX01.CPP, DAQDIGIOEX01.CPP, DAQDIGIOEX02.CPP, DAQADCEX02.CPP,
DAQADCEX03.CPP, DAQADCEX04.CPP, DAQADCEX05.CPP, DAQADCEX06.CPP,
DAQADCEX07.CPP, DAQDACEX01.CPP, DAQ9513_01.C, DBK02EX.CPP, DBK04EX.CPP,
DBK05EX.CPP, DBK07EX.CPP, DBK08EX.CPP, DBK09EX.CPP, DBK12_13EX.CPP,
DBK15EX.CPP, DBK16EX.CPP, DBK17EX.CPP, DBK18EX.CPP,
DBK81EX.CPP,DBK82_83EX.CPP, DBK90EX.CPP, DBK20_21EX.CPP, DBK23_24EX.CPP,
DBK25EX.CPP, DBK42EX.CPP, DBK43EX.CPP, DBK44EX.CPP, DBK45EX.CPP,
DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP, DBK53_54EX.CPP, DAQDACEX02.CPP,
DAQTMREX01.CPP, DAQDACEX03.CPP, DAQEX.FRM (VB)
```

daqCreateDevice

Format

```
daqCreateDevice(pDevInfo)
```

Purpose

daqCreateDevice creates a device in the Windows Registry using the specified device information which is equivalent to creating the device using the DaqX control panel applet.

Parameter Summary

Parameter	Type	Description
pDevInfo	DaqDevInfoPT	Pointer to the device information structure that defines the device to be created

Parameter Values

pDevInfo: see the DaqDevInfoT table below

Parameter Type Definitions

pDevInfo – (DaqDevInfoT)		
Definition	**Description**	**Format**
AliasName	Device alias name	STRING
DeviceType	Main chassis device type (see the DhiHardwareVersion table of the daqGetInfo function)	DaqHardwareVersion
DeviceSubType	Main chassis device sub type (see the DaqHardwareSubType table of the daqGetInfo function)	DaqHardwareSubType
Reserved1	Reserved for future use	DWORD
Reserved2	Reserved for future use	DWORD
InfoType	Indicates which one of the following structures contain valid interface specific device information (see the DaqInfoType table below)	DaqInfoType
Tcp	TCP interface information (see the DaqInfoTcpT below)	DaqInfoTcpT
TcpChild	TCP child device interface information (see the DaqInfoTcpChildT below)	DaqInfoTcpChildT
Parallel	Parallel port interface information (see the DaqInfoParallelT below)	DaqInfoParallelT
Pci	PCI and Compact PCI bus interface information (see the DaqInfoPciT below)	DaqInfoPciT
PcCard	PC Card interface information (see the DaqInfoPcCardT below)	DaqInfoPcCardT
Generic	ISA bus interface information (see the DaqInfoGenericT below)	DaqInfoGenericT

InfoType – (DaqInfoType)	
Definition	**Description**
DaqInfoTypeUnknown	Indicates that the DaqDevInfoT device information structure has no interface specific device information
DaqInfoTypeGeneric	Indicates that the Generic structure element of the DaqDevInfoT structure contains valid interface information
DaqInfoTypeTcp	Indicates that the Tcp structure element of the DaqDevInfoT structure contains valid interface information
DaqInfoTypeTcpChild	Indicates that the TcpChild structure element of the DaqDevInfoT structure contains valid interface information
DaqInfoTypeParallel	Indicates that the Parallel structure element of the DaqDevInfoT structure contains valid interface information
DaqInfoTypePci	Indicates that the Pci structure element of the DaqDevInfoT structure contains valid interface information
DaqInfoTypePcCard	Indicates that the PcCard structure element of the DaqDevInfoT structure contains valid interface information

Tcp – (DaqInfoTcpT)		
Definition	**Description**	**Format**
IPMode	IP Address connection mode (see DaqIPModeT table below)	DaqIPModeT
SerialNum	Device serial number (not required if IPMode is DaqIPModeManualIP)	STRING
IPAddress	Device IP address (not required if IPMode is DaqIPModeAutoDetect)	STRING

IPMode – (DaqIPModeT)	
Definition	**Description**
DaqIPModeAutoDetect	Indicates that the connection to a TCP device or TCP child device will be made based on the serial number of the device
DaqIPModeManualIP	Indicates that the connection to a TCP device or TCP child device will be made using a manually defined IP address

TcpChild – (DaqInfoTcpChildT)		
Definition	**Description**	**Format**
ParentAliasName	Alias name of the TCP parent device to which this device is connected	STRING
ExpPort	Expansion port number of the TCP parent device to which this device is connected	DWORD
Tcp	TCP interface information of the TCP parent device to which this device is connected (not required by daqCreateDevice, see the DaqInfoTcpT table above)	DaqInfoTcpT

Parallel – (DaqInfoParallelT)		
Definition	**Description**	**Format**
LPTPort	Parallel port number to which this device is connected	DWORD
Protocol	Parallel port protocol (see the following DaqProtocol table)	DaqProtocol

DaqProtocol	
Value Returned	**Protocol**
DaqProtocolNone	Communications not established
DaqProtocol4	Standard LPT Port 4-bit mode
DaqProtocol8	Standard LPT Port 8-bit mode
DaqProtocolSMC666	SMC 37C666 EPP mode
DaqProtocolFastEPP	WBK20/21 Fast EPP mode
DaqProtocolECP	Enhanced Capability Port
DaqProtocol8BitEPP	8-bit EPP mode
DaqProtocolISA	ISA bus card DaqBoards
DaqProtocolPcCard	PCCard for Daq (PCMCIA)
DaqProtocolUSB	USB protocol
DaqProtocolPCI	PCI bus Data Acquisition Boards
DaqProtocolCPCI	Compact PCI (cPCI) Data Acquisition Boards
DaqProtocolTCPIP	Ethernet

Pci – (DaqInfoPciT)		
Definition	**Description**	**Format**
SerialNum	Device serial number	DWORD
Bus	PCI bus number in which the device is installed	DWORD
Slot	PCI slot number in which the device is installed	DWORD

PcCard – (DaqInfoPcCardT)		
Definition	**Description**	**Format**
Socket	PC Card socket number	DWORD
BasePortAddr	Base port address	DWORD
InterruptLevel	Interrupt level	DWORD
DriverInstance	Driver instance	DWORD
DaqModel	Daq model number	DWORD

Generic – (DaqInfoGenericT)		
Definition	**Description**	**Format**
BasePortAddr	Base port address	DWORD
Protocol	Device protocol (see the DaqProtocol section of these tables)	DaqProtocol
InterruptLevel	Interrupt level	DWORD
DMAChannel	DMA channel	DWORD

Returns

DerrNoError if successful or a DaqError error code (see the DaqError table at the end of chapter 4)

Function Usage

This function can be used to programmatically create devices that can be opened using daqOpen. This function can be used instead of or in conjunction with the DaqX control panel applet. It can also be used with daqDeleteDevice to create devices on the fly or with daqGetDeviceInventory to dynamically create newly detected devices.

When creating a device, be sure that the alias name is unique and the device type is correct. The InfoType parameter should be set to indicate which interface information structure is used.

Prototypes

C/C++

daqCreateDevice(DaqDevInfoPT pDevInfo);

Visual BASIC

VBdaqCreateDevice&(DevInfo As DaqDevInfoT)

Program References

None

daqCvtChannelType

Format

    daqCvtChannelType (ChType, ChSubType, pChTypeStr, ChTypeStrSize)

Purpose

daqCvtChannelType converts a channel type and sub type to a string.

Parameter Summary

Parameter	Type	Description
ChType	DaqAdcExpChType	Channel type
ChSubType	DaqAdcExpChSubType	Channel sub type
pChTypeStr	LPSTR	C/C++ only: String format of the channel type
ChTypeStrSize	DWORD	C/C++ only: Size of the channel type string

Parameter Values

ChType: see DaqAdcExpChType of the daqGetInfo function
ChSubType: see DaqAdcExpChSubType of the daqGetInfo function
pChTypeStr: pointer to a string large enough to hold the channel type string
ChTypeStrSize: Size of the channel type string in bytes

Returns

Nothing in C/C++,
The channel type string in Visual BASIC

Function Usage

This function converts a channel type and sub type to a string.

Prototypes

C/C++

daqCvtChannelType(DaqAdcExpChType ChType, DaqAdcExpChSubType ChSubType, PCHAR
pChTypeStr, DWORD ChTypeStrSize);

Visual BASIC

VBdaqCvtChannelType$(ByVal ChType&, ByVal ChSubType&)

Program References

None

daqCvtHardwareType

Format

```
daqCvtHardwareType (HwType, HwSubType, pHwTypeStr, HwTypeStrSize)
```

Purpose

daqCvtHardwareType converts a hardware type and sub type to a string.

Parameter Summary

Parameter	Type	Description
HwType	DaqHardwareVersion	Hardware type
HwSubType	DaqHardwareSubType	Hardware sub type
pHwTypeStr	LPSTR	C/C++ only: String format of the hardware type
HwTypeStrSize	DWORD	C/C++ only: Size of the hardware type string

Parameter Values

HwType: see DaqHardwareVersion of the daqGetHardwareInfo function
HwSubType: see DaqHardwareSubType of the daqGetHardwareInfo function
pHwTypeStr: pointer to a string large enough to hold the hardware type string
HwTypeStrSize: Size of the hardware type string in bytes

Returns

Nothing in C/C++,
The hardware type string in Visual BASIC

Function Usage

This function converts a hardware type and sub type to a string.

Prototypes

C/C++

daqCvtHardwareType(DaqHardwareVersion HwType, DaqHardwareSubType HwSubType, PCHAR pHwTypeStr, DWORD HwTypeStrSize);

Visual BASIC

VBdaqCvtHardwareType$(ByVal HwType&, ByVal HwSubType&)

Program References

None

daqCvtLinearConvert

Format

```
daqCvtLinearConvert(counts, scans, fValues, nValues)
```

Purpose

`daqCvtLinearConvert` converts the ADC readings into floating point numbers using the linear relationship that was specified with `daqCvtLinearSetup`.

Parameter Summary

Parameter	Type	Description
counts	PWORD	Acquired ADC readings to be converted
scans	DWORD	Number of scans to be converted
fValues	PFLOAT	Array to hold the converted readings
nValues	DWORD	Size of the reading array

Parameter Values

`counts`: valid pointer to an array of integer words (2 bytes) which contain the data to perform the linear conversion

`scans`: the number of scans to be converted, from 1 to the size of the `counts` array (in scans)

`fValues`: valid pointer to an array of single precision floating point (4 bytes) values which will store the converted readings upon return of this command

`nValues`: the size of the `fValues` arrary should be at least equal to:

[`scans` * (channels in scan) * (the sample size, normally 2 bytes)]

Returns

`DerrNoError`   No error

Function Usage

> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The `daqCvtLinearConvert` function may be invoked repeatedly to perform multiple conversions, each using the same linear relationship. For convenience, `daqCvtLinearSetupConvert` can perform both setup and conversion.

Prototypes

C/C++

```
daqCvtLinearConvert(PWORD counts, DWORD scans, PFLOAT fValues, DWORD
nValues);
```

Visual BASIC

```
VBdaqCvtLinearConvert&(counts%, ByVal scans&, fValues!, ByVal nValues&)
```

Program References

```
None
```

daqCvtLinearSetup

Format

```
daqCvtLinearSetup(nscan, readingsPos, nReadings, signal1, voltage1,
signal2, voltage2, avg)
```

Purpose

daqCvtLinearSetup saves the data required for daqCvtLinearConvert to perform conversions.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Number of readings in a single scan
readingsPos	DWORD	Position within the scan of the first reading to convert
nReadings	DWORD	Number of consecutive ADC readings to convert
signal1	FLOAT	Transducer input signal that produces voltage1
voltage1	FLOAT	Transducer output voltage for input signal1
signal2	FLOAT	Transducer input signal that produces voltage2
voltage2	FLOAT	Transducer output voltage for input signal2
avg	DWORD	Type of averaging to use

Parameter Values

nscan: valid values range from 1 to 512

readingsPos: valid values range from 0 to (nscan − 1)

nReadings: valid values range from 1 to (nscan − readingsPos)

signal1: single precision floating point (4 bytes) value representing transducer input voltage

voltage1: single precision floating point (4 bytes) value representing transducer output voltage

signal2: single precision floating point (4 bytes) value representing transducer input voltage

voltage2: single precision floating point (4 bytes) value representing transducer output voltage

avg: see table below

Parameter Type Definitions

avg	
Definition	Description
avg = 0	Specifies block averaging--all scans are averaged together to compute a single value for each channel
avg = 1	Specifies no averaging--each scan's readings are converted into measured signals
avg ≥ 2	Specifies moving average--each scan's readings are averaged with the avg-1 preceding scans' readings before conversion.

Returns

DerrNoError   No error

Function Usage

> **Note!**
>
> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

For convenience, `daqCvtLinearSetupConvert` can perform both setup and conversion.

Moving Averages

To better illustrate the moving average feature of the `avg` parameter, suppose the `avg` value is set to 3. Since each scan's readings are averaged with the preceding scan, the results from the first scan (which has no preceding scan) are not averaged at all. However, the results from the second scan are averaged with the first scan, and the results from the third scan are averaged with the preceding two scans. After the third scan, each subsequent scan is averaged with the preceding two scans (since the `avg` value is 3, and `avg-1` is two). In another instance, if the `avg` value is set to 5, then each scan after the fifth scan would be averaged with the four preceding it.

Prototypes

C/C++

```
daqCvtLinearSetup(DWORD nscan, DWORD readingsPos, DWORD nReadings, FLOAT
signal1, FLOAT voltage1, FLOAT signal2, FLOAT voltage2, DWORD avg);
```

Visual BASIC

```
VBdaqCvtLinearSetup (ByVal nscan&, ByVal readingsPos&, ByVal nReadings&,
ByVal signal1!, ByVal voltage1!, ByVal signal2!, ByVal voltage2!, ByVal
avg&)
```

Program References

```
None
```

daqCvtLinearSetupConvert

Also See: **daqCvtLinearConvert, daqCvtLinearSetup,
daqCvtSetAdcRange**

Format

```
daqCvtLinearSetupConvert(nscan, readingsPos, nReadings, signal1, voltage1,
signal2, voltage2, avg, counts, scans, fValues, nValues)
```

Purpose

`daqCvtLinearSetupConvert` both sets up the linear conversion process and converts the ADC readings into floating point numbers.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Number of channels in a single scan
readingsPos	DWORD	Position within the scan of the first reading to convert
nReadings	DWORD	Number of consecutive ADC readings to convert
signal1	FLOAT	Transducer input signal that produces voltage1
voltage1	FLOAT	Transducer output voltage for input signal1
signal2	FLOAT	Transducer input signal that produces voltage2
voltage2	FLOAT	Transducer output voltage for input signal2
avg	DWORD	Type of averaging to use
counts	PWORD	Array of acquired ADC readings to be converted
scans	DWORD	Number of scans to be converted
fValues	PFLOAT	Array to hold the converted readings
nValues	DWORD	Size of the reading array

Parameter Values

nscan: valid values range from 1 to 512

readingsPos: valid values range from 0 to (nscan − 1)

nReadings: valid values range from 1 to (nscan − readingsPos)

signal1: single precision floating point (4 bytes) value representing transducer input voltage

voltage1: single precision floating point (4 bytes) value representing transducer output voltage

signal2: single precision floating point (4 bytes) value representing transducer input voltage

voltage2: single precision floating point (4 bytes) value representing transducer output voltage

avg: see table below

counts: valid pointer to an array of integer words (2 bytes) which contain the data to perform the linear conversion

scans: the number of scans to be converted from 1 to the size of the counts array (in scans)

fValues: valid pointer to an array of single precision floating point (4 bytes) values which will store the converted readings upon return from this command

nValues: the size of the fValues arrary should be at least equal to:
(scans * channels in scan * the sample size, normally 2)

Parameter Type Definitions

avg	
Definition	Description
avg = 0	Specifies block averaging--all scans are averaged together to compute a single value for each channel
avg = 1	Specifies no averaging--each scan's readings are converted into measured signals
avg ≥ 2	Specifies moving average--each scan's readings are averaged with the avg-1 preceding scans' readings before conversion.

Returns

DerrNoError   No error

Function Usage

> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> * Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The daqCvtLinearSetupConvert combines the functions of the both the daqCvtLinearConvert function and the daqCvtLinearSetup function.

Moving Averages

To better illustrate the moving average feature of the avg parameter, suppose the avg value is set to 3. Since each scan's readings are averaged with the preceding scan, the results from the first scan (which has no preceding scan) are not averaged at all. However, the results from the second scan are averaged with the first scan, and the results from the third scan are averaged with the preceding two scans. After the third scan, each subsequent scan is averaged with the preceding two scans (since the avg value is 3, and avg-1 is two). In another instance, if the avg value is set to 5, then each scan after the fifth scan would be averaged with the four preceding it.

Prototypes

C/C++

```
daqCvtLinearSetupConvert(DWORD nscan, DWORD readingsPos, DWORD nReadings,
FLOAT signal1, FLOAT voltage1, FLOAT signal2, FLOAT voltage2, DWORD avg,
PWORD counts, DWORD scans, PFLOAT fValues, DWORD nValues);
```

Visual BASIC

```
VBdaqCvtLinearSetupConvert&(ByVal nscan&, ByVal readingsPos&, ByVal
nReadings&, ByVal signal1!, ByVal voltage1!, ByVal signal2!, ByVal
voltage2!, ByVal avg&, counts%, ByVal scans&, fValues!, ByVal nValues&)
```

Program References

None

daqCvtRawDataFormat

Format

```
daqCvtRawDataFormat(buf, action, lastRetCount, scanCount, chanCount)
```

Purpose

daqCvtRawDataFormat converts raw data to a specified format.

Parameter Summary

Parameter	Type	Description
buf	PWORD	Pointer to the buffer containing the raw data
action	DaqAdcCvtAction	Type of conversion action to perform on the raw data
lastRetCount	DWORD	Last value in the retCount parameter returned from the daqAdcTransferGetStat function
scanCount	DWORD	Length of the raw data buffer in scans
chanCount	DWORD	Number of channels per scan in the raw data buffer

Parameter Values

buf: a pointer to an array of values from 0 to 65,535.

action: see table below

lastRetCount: valid values range from 1 to 4,294,967,295; however, memory limitations may apply

scanCount: valid values range from 1 to 4,294,967,295; however, memory limitations may apply

chanCount: valid values range from 1 to 72

Parameter Type Definitions

action-(DaqAdcCvtAction)	
Definition	Description
DacaUnpack	Decompresses raw data
DacaRotate	Reformats a circular buffer into a linear buffer

Returns

DerrNoError   No error

Function Usage

> **Note!** **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The `buf` parameter specifies the pointer to the data buffer containing the raw data. Prior to calling this function, this user-allocated buffer should already contain the entire raw data transfer. Upon completion, this data buffer will contain the converted data (the buffer must be able to contain all the converted data).

The `scanCount` parameter specifies the length of the raw buffer in scans. Since the converted data will overwrite the raw data in the buffer, make sure the specified buffer is large enough to contain all of the converted data.

Note! **This function should be called after the raw data has been acquired. See the transfer data functions (of the form `daqAdcTransfer`…) for more details on the collection of raw data.**

Prototypes

C/C++

```
daqCvtRawDataFormat(PWORD buf, DaqAdcCvtAction action, DWORD lastRetCount,
DWORD scanCount, DWORD chanCount);
```

Visual BASIC

```
VBdaqCvtRawDataFormat&(buf%, ByVal action&, ByVal lastRetCount&, ByVal
scanCount&, ByVal chanCount&)
```

Program References

```
None
```

daqCvtRtdConvert

Format

    daqCvtRtdConvert(counts, scans, temp, ntemp)

Purpose

`daqCvtRtdConvert` takes raw acquisiton data and converts it to a Celsius scale.

Parameter Summary

Parameter	Type	Description
counts	PWORD	Array of one or more scans of raw data as received from the device
scans	DWORD	Number of scans of raw data in `counts` parameter (number of pre-trigger + post-trigger scans)
temp	PSHORT	Variable array to hold converted temperatures
ntemp	DWORD	Size of temperature array

Parameter Values

`counts:` valid pointer to an array of integer words (2 bytes) which contain the raw data to perform the RTD conversion

`scans:` the number of scans to be converted from 1 to the size of the `counts` array (in scans)

`temp:` valid pointer to an array of single precision floating point (4 bytes) values which will store the converted readings upon return from this command

`ntemp:` The value of the `ntemp` parameter should be equal to number of RTDs specified in setup times the number of scans. If averaging is used, then `ntemp` should be equal to: [(the number of RTDs) * `scans` * (the value of the `avg` parameter as set in the `daqCvtRtdSetup` function)]

Returns

DerrRtdNoSetup	Setup was not called
DerrRtdTArraySize	Temperature array is not large enough
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The `daqCvtRtdConvert` function takes acquired data from a resistance temperature detector (RTD) and converts them to temperature readings in Celsius, with a resolution in tenths of a degree (0.1°C). Before this commannd is executed, the RTD conversion should be configured using the `daqCvtRtdSetup` function. The total number of data conversions [`scans` * (RTD channels per scan) * 4] must be less than 32,767.

For convenience, both setup and conversion can be performed at once by `daqCvtRtdSetupConvert`.

Using Resistance Temperature Detectors (RTDs)

Many devices can measure temperatures through use of DBK9 RTD expansion cards. Up to 8 RTDs can attach to each DBK9 and up to 32 DBK9s can be attached to a single device; allowing for up to 256 temperature readings. The software currently supports 100, 500, and 1000 ohm RTDs. The RTDs must all be of the same type, and the reading groups for all RTDs must follow each other in the scan sequence. All non-RTD data conversion, if any, must be done by other means.

The RTD measurement functions are designed for simple temperature measurement in which each RTD channel is read 4 times. These 4 readings must be grouped together consecutively during a scan in the following order, with the appropriate gain: `Dbk9VoltageA` (gain=0), `Dbk9VoltageB` (gain=1), `Dbk9VoltageC` (gain=3), and `Dbk9VoltageD` (gain=3).

The temperature conversion functions use input data from one or more device scans. They take 4 voltage readings for each RTD channel, apply the appropriate averaging method, convert the voltages to a resistance, and then (using the appropriate curves for the RTD type) convert the resistance into a temperature. To illustrate, suppose the data in the following table was gathered:

Scan	Readings Channel 0				Readings Channel 1			
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	Ch 0 Va	Ch 0 Vb	Ch 0 Vc	Ch 0 Vd	Ch 1 Va	Ch 1 Vb	Ch 1 Vc	Ch 1 Vd
2	Ch 0 Va	Ch 0 Vb	Ch 0 Vc	Ch 0 Vd	Ch 1 Va	Ch 1 Vb	Ch 1 Vc	Ch 1 Vd
3	Ch 0 Va	Ch 0 Vb	Ch 0 Vc	Ch 0 Vd	Ch 1 Va	Ch 1 Vb	Ch 1 Vc	Ch 1 Vd
4	Ch 0 Va	Ch 0 Vb	Ch 0 Vc	Ch 0 Vd	Ch 1 Va	Ch 1 Vb	Ch 1 Vc	Ch 1 Vd
5	Ch 0 Va	Ch 0 Vb	Ch 0 Vc	Ch 0 Vd	Ch 1 Va	Ch 1 Vb	Ch 1 Vc	Ch 1 Vd

The 4 readings for each channel are grouped together in order. If this scan data is passed to `daqCvtRtdConvert`(through the `counts` parameter) with averaging disabled (`avg` parameter in `daqCvtRtdSetup` set to 1), the function will return the temp parameters shown in the table. Temperatures returned will be in tenths of a degree Celsius.

Temperatures		
Scan	**0**	**1**
1	Ch 0 °C	Ch 1 °C
2	Ch 0 °C	Ch 1 °C
3	Ch 0 °C	Ch 1 °C
4	Ch 0 °C	Ch 1 °C
5	Ch 0 °C	Ch 1 °C

If the scan data is passed to `daqCvtRtdConvert` (through the `counts` parameter) with averaging set to block averaging (`avg` parameter in `daqCvtRtdSetup` set to 0), the function will return the `temp` parameter values shown in the table.

Temperatures		
	0	**1**
Average of all Temps	Ch 0 °C	Ch 1 °C

Prototypes

C/C++

```
daqCvtRtdConvert(PWORD counts, DWORD scans, PSHORT temp, DWORD ntemp);
```

Visual BASIC

```
VBdaqCvtRtdConvert&(counts%, ByVal scans&, temp%, ByVal ntemp&)
```

Program References

```
None
```

daqCvtRtdSetup

Format

```
daqCvtRtdSetup(nscan, startPosition, nRtd, rtdType, avg)
```

Purpose

`daqCvtRtdSetup` sets up parameters for subsequent resistance temperature detector (RTD) data conversions.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Total number of channel readings in a single scan
startPosition	DWORD	Position of the first RTD reading group in the scan
nRtd	DWORD	Number of the RTDs' signals that are to be converted to temperature values
rtdType	RtdType	Type of RTDs being used to collect data
avg	DWORD	Type of averaging to use

Parameter Values

nscan: valid values range from 1 to 512; this number should be equivalent to (the number of RTDs * 4) + any additional channels

startPosition: valid values range from 1 to 507; the first scan position will be position 0, and the last will be position (nscan-1)

nRtd: valid values range from 1 to 128

rtdType: see table below

avg: see table below

Parameter Type Definitions

rtdType-(RtdType)	
Definition	Description
Dbk9RtdType100	Value for a 100 ohm RTD
Dbk9RtdType500	Value for a 500 ohm RTD
Dbk9RtdType1K	Value for a 1000 ohm RTD

avg	
Definition	Description
avg = 0	Specifies block averaging--all scans are averaged together to compute a single value for each channel
avg = 1	Specifies no averaging--each scan's readings are converted into measured signals
avg ≥ 2	Specifies moving average--each scan's readings are averaged with the avg-1 preceding scans' readings before conversion.

Returns

DerrRtdParam	Setup parameter out-of-range
DerrRtdValue	Invalid RTD type
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

> **Note!**
> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

For more detailed information on using RTDs refer to the entry for the `daqCvtRtdConvert` function. For convenience, both setup and conversion can be performed at once by `daqCvtRtdSetupConvert`.

Moving Averages

To better illustrate the moving average feature of the `avg` parameter, suppose the `avg` value is set to 3. Since each scan's readings are averaged with the preceding scan, the results from the first scan (which has no preceding scan) are not averaged at all. However, the results from the second scan are averaged with the first scan, and the results from the third scan are averaged with the preceding two scans. After the third scan, each subsequent scan is averaged with the preceding two scans (since the `avg` value is 3, and `avg-1` is two). In another instance, if the `avg` value is set to 5, then each scan after the fifth scan would be averaged with the four preceding it.

Prototypes

C/C++

```
daqCvtRtdSetup(DWORD nscan, DWORD startPosition, DWORD nRtd, RtdType
rtdType, DWORD avg);
```

Visual BASIC

```
VBdaqCvtRtdSetup&(ByVal nscan&, ByVal startPosition&, ByVal nRtd&, ByVal
rtdType&, ByVal avg&)
```

Program References

```
None
```

daqCvtRtdSetupConvert

Format

```
daqCvtRtdSetupConvert(nscan, startPosition, nRtd, rtdType, avg, counts,
scans, temp, ntemp)
```

Purpose

`daqCvtRtdSetupConvert` both sets up the resistance temperature detector (RTD) data conversion process and converts that RTD data to a Celsius scale.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Total number of channel readings in a scan
startPosition	DWORD	Position of the first RTD reading group in the scan
nRtd	DWORD	Number of RTDs beings scanned
rtdType	RtdType	Type of RTDs being used
avg	DWORD	Type of averaging to be used
counts	PWORD	Pointer to raw A/D data from one or more scans
scans	DWORD	Number of scans of raw data in counts (number of pre-trigger + post-trigger scans)
temp	PSHORT	Variable array to hold converted temperatures
ntemp	DWORD	Size of temperature array

Parameter Values

nscan: valid values range from 1 to 512; this number should be equivalent to:

[(the number of RTDs * 4) + (any additional channels)]

startPosition: valid values range from 1 to 507; the first scan position will be position 0, and the last will be position (nscan-1)

nRtd: valid values range from 1 to 128

rtdType: see table below

avg: see table below

counts: valid pointer to an array of integer words (2 bytes) which contain the raw data to perform the RTD conversion

scans: the number of scans to be converted from 1 to the size of the counts array (in scans)

temp: valid pointer to an array of single precision floating point (4 bytes) values which will store the converted readings upon return from this command

ntemp: the value of the ntemp parameter should be equal to number of RTDs specified in setup times the number of scans. If averaging is used, then ntemp should be equal to:

[(number of RTDs) * (scans) * (the value of the avg parameter as set in the daqCvtRtdSetup function)].

Parameter Type Definitions

RtdType-(`RtdType`)	
Definition	Description
`Dbk9RtdType100`	Value for a 100 ohm RTD
`Dbk9RtdType500`	Value for a 500 ohm RTD
`Dbk9RtdType1K`	Value for a 1000 ohm RTD

Avg	
Definition	Description
`avg = 0`	Specifies block averaging--all scans are averaged together to compute a single value for each channel
`avg = 1`	Specifies no averaging--each scan's readings are converted into measured signals
`avg ≥ 2`	Specifies moving average--each scan's readings are averaged with the `avg-1` preceding scans' readings before conversion.

Returns

`DerrRtdParam`	Setup parameter out-of-range
`DerrRtdTArraySize`	Temperature storage array not large enough
`DerrRtdValue`	Invalid RTD type
`DerrNoError`	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

> For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).
>
> For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).
>
> See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.
>
> * Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The `daqCvtRtdSetupConvert` combines the operations of the both the `daqCvtRtdConvert` function and the `daqCvtRtdSetup` function. For more detailed information on using RTDs refer to the entry for the `daqCvtRtdConvert` function.

Moving Averages

To better illustrate the moving average feature of the `avg` parameter, suppose the `avg` value is set to 3. Since each scan's readings are averaged with the preceding scan, the results from the first scan (which has no preceding scan) are not averaged at all. However, the results from the second scan are averaged with the first scan, and the results from the third scan are averaged with the preceding two scans. After the third scan, each subsequent scan is averaged with the preceding two scans (since the `avg` value is 3, and `avg-1` is two). In another instance, if the `avg` value is set to 5, then each scan after the fifth scan would be averaged with the four preceding it.

Prototypes

C/C++

```
daqCvtRtdSetupConvert(DWORD nscan, DWORD startPosition, DWORD nRtd,
RtdType rtdType, DWORD avg, PWORD counts, DWORD scans, PSHORT temp, DWORD
ntemp);
```

Visual BASIC

```
VBdaqCvtRtdSetupConvert&(ByVal nscan&, ByVal startPosition&, ByVal nRtd&,
ByVal rtdType&, ByVal avg&, counts%, ByVal scans&, temp%, ByVal ntemp&)
```

Program References

```
None
```

daqCvtSetAdcRange

Format

daqCvtSetAdcRange (Admin, Admax)

Purpose

daqCvtSetAdcRange sets the ADC range for use by the conversion functions (i.e., all functions of the form daqCvt…).

Parameter Summary

Parameter	Type	Description
Admin	FLOAT	A/D minimum voltage range (in volts)
Admax	FLOAT	A/D maximum voltage range (in volts)

Parameter Values:

Admin: valid values depend on the device and signal being processed
Admax: valid values depend on the device and signal being processed

Returns

DerrNoError   No error

Function Usage

> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The daqCvtSetAdcRange function is used by the conversion functions to establish the range of an acquisition. The voltage range is dependent on the acquisition's gain values as set in the daqAdcSetScan function—see daqAdcSetScan for more information.

Prototypes

C/C++

daqCvtSetAdcRange(FLOAT Admin, FLOAT Admax);

Visual BASIC

VBdaqCvtSetAdcRange&(ByVal ADmin!, ByVal ADmax!)

Program References

None

---

daqCvtTCConvert and daqCvTCConvertF

Also See: **daqCvtTCSetup, daqCvtTCSetupConvert, daqCvtTCSetupConvertF**

Format

```
daqCvtTCConvert(counts, scans, temp, ntemp)
```

Purpose

`daqCvtTCConvert` and `daqCvtTCConvertF` each convert raw data acquired from a thermocouple (T/C) to a Celsius scale. See Note 1.

Parameter Summary

Parameter	Type	Description
counts	PWORD	Array of one or more scans of raw data
scans	DWORD	Number of scans of data in `counts` array (number of pre-trigger + post-trigger scans)
temp	PSHORT	Variable array to hold converted temperature results (**Note 1**)
ntemp	DWORD	Number of entries in the temperature array

Parameter Values

`counts`: valid values range from 0 to 65,536 (each raw data item may be any 16-bit value)

`scans`: valid value range from 1 to 4,294,967,295; however, memory limitations may apply

`temp`: valid converted values stored in this array range from -2,000 (-200°C) to +13,720 (+1,372°C), depending on the thermocouple type

`ntemp`: value of the `ntemp` parameter should be equal to number of T/Cs specified in setup times the number of scans. If averaging is used, then `ntemp` should be equal to: [(number of T/Cs) * (scans) * (the value of the `avg` parameter as set in the `daqCvtTCSetup` function)]

Returns

DerrTCE_NOSETUP	Setup was not called
DerrTCE_PARAM	Parameter out-of-range
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> *** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.**

Note 1: daqCvtTCConvert converts temperature readings to values in tenths of degrees Celsius (0.1°C). daqCvtTCConvertF stores actual temperature values, i.e., actual instead of a value converted to the nearest tenth of a degree °C.

The `daqCvtTCConvert` takes raw A/D readings from a thermocouple (T/C) and converts them to temperature readings in tenths of degrees Celsius (0.1°C), **see Note 1**. The `temp` array actually stores the converted values as 10 times the Celsius temperature--for example, 50°C would be represented as 500 and -10°C would be -100. The value in the `ntemp` parameter is checked by the functions to avoid writing past the end of the array.

Note 1: Unlike daqCvtTCConvert, daqCvtTCConvertF stores actual temperature values instead of values converted to temperature readings in tenths of degrees Celsius.

Before this commannd is executed, the conversion should be configured using the `daqCvtTCSetup` function. For convenience, both setup and conversion can be performed at once by `daqCvtTCSetupConvert`. All non-thermocouple data conversion, if any, must be done by other means.

The total number of data conversions (`scan` * channels per scan) must be less than 32,768.

Using Thermocouples (T/Cs)

Some products can measure thermocouple temperatures using a internal thermocouple channels or expansion thermocouple modules or cards. These products normally include a cold-junction compensation circuit (CJC) channel. This channel must be used in order to compensate for the temperature at the juncture of the thermocouple to the device. Also, some temperature measurement devices incorporate a zero offset channel which also can be used to null any offset errors inherent in the system. The CJC channel and zero-offset channel location depend upon the module or card type. Please refer to your hardware documentation for your temperature measurement to determine the location of the CJC and zero channels (if applicable). You can also refer to the examples listed below for programming your specific device. The following example uses Channel 0 as the CJC channel and Channel 1 is shorted for performing auto-zero compensation (as described in the "Correcting Data" section below). This particular example assumes 14 TC channels so Channels 2 through 15 accept thermocouples for temperature measurement. Up The software supports type J, K, T, E, N28, N14, S, R and B thermocouples.

The three temperature measurement conversion functions (`daqCvtTCConvert, daqCvtTCSetup,`and `daqCvtTCSetupConvert`) are designed for temperature measurement where the cold-junction compensation circuit (CJC) channel (channel 0) reading from the T/C card is immediately followed in the scan sequence by the T/C channel readings. All readings must be from the same type of T/C. The CJC and T/C readings are taken with the optimal gains (as described in the "T/C Gains Settings" section below).

How The T/C Acquisition And Conversion Works

The temperature conversion functions take input data from one or more scans from the device. They then examine the CJC and thermocouple readings within that scan and (after optional averaging), convert them to temperatures which are stored as output. The procedure for the scan varies, depending on whether auto-zeroing has been enabled or disabled setting If auto-zeroing has been enabled, the scan readings would resemble the following table:

Scan	Reading					
	0	**1**	**2**	**3**	**4**	**5**
1	CJC Zero	J Zero	CJC	J1a	J1b	J1c
2	CJC Zero	J Zero	CJC	J2a	J2b	J2c
3	CJC Zero	J Zero	CJC	J3a	J3b	J3c
4	CJC Zero	J Zero	CJC	J4a	J4b	J4c

The first 2 readings of each scan are non-temperature voltage readings to compensate for the CJC circuit and the shorted channel 0. The third reading is from the CJC, and the remaining 3 readings are from 3 type J thermocouples. The first 2 readings will be used to remove offset errors in the CJC and T/C reading. However, if the auto-zero feature is disabled, the first 2 readings will be ignored, as shown in the following table:

Scan	Reading					
	Shorted Channels		**0**	**1**	**2**	**3**
1	V (ignored)	V (ignored)	CJC	J1a	J1b	J1c
2	V (ignored)	V (ignored)	CJC	J2a	J2b	J2c
3	V (ignored)	V (ignored)	CJC	J3a	J3b	J3c
4	V (ignored)	V (ignored)	CJC	J4a	J4b	J4c

In either case, the CJC and T/C readings are used to produce one temperature result for each T/C reading. Thus, the 24 original readings are reduced to 12 temperatures.

To measure temperatures, the scan must be set up so the T/C measurements consecutively follow their corresponding CJC measurement (the CJC measurement need not be the first element in the scan). If auto-zeroing is enabled, the CJC measurement must be preceded by both a CJC zero measurement and a T/C zero measurement.

All of the thermocouples converted with a single invocation of the conversion functions must be of the same type: J, K, T, E, N28, N14, S, R, or B. To measure with more than one type of thermocouple, they must be sorted by type within the scan, and each type must be preceded by the related CJC.

The scan is not restricted to thermocouple measurements. The scan may include other types of signals such as voltage, current, or digital input; but conversion of these readings cannot be performed by the temperature conversion functions cannot handle them.

T/C Gains Settings

The temperature measurements must be made with the correct gain settings. The gain settings for the different thermocouple types depend on the bipolar/unipolar setting of the device, and the type of card being used. PC cards can only be set for bipolar acquisitions, and have their own set of gain codes. The gains settings should be established with the `daqAdcSetScan` function, as specified in the following tables.

Note! Unipolar operations are not recommended for thermocouple measurement unless the measured temperatures will be greater than the device temperature.

DBK19 and DBK52 Gain Codes			
T/C Type	**Unipolar Gain Code**	**Bipolar Gain Code**	**Bipolar Gain Codes for** Daq/2000 Series devices* and PC Cards
CJC	Dbk19UniCJC	Dbk19BiCJC	Dbk19PCCBiCJC
J	Dbk19UniTypeJ	Dbk19BiTypeJ	Dbk19PCCBiTypeJ
K	Dbk19UniTypeK	Dbk19BiTypeK	Dbk19PCCBiTypeK
T	Dbk19UniTypeT	Dbk19BiTypeT	Dbk19PCCBiTypeT
E	Dbk19UniTypeE	Dbk19BiTypeE	Dbk19PCCBiTypeE
N28	Dbk19UniTypeN28	Dbk19BiTypeN28	Dbk19PCCBiTypeN28
N14	Dbk19UniTypeN14	Dbk19BiTypeN14	Dbk19PCCBiTypeN14
S	Dbk19UniTypeS	Dbk19BiTypeS	Dbk19PCCBiTypeS
R	Dbk19UniTypeR	Dbk19BiTypeR	Dbk19PCCBiTypeR
B	Dbk19UniTypeB	Dbk19BiTypeB	Dbk19PCCBiTypeB

* Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

DBK81,DBK82,DBK83,DBK84 and DaqBook/2020 Gain Codes			
T/C Type	Unipolar Gain Code	Bipolar Gain Code	Bipolar Gain Codes for Daq/2000 Series devices* and PC Cards
CJC	Dbk81UniCJC	Dbk81BiCJC	Dbk81PCCBiCJC
J	Dbk81UniTypeJ	Dbk81BiTypeJ	Dbk81PCCBiTypeJ
K	Dbk81UniTypeK	Dbk81BiTypeK	Dbk81PCCBiTypeK
T	Dbk81UniTypeT	Dbk81BiTypeT	Dbk81PCCBiTypeT
E	Dbk81UniTypeE	Dbk81BiTypeE	Dbk81PCCBiTypeE
N28	Dbk81UniTypeN28	Dbk81BiTypeN28	Dbk81PCCBiTypeN28
N14	Dbk81UniTypeN14	Dbk81BiTypeN14	Dbk81PCCBiTypeN14
S	Dbk81UniTypeS	Dbk81BiTypeS	Dbk81PCCBiTypeS
R	Dbk81UniTypeR	Dbk81BiTypeR	Dbk81PCCBiTypeR
B	Dbk81UniTypeB	Dbk81BiTypeB	Dbk81PCCBiTypeB

* Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

DBK90 and DBK100 Gain Codes			
T/C Type	Unipolar Gain Code	Bipolar Gain Code	Bipolar Gain Codes for Daq/2000 Series devices* and PC Cards
CJC	Dbk90UniCJC	Dbk90BiCJC	Dbk90PCCBiCJC
J	Dbk90UniTypeJ	Dbk90BiTypeJ	Dbk90PCCBiTypeJ
K	Dbk90UniTypeK	Dbk90BiTypeK	Dbk90PCCBiTypeK
T	Dbk90UniTypeT	Dbk90BiTypeT	Dbk90PCCBiTypeT
E	Dbk90UniTypeE	Dbk90BiTypeE	Dbk90PCCBiTypeE
N28	Dbk90UniTypeN28	Dbk90BiTypeN28	Dbk90PCCBiTypeN28
N14	Dbk90UniTypeN14	Dbk90BiTypeN14	Dbk90PCCBiTypeN14
S	Dbk90UniTypeS	Dbk90BiTypeS	Dbk90PCCBiTypeS
R	Dbk90UniTypeR	Dbk90BiTypeR	Dbk90PCCBiTypeR
B	Dbk90UniTypeB	Dbk90BiTypeB	Dbk90PCCBiTypeB

* Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

Note! **Unipolar operations are not recommended for thermocouple measurement unless the measured temperatures will be greater than the device temperature.**

When using gain values from the preceding tables to measure TCs, the following temperature ranges apply:

Thermocouple mV Outputs For Temperature Ranges Depending on Ambient Temperature						
T/C Type	Measured Temperature Range @ 0°C ambient		Measured Temperature Range @ 25°C ambient		Measured Temperature Range @ 50°C ambient	
	Temp (°C)	0°C Output (mV)	Temp (°C)	25°C Output (mV)	Temp (°C)	50°C Output (mV)
J	-200 to 760	-7.9 to 42.9	-200 to 760	-9.2 to 41.6	-200 to 760	-11.8 to 39.0
K	-200 to 1372	-5.9 to 54.9	-200 to 1372	-6.9 to 53.9	-200 to 1372	-8.9 to 52.9 (50.0)
T	-200 to 400	-5.6 to 20.9	-200 to 400	-6.6 to 19.9	-200 to 400	-8.7 to 17.7
E	-270 to 1000	-9.8 to 76.4	-270 to 1000	-11.3 to 74.9	-270 to 1000	-14.5 to 71.7
N28	-270 to 400	-4.3 to 13.0	-270 to 400	-5.0 to 12.3	-270 to 400	-6.4 to 10.9
N14	0 to 1300	0.0 to 47.5	0 to 1300	-0.7 to 46.8	0 to 1300	-2.0 to 45.5
S	-50 to 1780	-0.2 to 18.8	-50 to 1780	-0.4 to 18.7	-50 to 1780	-0.7 to 18.4
R	-50 to 1780	-0.2 to 21.3	-50 to 1780	-0.4 to 21.1	-50 to 1780	-0.7 to 20.8
B	50 to 1780	0.0 to 13.4	50 to 1780	0.0 to 13.4	50 to 1780	0.0 to 13.4

Correcting Data

Two software techniques (software calibration and zero compensation) can be used to increase the accuracy of a temperature module or card.

- Software calibration uses gain and offset calibration constants, unique to each card, to compensate for inherent errors on the card.
- Zero compensation is a method by which any offset voltage on the card can be removed at run-time. This is done by measuring a shorted channel at the same gain on the actual input to find the offset, and subtracting this value from the actual reading.

The thermocouple linearization function has a special auto-zero compensation feature that will perform zero compensation on the raw thermocouple data before linearizing when using a temperature module or card. The auto-zero feature is enabled by default, but can be disabled using the daqAutoZeroCompensate function. It is not available when using unipolar mode.

If a temperature module or card is used with *auto-zeroing enabled*, the CJC channel reading described above must be preceded by 2 readings from the shorted channel (channel 1). The first shorted reading must be at the same gain setting as the CJC reading. The other shorted reading must be at the gain of the T/C to be converted. If, instead, *software calibration* is used with the temperature module or card, the calibration constants for the card to be used should be entered into the calibration file.

Prototypes

C/C++

```
daqCvtTCConvert(PWORD counts, DWORD scans, PSHORT temp, DWORD ntemp);
```

Visual BASIC

```
VBdaqCvtTCConvert&(counts%, ByVal scans&, temp%, ByVal ntemp&)
```

Program References

```
DBK19EX.CPP,DBK81EX.CPP, DBK82_83EX.CPP, DBK90EX.CPP
```

daqCvtTCSetup

Also See: **daqCvtTCConvert, daqCvtTCConvertF, daqCvtTCSetupConvert, daqCvtTCSetupConvertF**

Format

```
daqCvtTCSetup(nscan, cjcPosition, ntc, tcType, bipolar, avg)
```

Purpose

daqCvtTCSetup sets up the conversion of data acquired from a thermocouple (T/C).

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Number of channel readings in a single scan
cjcPosition	DWORD	Position of the cold-junction compensation circuit (CJC) reading within each scan--not the CJC zero reading, if any
ntc	DWORD	Number of thermocouple signals that are to be converted to temperature values
tcType	TCType	Type of thermocouples that generated the measurements
bipolar	BOOL	Must be set true if the readings were acquired with the device set for bipolar operation; must be set false (zero) for unipolar operation
avg	DWORD	Type of averaging to be performed

Parameter Values

nscan: valid values range from 2 to 512; this number should be equal to
(the number of T/Cs + the number of CJCs + any zero readings + any additional voltage readings)

cjcPosition: valid values depend on whether or not zero compensation is
being used: values range from 0 to
(nscan-2) with no zero compensation, and from 2 to (nscan-2) with zero compensation

ntc: valid values range from 1 to [(nscan-1)-cjcPosition]

tcType: see table below

bipolar: valid values are either true ($\neq 0$) or false ($= 0$)

avg: see table below

Parameter Type Definitions

tcType-(TCType)	
Definition	Description
Dbk19TCTypeJ	Specifies a Type J thermocouple
Dbk19TCTypeK	Specifies a Type K thermocouple
Dbk19TCTypeT	Specifies a Type T thermocouple
Dbk19TCTypeE	Specifies a Type E thermocouple
Dbk19TCTypeN28	Specifies a Type N28 thermocouple
Dbk19TCTypeN14	Specifies a Type N14 thermocouple
Dbk19TCTypeS	Specifies a Type S thermocouple
Dbk19TCTypeR	Specifies a Type R thermocouple
Dbk19TCTypeB	Specifies a Type B thermocouple

avg	
Definition	Description
avg = 0	Specifies block averaging--all scans are averaged together to compute a single value for each channel
avg = 1	Specifies no averaging--each scan's readings are converted into measured signals
avg ≥ 2	Specifies moving average--each scan's readings are averaged with the avg-1 preceding scans' readings before conversion

Returns

DerrTCE_TYPE	Invalid thermocouple type
DerrTCE_PARAM	Parameter out-of-range
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> *** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.**

Note: The following example assumes a CJC position of "0" and a Channel Zero position of "1." The position of these channels is device dependent. Refer to your device-specific hardware documentation to determine the appropriate position of these channels for your application.

The `daqCvtTCSetup` function is used to set the values required to convert raw thermocouple (T/C) data to data on a Celsius scale. It must be follwed with the `daqCvtTCConvert` function to actually convert the data. For convenience, both setup and conversion can be performed at once by `daqCvtTCSetupConvert`. All of the T/C conversion functions (i.e., functions of the form `daqCvtTC...`) can convert several consecutive scans of data in a single invocation.

The `cjcPosition` parameter determines the positon of the CJC reading in the scan group. Each scan of temperature data must include a reading of the CJC signal on the expansion board to which the thermocouples are attached. If zero compensation is being used, then the first two scans in each group are reserved for zero compensation data, and the CJC scan must follow immediately thereafter. The illustration below shows a how a typical T/C acquisition with zero compensation would collect data:
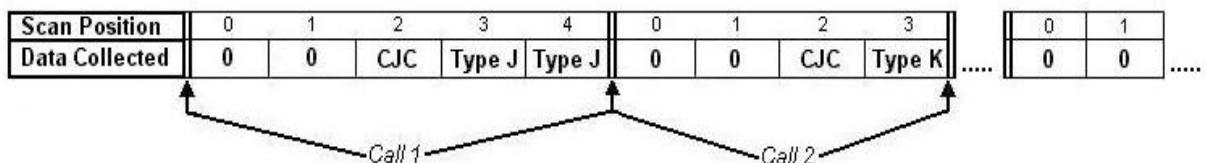
The first reading of the scan is position 0, and the last reading is (`nscan-1`). The thermocouple signal readings must immediately follow the CJC reading in the scan data, so the first thermocouple signal must be at scan position (`cjcPosition+1`), the next is at (`cjcPosition+2`), and so on. In the illustration, we see that two type J thermocouple readings are taken directly after the CJC scan. In the following scan group, two zero scans are followed by a CJC scan, which is itself followed by one type K thermocouple scan. Note that for any single call of the conversion function, all thermocouples in a scan group must be of the same type.

If zero compensation is not being used in a scan, the CJC scan is placed in position 0, followed by the thermocouple scans. The following illustration shows how this data would be organized:
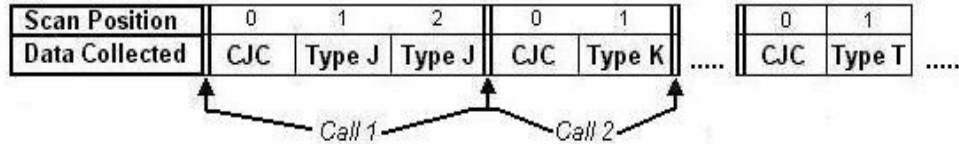
The CJC readings must be taken with the appropriate gains set with the `daqAdcSetScan` function. The required gain settings for the CJC and thermocouple channels change depending on the unipolar/bipolar mode—see the `daqCvtTCConvert` function for specific info on T/C gains settings.

Moving Averages

To better illustrate the moving average feature of the `avg` parameter, suppose the `avg` value is set to 3. Since each scan's readings are averaged with the preceding scan, the results from the first scan (which has no preceding scan) are not averaged at all. However, the results from the second scan are averaged with the first scan, and the results from the third scan are averaged with the preceding two scans. After the third scan, each subsequent scan is averaged with the preceding two scans (since the `avg` value is 3, and `avg-1` is two). In another instance, if the `avg` value is set to 5, then each scan after the fifth scan would be averaged with the four preceding it.

Prototypes

C/C++

```
daqCvtTCSetup(DWORD nscan, DWORD cjcPosition, DWORD ntc, TCType tcType,
BOOL bipolar, DWORD avg);
```

Visual BASIC

```
VBdaqCvtTCSetup&(ByVal nscan&, ByVal cjcPosition&, ByVal ntc&, ByVal
tcType&, ByVal bipolar&, ByVal avg&)
```

Program References

```
DBK19EX.CPP
```

daqCvtTCSetupConvert and daqCvtTCSetupConvertF

Format

```
daqCvtCSetupConvert(nscan, cjcPosition, ntc, tcType, bipolar, avg, counts,
scans, temp, ntemp)
```

Purpose

daqCvtTCSetupConvert and daqCvtTCSetupConvertF are each used to set up and convert raw thermocouple (T/C) data to data on a Celsius scale. See Note 1.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Number of channel readings in a single scan
cjcPosition	DWORD	Position of the CJC reading within the scan
ntc	DWORD	Number of thermocouple readings that immediately follow the CJC reading within the scan
tcType	TCType	Type of thermocouples being measured
bipolar	BOOL	Must be set `true` if the readings were acquired with the device set for bipolar operation; must be set `false` for unipolar operation
avg	DWORD	Type of averaging to use
counts	PWORD	Array of raw data from one or more scans
scans	DWORD	Number of scans of raw data in `counts` array (number of pre-trigger + post-trigger scans)
temp	PSHORT	Variable array to hold converted temperature results (see Note 1)
ntemp	DWORD	Number of entries in the temperature array

Parameter Values

nscan: valid values range from 2 to 512; this number should be equal to: [(the number of T/Cs) + (the number of CJCs) + (any zero readings) + (any additional voltage readings)]

cjcPosition: valid values depend on whether or not zero compensation is being used: values range from 0 to (nscan-2) with no zero compensation, and from 2 to (nscan-2) with zero compensation

ntc: valid values range from 1 to [(nscan-1)-cjcPosition]

tcType: see table below

bipolar: valid values are either true ($\neq 0$) or false ($= 0$)

avg: see table below

counts: valid values range from 0 to 65,536 (each raw data item may be any 16-bit value)

scans: valid value range from 1 to 4,294,967,295; however, memory limitations may apply

temp: valid converted values stored in this array range from –2,000 (-200°C) to +13,720 (+1,372°C), depending on the thermocouple type. (See Note 1)

ntemp: the value of the ntemp parameter should be equal to number of T/Cs specified in setup times the number of scans. If averaging is used, then ntemp should be equal to: [(number of T/Cs) * (scans) * (the value of the avg parameter as set in the daqCvtTCSetup function)]

Note 1: daqCvtTCSetupConvert converts temperature readings to values in tenths of degrees Celsius (0.1°C). daqCvtTCSetupConvertF stores actual temperature values, i.e., actual instead of a value converted to the nearest tenth of a degree °C.

Parameter Type Definitions

tcType-(TCType)	DBK19 and DBK52
Definition	**Description**
Dbk19TCTypeJ	Specifies a Type J thermocouple
Dbk19TCTypeK	Specifies a Type K thermocouple
Dbk19TCTypeT	Specifies a Type T thermocouple
Dbk19TCTypeE	Specifies a Type E thermocouple
Dbk19TCTypeN28	Specifies a Type N28 thermocouple
Dbk19TCTypeN14	Specifies a Type N14 thermocouple
Dbk19TCTypeS	Specifies a Type S thermocouple
Dbk19TCTypeR	Specifies a Type R thermocouple
Dbk19TCTypeB	Specifies a Type B thermocouple

tcType-(TCType)	DBK81,DBK82,DBK83,DBK84,and DaqBook/2020
Definition	**Description**
Dbk81TCTypeJ	Specifies a Type J thermocouple
Dbk81TCTypeK	Specifies a Type K thermocouple
Dbk81TCTypeT	Specifies a Type T thermocouple
Dbk81TCTypeE	Specifies a Type E thermocouple
Dbk81TCTypeN28	Specifies a Type N28 thermocouple
Dbk81TCTypeN14	Specifies a Type N14 thermocouple
Dbk81TCTypeS	Specifies a Type S thermocouple
Dbk81TCTypeR	Specifies a Type R thermocouple
Dbk81TCTypeB	Specifies a Type B thermocouple

tcType-(TCType)	DBK90 and DBK100
Definition	**Description**
Dbk90TCTypeJ	Specifies a Type J thermocouple
Dbk90TCTypeK	Specifies a Type K thermocouple
Dbk90TCTypeT	Specifies a Type T thermocouple
Dbk90TCTypeE	Specifies a Type E thermocouple
Dbk90TCTypeN28	Specifies a Type N28 thermocouple
Dbk90TCTypeN14	Specifies a Type N14 thermocouple
Dbk90TCTypeS	Specifies a Type S thermocouple
Dbk90TCTypeR	Specifies a Type R thermocouple
Dbk90TCTypeB	Specifies a Type B thermocouple

avg	
Definition	**Description**
avg = 0	Specifies block averaging--all scans are averaged together to compute a single value for each channel
avg = 1	Specifies no averaging--each scan's readings are converted into measured signals
avg ≥ 2	Specifies moving average--each scan's readings are averaged with the avg-1 preceding scans' readings before conversion

Returns

DerrTCE_TYPE	Invalid thermocouple type
DerrTCE_PARAM	Parameter out-of-range
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

> **Note!**
>
> **For all functions of the form `daqCvt...`, raw data to be converted *must* be unsigned (i.e., the `dafUnsigned` value must be set using the `daqAdcSetScan` function).**
>
> **For T/C and RTD conversion, users of Daq/2000 Series devices* and/or Daq PC Cards must be sure that the main unit is using ten volt range gains settings, either unipolar (0 V to 10 V) or bipolar (-5 V to +5 V).**
>
> **See the "T/C Gains Settings" section in the `daqCvtTCConvert` function entry for the appropriate settings.**
>
> ***** Daq/2000 Series devices include DaqBook/2000 Series, DaqScan/2000 Series, DaqLab/2000 Series, and DaqBoard/2000 Series devices.

The daqCvtTCSetupConvert function combines the operationns of the both the daqCvtTCConvert function and the daqCvtTCSetup function. For more detailed information on using thermocouples (T/Cs) and converting data refer to the entries for daqCvtTCConvert and daqCvtTCSetup

The daqCvtTCSetupConvertF function combines the operationns of the both the daqCvtTCConvertF function and the daqCvtTCSetup function. For more detailed information on using thermocouples (T/Cs) and converting data refer to the entries for daqCvtTCConvertF and daqCvtTCSetup

Prototypes

C/C++

```
daqCvtTCSetupConvert(DWORD nscan, DWORD cjcPosition, DWORD ntc, TCType
tcType, BOOL bipolar, DWORD avg, PWORD counts, DWORD scans, PSHORT temp,
DWORD ntemp);
```

Visual BASIC

```
VBdaqCvtTCSetupConvert&(ByVal nscan&, ByVal cjcPosition&, ByVal ntc&,
ByVal tcType&, ByVal bipolar&, ByVal avg&, counts%, ByVal scans&, temp%,
ByVal ntemp&)
```

Program References

```
DBK19EX.CPP, DBK52EX.CPP, DBK81EX.CPP, DBK82_83EX.CPP, DBK90EX.CPP
```

This page is intentionally blank.

daqDacSetOutputMode

Format

```
daqDacSetOutputMode(handle, deviceType, chan, outputMode)
```

Purpose

`daqDacSetOutputMode` sets the output mode of the digital-to-analog converter (DAC) or digital pattern output operations for the specified channel.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the DAC waveform output mode will be set
deviceType	DaqDacDeviceType	Specifies the device type
chan	DWORD	Specifies the DAC / digital output channel number
outputMode	DaqDacOutputMode	Defines the DAC waveform output mode to use

Parameter Values

handle: obtained from the `daqOpen` function.

deviceType: see the table in the "Selecting the Output Channel" section below for details

chan: valid values depend on the value of the `deviceType` parameter; see the table in the "Selecting the Output Channel" section below for details

outputMode: see table below

Parameter Type Definitions

OutputMode–(DaqDacOutputMode)	
Definition	**Description**
DdomVoltage	Specifies a single, DC voltage or current output mode.
DdomStaticWave	Specifies static waveform/pattern output mode.
DdomDynamicWave	Specifies dynamic waveform/pattern output mode. (Does Not Apply to DaqBoard/500 Series)
DdomDigitalDirect	Specifies direct mode for P3 digital. (Does Not Apply to DaqBoard/500 Series or /3000 Series)
DdomBipolar	Specifies bipolar output mode. (DaqBoard/500 only)

Returns

DerrNoError   No error

Function Usage

Normally, the `daqDacSetOutputMode` function must be called prior to any output operations on the specified channels. Generally, this function configures the specified port for DC style or waveform output from DACs or pattern output from the 16-bit P3 DIO port.

Selecting the Output Channel

The output channel for which to set the mode is determined by the `chan` and the `deviceType` parameters. The device type is represented by predefined settings described below. The channel is simply an integer (starting at 0) which represents the channel location relative to the first channel of the same type. The table below describes this relationship:

Device Type (deviceType)	Channel (chan)	Description
DddtLocal	0	Set output mode for DAC0 on the main unit device
	1	Set output mode for DAC1 on the main unit device
	2	Set output mode for DAC2 on the main unit device
	3	Set output mode for DAC3 on the main unit device
DddtDbk	$N*4 + 0$	Set output mode for channel0 on a DBK2 or DBK5 on bank N
	$N*4 + 1$	Set output mode for channel1 on a DBK2 or DBK5 on bank N
	$N*4 + 2$	Set output mode for channel2 on a DBK2 or DBK5 on bank N
	$N*4 + 3$	Set output mode for channel3 on a DBK2 or DBK5 on bank N
DddtLocalDigital	0	Set Output Mode for the local P3 16-bit DIO channel on the main unit. This setting only applies when using the local P3 16-bit DIO channel for streamed output.

Setting the Output Mode

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form `daqDac…`. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.

The `outputMode` parameter indicates the type of output update to be performed on the specified DAC/Digital output channel. Generally, the output mode is either direct, asynchronous update or a waveform/pattern output.

There are two basic types of waveform/pattern output. One is a static mode that allows downloading the entire waveform/pattern output buffer to the internal FIFO on the device for unattended output. The other is a dynamic mode that allows continuous update of the output by the application. The settings for all of the possible modes are as follows:

`DdomVoltage`– Specifies a single, DC voltage or current output mode. This mode defines the output of the specified DAC channel to be updated only when written to explicitly. The valid range over which actual voltage and current values can be written to the port depends upon the specified range of the local DAC device or the DBK2 or DBK5 (see output range specifications for your device). When this mode is set with the `daqDacSetOutputMode` function, no change to the current output state of the channel will be performed. See `daqDacWt` and `daqDacWtMany` for the actual writing of the DAC channel values. No waveform/pattern outputs can be generated for the specified channel while the channel is in the `DdomVoltage` mode. This setting represents the default setting for all channels.

`DdomStaticWave-` Specifies static waveform/pattern output mode. This mode allows the generation of a non-streamed waveform output to the specified DAC/Digital Output channel. In this mode, the aggregate size of the waveform/pattern output buffer must be less than or equal to the size of the internal waveform/pattern output FIFO in the device. This allows the entire waveform/pattern output buffer to be loaded into the device's internal output FIFO. Once the sample updates have been transferred (or downloaded) to the device, the device is responsible for outputting the data. No other further sample update management needs to be performed by the application (other than monitoring the progress of the waveform/pattern output). While the size is limited and no changes to the waveform can be made once the output is started, this mode has the advantage of not having to periodically feed output data (through the program) to the device for the waveform/pattern output to continue.

Device	Output FIFO Size (Static Waveform/Pattern Output)
DaqBoard (ISA)	4,096 total update samples
DaqBoard/500	128,000 total update samples (allocated in PC memory)
DaqBoard/1000 Series	128,000 total update samples (allocated in PC memory)
DaqBoard/2000 Series Devices which support DAC	512,000 total update samples (allocated in PC memory)
DaqBook/2000 Series DaqLab/2000 Series DaqScan/2000 Series	128,000 total update samples (internal FIFO)
DaqBoard/3000 Series	512,000 total update samples (allocated in PC memory)

The aggregate waveform/pattern output buffer size is equal to (the number of sample updates for each channel) * (the number of channels configured for static waveform mode).

`DdomDynamicWave-` Specifies dynamic waveform/pattern output mode. This mode allows continual, dynamic updating of the waveform/pattern output. Dynamic waveform/pattern output generation is not size dependent, and waveform/pattern output updating can be performed indefinitely. Actual waveform/pattern output generation updating is performed by continually feeding waveform/pattern data to the device using the `daqDacWaveSetBuffer` and `daqDacTransferStart` routines to continually fill the device's waveform/pattern output FIFO. The waveform transfer operation to the waveform/pattern output FIFO of the device can be halted at any time with `daqDacTransferStop`; Stopping the transfer to the waveform/pattern output FIFO of the device, however, will not stop the device from outputting what remains in its FIFO. To stop the device from outputting samples from its internal waveform/pattern output FIFO, the `daqAdcDisarm` function must be used.

The `DdomDynamicWave` mode is not valid forDaqBoard/500 Series boards.

Using Static and Dynamic Output Modes

The `DdomStaticWave` and `DdomDynamicWave` output modes allow the configuration of one or more channels for waveform/pattern output from DAC or digital input/output (DIO) channels. However, the `DdomStaticWave` and `DdomDynamicWave` output modes have special considerations when they are used.

Both modes allow waveform/pattern output to any of the available DAC or DIO channels. However, when using these modes, each channel must be configured to use the same mode--either `DdomStaticWave` or `DdomDynamicWave`. So, it is not possible to mix `DdomStaticWave` and `DdomStaticWave` configurations on the same device for a given waveform/pattern output. There is no problem with configuring one or more channels as direct (`DdomVoltage`) output along with other channels that are configured for waveform/pattern output.

When configuring the update transfer buffer via `daqDacWaveSetBuffer` in static mode, the update transfer buffer stores the update samples in a seperate buffer for each channel. However, in dynamic mode the update transfer buffer stores the update samples for all the channels configured for waveform/pattern output, not just for the channel specified. Since the buffer is shared, all waveform/pattern output channels must have the same number of sample updates. This means that the transfer buffer size depends upon the total number of channels configured for waveform/pattern output and the total number of updates for each channel. See `daqDacWaveSetBuffer` for more details.

The update clock for all waveform/pattern output channels must be the same source. All DAC and P3 16-bit Digital channels must use the same update clock for updating their outputs. See the entry for `daqDacWaveSetClockSource` for more details.

The DaqBoard/2000 Series devices allow digital output data to be streamed to the P3 16-bit DIO port for pattern output. This digital output data can be streamed concurrently with waveform output to the DAC channels specified using the same output update clock. Alternatively, the streamed digital output can be streamed exclusively to the P3 16-bit DIO output channel without generating waveform output to any of the DAC channels. However, P3 Digital Pattern Output and DAC waveform output cannot be performed concurrently with *different* update clocks.

Prototypes

C/C++

```
daqDacSetOutputMode(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan, DaqDacOutputMode outputMode);
```

Visual BASIC

```
VBdaqDacSetOutputMode&(ByVal handle&, ByVal deviceType&, ByVal chan&,
ByVal outputMode&)
```

Program References

```
DAQDACEX02.CPP, DAQDACEX03.CPP, DBK02EX.CPP, DBK05.CPP, DAQEX.FRM (VB)
```

daqDacTransferGetStat

Format

```
daqDacTransferGetStat(handle, deviceType, chan, active, retCount)
```

Purpose

daqDacTransferGetStat returns the current status and the total transfer count of the current waveform/pattern output channel.

> **DAC output mode must be set to `DdomDynamicWave` for this function to be called. See the `daqDacSetOutputMode` function for details.**

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle of the device from which to retrieve current waveform/pattern output transfer status
deviceType	DaqDacDeviceType	Specifies the DAC / digital output type
chan	DWORD	Specifies the DAC / digital output channel
active	PDWORD	Indicates the current state of the acquisition and transfer in the form of a bit mask
retCount	PDWORD	Total number of waveform/pattern output samples transferred for the current waveform/pattern output transfer

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the value chosen for the deviceType parameter; see deviceType table below

active: see table below

retCount: the total number of updates can range from 0 to 4,294,967,295 updates; however, memory limitations may apply

Parameter Type Definitions

deviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Retrieve the status of a waveform/pattern output transfer where DAC0 is one of the channels being output.
	1	Retrieve the status of a waveform/pattern output transfer where DAC1 is one of the channels being output.
	2	Retrieve the status of a waveform/pattern output transfer where DAC2 is one of the channels being output.
	3	Retrieve the status of a waveform/pattern output transfer where DAC3 is one of the channels being output.
DddtLocalDigital	0	Retrieve the status of a waveform/pattern output transfer where the 16-bit P3 Digital port is one of the channels being output.

active	
Definition	**Description**
DdafWaveformActive	A waveform/pattern output is active. The trigger may or may not yet have occurred, but the waveform/pattern output has at least been armed.
DdafWaveformTriggered	The waveform/pattern output has been triggered. Data is now being streamed from the specified channels of the devices DAC channel(s) and/or P3 16-bit digital port.
DdafTransferActive	A transfer from PC-based buffers/files to the output FIFO on the device is currently active.

Returns

DerrNoError   No error

Function Usage

The daqDacTransferGetStat function will get the current status of a dynamic waveform/pattern output transfer for the specified DAC or digital output channel.

> **The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form `daqDac…`. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.**

Waveform/Pattern Output Progress

In the subsequent discussion, these three terms are used as follows:

- A *sample* refers to the data for a single output channel.
- An *update* refers to data for all channels that are configured for waveform/pattern output. An update consists of all data required to update all channels during a single output clock pulse.
- The *write position* of the buffer described below is a pointer to the next update block to be written to the internal FIFO on the device. If using a circular buffer, then the data that has already been written but not yet updated by the application may then be updated by the application. When using a circular buffer, it is left to the application to maintain the pointer(s) to the data that it has updated.

The retCount parameter can be defined in two manners, depending on the output mode. If the output is set to dynamic mode with the DdwmNShot or DdwmInfinite values set (using the daqDacWaveSetMode function), retCount is equal to the number of updates read in from the dynamic buffer. However, if either the output is set to either static mode, or dynamic mode with the DdwmNFileIterations value set (using the daqDacWaveSetMode function), retCount is equal to the number of DAC updates since triggered. See the daqDacWaveSetBuffer function on for more information on buffer allocation modes.

The current write position within the buffer (measured in updates) is equal to

$$retCount \bmod scanCount$$

where *mod* is defined as the integer remainder of dividing retCount by scanCount (as defined by the daqDacWaveSetBuffer function). Since all channels in the waveform/pattern output are updated concurrently, this number represents the number of updates performed for all channels.

The current write position within the buffer (measured in total samples) is equal to

$$(retCount \bmod scanCount) * (\text{total output channels})$$

where the value of (total output channels) is the total number of channels that are configured for waveform/pattern output (channels whose output mode is set to DdomStaticWave or DdomDynamicWave). Since each sample is a 2-byte word, this number can be multiplied by 2 to get the current write position within the buffer in bytes.

Putting it all together

The following table shows a number of different scenarios (each assumes a total scan count of 1,000,000 output updates). These scenarios are representative of some typical waveform/pattern output states but do not necessarily represent all of the possible states.

States (`active`)	Progress (`retCount`)	Description

`DdafWaveformActive +`
`DdafWaveformTriggered +`
`DdafTransferActive`
	210,000	The waveform/pattern output is active and has been triggered and is currently outputting updates to the DAC/Digital Output ports specified. A transfer to buffer is also active and a total of 210,000 updates have been output so far.

`DdafWaveformActive +` `DdafTransferActive`	0	The waveform/pattern output has been armed but has not yet been triggered. A transfer to buffer is also active but no updates have been output so far because the trigger event has not yet occurred.
`DdafTransferActive`	0	A waveform/pattern output transfer has been configured and started (`daqAdcTransferSetBuffer`, `daqDacTransferStart`) but the acquisition has not yet been armed. *The acquisition needs to be armed using* `daqDacWaveArm`.
`(0)`	1,000,000	The waveform/pattern output has triggered and has completed. The transfer is not active and 1,000,000 updates have been performed. So the both the transfer and the waveform/pattern output from the device are complete.

Prototypes

C/C++

```
daqDacTransferGetStat(DaqHandleT handle, DaqDacDeviceType deviceType,
DWORD chan, PDWORD active, PDWORD retCount);
```

Visual BASIC

```
VBdaqDacTransferGetStat&(ByVal handle&, ByVal deviceType&, ByVal chan&,
active&, retCount&)
```

Program References

```
None
```

daqDacTransferStart

Format

```
daqDacTransferStart(handle, deviceType, chan)
```

Purpose

`daqDacTransferStart` initiates a dynamic waveform/pattern output transfer to the output FIFO on the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which a waveform transfer is to be initiated
deviceType	DaqDacDeviceType	Specifies the type
chan	DWORD	Specifies the channel

Parameter Values

handle: obtained from the `daqOpen` function

deviceType: see table below

chan: valid values depend on the values chosen for the `deviceType` parameter; see the `deviceType` table below

Parameter Type Definitions

DeviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form `daqDac…`. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.

The waveform/pattern output transfer will be performed from the waveform/pattern output buffer configured using the `daqDacWaveSetBuffer` function to the device's internal output FIFO. This transfer will continue until:

- The entire waveform/pattern output buffer has been transferred to the internal output FIFO on the device.
- The transfer is halted (with the `daqDacTransferStop` function).
- The waveform/pattern output is disarmed (with the `daqDacWaveDisarm` function).

The waveform output can be specified for any DAC channel or the 16-bit DIO channel located on the P3 port, However, the transfer is initiated for all channels configured for dynamic waveform/pattern output mode (`DdomDynamicWave`). The `daqDacSetOutputMode` function should be used to set the output mode to `DdomDynamicWave` prior to calling this function.

This function should be used only with waveform/pattern output modes.

Prototypes

C/C++

```
daqDacTransferStart(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan);
```

Visual BASIC

```
VBdaqDacTransferStart&(ByVal handle&, ByVal deviceType&, ByVal chan&)
```

Program References

```
None
```

daqDacTransferStop

Format

```
daqDacTransferStop(handle, deviceType, chan)
```

Purpose

daqDacTransferStop stops a dynamic waveform/pattern output transfer for the specified DAC or digital output channel if one is currently active.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the current DAC waveform transfer will be stopped
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the value chosen for the deviceType parameter; see the deviceType table below

Parameter Type Definitions

DeviceType–(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Stop the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Stop the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Stop the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Stop the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Stop the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form `daqDac`.... This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.

The `daqDacTransferStop` function will terminate the transfer of waveform/pattern output data; however, it will not halt the waveform output on the specified channel. Output data already sent to the devices waveform/pattern output FIFO will continue to be output until there is no more data in the FIFO. The transfer may be re-initiated for the same waveform/pattern output transfer buffer or another buffer by again calling the `daqDacTransferStart` function. To terminate the waveform output as well as the transfer, refer to the `daqDacWaveDisarm` function. The output mode must be set to `DdomDynamicWave` via the `daqDacSetOutputMode` function prior to calling this function.

This function should be used only with waveform/pattern output modes.

Prototypes

C/C++

```
daqDacTransferStop(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan);
```

Visual BASIC

```
VBdaqDacTransferStop&(ByVal handle&, ByVal deviceType&, ByVal chan&)
```

Program References

```
None
```

daqDacWaveArm

Format

```
daqDacWaveArm(handle, deviceType)
```

Purpose

`daqDacWaveArm` arms a waveform/pattern output for all channels configured for waveform/pattern output modes for the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which a DAC waveform output will be armed
deviceType	DaqDacDeviceType	Specifies the DAC type

Parameter Values

handle: obtained from the `daqOpen` function
deviceType: see table below

Parameter Type Definitions

deviceType-(DaqDacDeviceType)	
Definition	**Description**
DddtLocal	DAC output from P1 for ISA type DaqBoards; DAC output from P3.
DddtLocalDigital	DAC output from the 16-bit P3 digital port.

Returns

DerrNoError   No error

Function Usage

> **The term** *waveform/pattern output* **is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.**

The `daqDacWaveArm` function enables a waveform/pattern output based upon the current waveform/pattern output channel configuration. Channels configured for waveform/pattern output need to be set either `DdomStaticWave` or `DdomDynamicWave` output mode to be included in the waveform/pattern output operation
(see `daqDacSetOutputMode` for more details on configuration channel output modes).

Before Invoking `daqDacWaveArm`

This function requires that all waveform channels are properly set to the correct modes and that the waveform/pattern output trigger events, the output modes, and the update clock are all set before this function is invoked. Specifically, the following steps need to be performed before calling this function:

- Configure all the output channels to be included in the waveform/pattern output as static or dynamic output mode (`DdomStaticWave` or `DdomDynamicWave`) via the `daqDacSetOutputMode` function.

- If streaming output from a file, then set the disk file using the `daqDacWaveSetDiskFile` for each channel that as been configured for waveform/pattern output (the disk file should name should be the same for all output channels).

- Set the waveform/pattern output clock source for each output channel via the `daqDacWaveSetClockSource` function.

- If the selected clock source uses the internal waveform/pattern output pacer clock, then set the output update frequency using the `daqDacWaveSetFreq` function.

- Set the trigger event which will initiate the waveform/pattern output via the `daqDacWaveSetTrig` function.

- Configure the update mode and waveform/pattern output termination conditions using the `daqDacWaveSetMode` function.

- Though it is not required, it is also recommended that the transfer buffer is allocated and the transfer is enabled prior to invoking this function (`daqDacWaveSetBuffer` and the `daqDacTransferStart` functions.

After Invoking `daqDacWaveArm`

Once the preliminary steps have been taken to setup up `daqDacWaveArm`, it can be used to arm the device for waveform/pattern output. However, the device will not begin the actual output of data to the ports configured for waveform/pattern output until the trigger event has occurred (unless the trigger event is configured as `DdtsImmediate`). The trigger event is configured using the `daqDacWaveSetTrig` function.

Once triggered, the device will begin outputting data from its internal output FIFO. Therefore, it is important to have already transferred at least the first block of data from the application allocated output transfer buffer to the device's internal output buffer before the trigger event occurs. For more information on transferring data from the application to the internal output FIFO, see `daqDacWaveSetBuffer` and `daqDacTransferStart`. While the waveform/pattern output operation is occurring, the progress of the waveform/pattern output may be monitored using the `daqDacTransferStart` function.

The waveform/pattern output will terminate based upon the update mode set by the `daqDacWaveSetMode` function. Regardless of the update mode, however, any waveform/pattern output may be terminated by issuing the `daqDacWaveDisarm` function for the device.

When this function is invoked, the driver determines if there are any problems with any of the waveform/pattern output configuration parameters issued by application. Since the bulk of the actual waveform/pattern output configuration is performed when this function is invoked, any potential configuration problems or parameter value conflicts will be detected here.

Prototypes

C/C++

```
daqDacWaveArm(DaqHandleT handle, DaqDacDeviceType deviceType);
```

Visual BASIC

```
VBdaqDacWaveArm&(ByVal handle&, ByVal deviceType&)
```

Program References

```
DAQDACEX02.CPP, DAQDACEX03.CPP
```

daqDacWaveDisarm

Format

```
daqDacWaveDisarm(handle, deviceType)
```

Purpose

daqDacWaveDisarm disarms a waveform/pattern output if one is active on the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which a DAC waveform output will be disarmed
deviceType	DaqDacDeviceType	Specifies the DAC type

Parameter Values

handle: obtained from the daqOpen function
deviceType: see table below

Parameter Type Definitions

deviceType-(DaqDacDeviceType)	
Definition	**Description**
DddtLocal	DAC output from P2 of the device
DddtLocalDigital	DAC output from the 16-bit P3 digital port

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac.… This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.

The daqDacWaveDisarm function will disable the waveform/pattern output on the specified device and terminate any output buffer transfers that are currently active (see daqDacTransferStop for more details on output buffer transfer termination). Waveform/pattern output will be terminated immediately, regardless of the current state of the waveform/pattern output or the state of the digital-to-analog converter (DAC) or digital input/output (DIO) channels from which the waveform/pattern output is being generated.

Prototypes

C/C++

```
daqDacWaveDisarm(DaqHandleT handle, DaqDacDeviceType deviceType);
```

Visual BASIC

```
VBdaqDacWaveDisarm&(ByVal handle&, ByVal deviceType&)
```

Program References

DAQDACEX02.CPP, DAQDACEX03.CPP

---

daqDacWaveGetFreq

Also See: `daqDacWaveArm,`
`daqDacWaveDisarm, daqDacWaveSetFreq`

Format

```
daqDacWaveGetFreq(handle, deviceType, chan, freq)
```

Purpose

`daqDacWaveGetFreq` retrieves the current setting for the waveform/pattern output update frequency (in Hz) for the specified device (selected by `daqDacWaveSetClockSource`).

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which to retrieve the current waveform output frequency
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
freq	PFLOAT	Returns the current DAC waveform output frequency setting

Parameter Values

`handle:` obtained from the `daqOpen` function

`deviceType:` see table below

`chan:` valid values depend on the values chosen for the `deviceType` parameter; see the `deviceType` table below

`freq:` pointer to a single precision floating point value (4 bytes) that will store the update frequency update upon return from this function.

Parameter Type Definitions

DeviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.

The daqDacWaveGetFreq function returns the frequency via the freq parameter, The frequency is dependent upon the clock source chosen for the selected device (see daqDacWaveSetClockSource. The freq parameter is a pointer to a single-precision floating point variable that must be allocated by the calling application. The waveform/pattern output update frequency is programmed with the daqDacWaveSetFreq function.

The returned frequency value (freq) will not be valid until the waveform is armed with the daqDacWaveArm function.

Getting the Actual Output Update Frequency

The frequency programmed using the daqDacWaveSetFreq function may not be obtainable. The reason for this is that either the frequency specified was outside of the operating update frequency of the output pacer clock, or the frequency is not attainable due to the resolution of the pacer clock. If the latter is the case, then the output pacer clock will be programmed to the nearest attainable frequency. If the frequency is outside of the operating range of the pacer clock, then an error will be generated. See the daqDacWaveGetFreq function for more information on the actual setting of the output pacer clock frequency.

The specifications section of the device user's manual provides the frequency settings for the device.

Prototypes

C/C++

```
daqDacWaveGetFreq(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan, PFLOAT freq);
```

Visual BASIC

```
VBdaqDacWaveGetFreq&(ByVal handle&, ByVal deviceType&, ByVal chan&, freq!)
```

Program References

None

daqDacWaveSetBuffer

Format

```
daqDacWaveSetBuffer(handle, deviceType, chan, buf, scanCount,
transferMask)
```

Purpose

daqDacWaveSetBuffer configures a waveform/pattern output transfer buffer for the specified device and channel.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which a DAC waveform transfer buffer will be configured
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
buf	PWORD	Pointer to the user allocated waveform transfer buffer
scanCount	DWORD	Length of the waveform buffer in output samples
transferMask	DWORD	Configures the buffer transfer mode

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table below

buf: pointer to buffer which contains scanCount updates for each channel in waveform/pattern output mode

scanCount: length of the buffer in updates for each channel can be 1 to 4,294,967,295; however, memory limitations may apply

transferMask: see table below

Parameter Type Definitions

DeviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

transferMask	
Definition	**Description**
DdtmUserBuffer	Selects a user buffer for waveform/pattern output
DdtmDriverBuffer	Selects a driver buffer for waveform/pattern output

Returns

```
DerrNoError        No error
```

Function Usage

> **The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.**

The `daqDacWaveSetBuffer` function allows you to configure a waveform/pattern output transfer buffer for the specified device and channel. This function may be used to configure a user-supplied buffer for transferring user supplied data to any of the output channels capable of performing waveform or streamed output.

Buffer Location, Length, and Settings

The `buf` parameter is the address of the waveform/pattern output transfer buffer allocated by the application. If the application is supplying the buffer, then this value must be an address to an adequately allocated buffer

The `scanCount` parameter is the total length of the output transfer buffer in updates per channel. The number of channels configured for waveform/pattern output determines the total number of samples required for each update. Therefore the buffer size to be allocated (in bytes):

> `scanCount` * (number of output channels configured for waveform/pattern output)
> * (the sample size: 2 bytes)

Buffer Management

Buffer management for waveform/pattern output depends upon the output mode setting, described as follows:

Static Waveform/Pattern Mode

If the waveform/pattern output mode has been defined to be static (see `daqDacSetOutputMode`, then the following steps must be performed to complete the static waveform/pattern output operation:

- Specify the output transfer buffer location and details with the `daqDacWaveSetBuffer` function. The specified `buf` parameter must point to memory which has already been allocated by the application prior to calling this function. The allocated buffer must be large enough to hold all of the output updates required for each channel to complete the entire static waveform/pattern output, as determined by the current waveform/pattern output channel configuration. The output data must fit entirely within the internal waveform/pattern output FIFO on the device.
- Configure any waveform/pattern settings (such as update mode, trigger sources, frequency settings that need to be configured.
- Call the `daqDacTransferStart` function to download the output transfer buffer into the internal waveform/pattern output FIFO on the device.
- Arm the waveform/pattern output by issuing the `daqDacWaveSetTrig` function.
- Trigger the waveform/pattern output (if necessary).
- The status of the static waveform/pattern output can be monitored by calling the `daqDacTransferGetStat` function.

Dynamic Waveform/Pattern Mode with Circular Buffer

If the waveform/pattern output mode has been defined to be dynamic (see `daqDacSetOutputMode`) and a circular output transfer buffer is to be used, then the following will need to be performed to complete the dynamic waveform/pattern output operation:

- Specify the output transfer buffer location and details with the `daqDacWaveSetBuffer` function. The specified `buf` parameter must point to memory which has already been allocated by the application prior to calling this function. The allocated buffer must be large enough to hold the number of output updates for each channel required for the block of data to be transferred as determined by the current waveform/pattern output channel configuration.

- Configure any waveform/pattern settings (such as update mode, trigger sources, frequency settings that need to be configured

- Call the `daqDacTransferStart` function to download the output transfer buffer into the internal waveform/pattern output FIFO on the device.

- Arm the waveform/pattern output by issuing the `daqDacWaveSetTrig` function.

- Trigger the waveform/pattern output (if necessary).

- Monitor status of the dynamic waveform/pattern output calling the `daqDacTransferGetStat` function. The total amount of data transferred from the circular output transfer buffer to the waveform/pattern output FIFO on the device will be returned in the `retCount` parameter. The current position of the next data block to be written to the waveform/pattern output FIFO on the device can be calculated using the value of the `retCount` parameter. Please refer to the `daqDacTransferGetStat` function for a complete discussion of calculating the current write position pointer.

In general, the size allocated should be at least large enough to handle any delays that may be encountered between buffer updates by the application without the buffer running completely empty. As the waveform/pattern output progresses, the waveform/pattern output FIFO on the device will be filled by emptying the unread data in the output transfer buffer. This value can vary, depending upon the ability of the application to frequently update the buffer, and the update rate at which the outputs are clocked. However, a good general rule is to make the size of the output transfer buffer at least that of the size of the waveform/pattern output FIFO on the device.The filling of the output FIFO with data in the output transfer buffer is performed automatically by the driver without checking whether the application has updated the buffer. Therefore, the application needs to be aware of the current write position within the buffer, as well as feeding new data updates to the buffer portions which have already been written but not updated. This function should be called with the `DdtmCycleOn` flag set.

Dynamic Output Transfer Buffer Organization

The output transfer buffer is organized into sequences of update data blocks. Each update data block represent the sample data necessary to update all output channels configured for waveform/pattern output. The sample data is ordered according to the output channel for which it will be output. Each update data block is ordered as follows:

DAC0	DAC1	DAC2	DAC3	P3 DO

If any of the above channels is not configured for waveform/pattern output, then it will simply not be included in the update data block--however, the channel ordering will not change. If, for example, a waveform/pattern output is configured for DAC0, DAC2 and P3 DO will have the following update data block:

DAC0	DAC2	P3 DO

The transfer output buffer is organized into update data blocks. When the output update clock fires, the next available update data block is loaded from the internal waveform/pattern output FIFO on the device to the port, which then presents the values to either the DAC or digital port for output.

Data Presented to Outputs									
DAC0	DAC1	DAC2	DAC3	P3D0	DAC0	DAC1	DAC2	DAC3	P3D0

Update clock   Update clock

◄——————————————— Processing Output FIFO ———————————————►
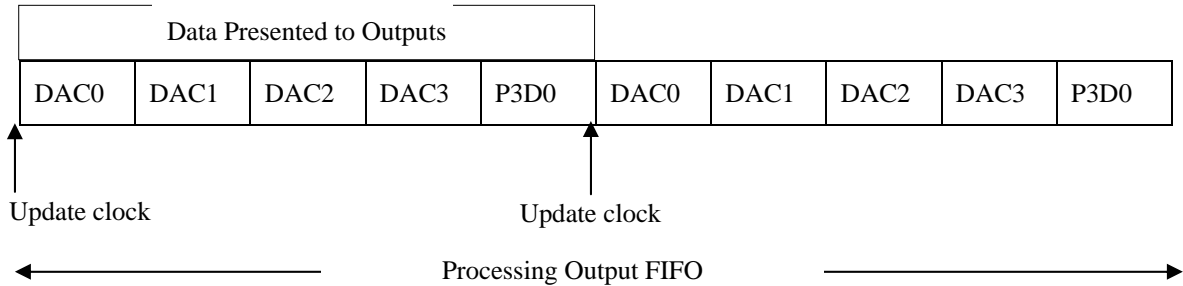
Likewise, the buffer is organized in a similar manner.

DAC0	DAC1	DAC2	DAC3	P3D0	DAC0	DAC1	DAC2	DAC3	P3D0	…
Update Data Block 0					Update Data Block 1					
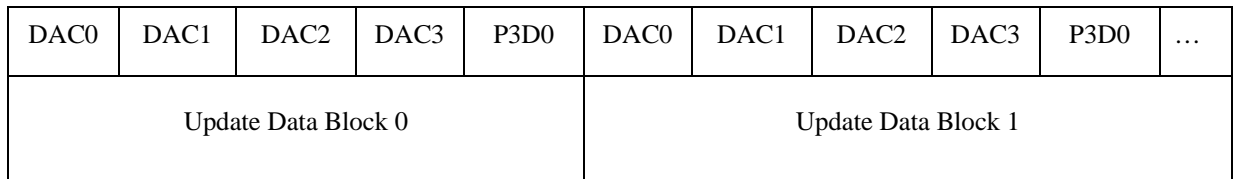
As the output FIFO empties, the driver will automatically transfer data from the output transfer buffer to the output FIFO on the device. This transfer process will continue until either the waveform/pattern output transfer is stopped via the `daqDacTransferStop` function, the waveform/pattern output is halted with the `daqDacWaveDisarm` function or the waveform/pattern output is normally terminated as defined by the `daqDacWaveSetMode` function.

Buffer Output Channel Data Format

The format for each update sample in the buffer is a 16-bit integer. This parameter is an integer value ranging from 0 to 65,535. This is a 16-bit value and the 16-bit value spans the range over which the output can vary, depending upon the specified range of the port for the device.

Some DAC channels use 16-bit D/A converters--in these converters, each bit in the 16-bit integer corresponds to one bit programmed to the D/A converter. However, some DAC channels have 12-bit D/A converters. In this case, the low order nibble (4-bits) of the 16-bit sample will be ignored.

The specifications section of the device user's manual provides the ranges and the associated resolution values for the device.

Prototypes

C/C++

```
daqDacWaveSetBuffer(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan, PWORD buf, DWORD scanCount, DWORD transferMask);
```

Visual BASIC

```
VBdaqDacWaveSetBuffer&(ByVal handle&, ByVal deviceType&, ByVal chan&,
buf%(), ByVal scanCount&, ByVal transferMask&)
```

Program References

```
DAQDACEX02.CPP
```

daqDacWaveSetClockSource

Also See: **daqDacWaveSetFreq, daqDacWaveGetFreq**

Format

daqDacWaveSetClockSource(handle, deviceType, chan, clockSource)

Purpose

daqDacWaveSetClockSource sets the clock source for waveform/pattern output to the digital-to-analog converter channels (DAC) and the digital input/output (DIO) channels.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the waveform output clock source will be set
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
clockSource	DaqDacClockSource	Set the clock to the specified source

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table below

clockSource: see table below

Parameter Type Definitions

deviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

clockSource-(DaqDacClockSource)

Note: For daq9513 specific commands [used only for DaqBoard and DaqBooks of the 100 and 200 Series] refer to Appendix D.

Definition	Description
DdcsDacClock	Specifies using the internal waveform/pattern output pacer clock on the device (see **daqDacWaveSetFreq**).
DdcsGatedDacClock	Specifies using the internal waveform/pattern output clock which is then gated through an external input (TTL input). Refer to pinout in user's manual for connection location.
DdcsAdcClock	Specifies that the waveform/pattern output clock follow that of current acquisition clock setting (see daqAdcSetClockSource).
DdcsExternalTTL	Specifies an external timebase supplied via External Dac Pacer Clock Input. Refer to pinout in user's manual for connection location.
DdcsRisingEdge	Clock Control Flag DdcsRisingEdge = 0x0
DdcsFallingEdge	Clock Control Flag DdcsFallingEdge = 0x100
DdcsOutputDisable	Output Control Flag Disables the Dac internal clock output (see note 1).
DdcsOutputEnable	Output Control Flag Enables the Dac internal clock output (see note 1).

Note 1: This note applies to the Daq Devices listed in the first block of the above table.

To enable the pacer output you must include the clock source with a parameter that enables the output clock; in other words, you have to write your **daqDacWaveSetClockSource** command as follows:

C/C++ Style:

```
daqDacWaveSetClockSource (handle, deviceType, chan,
    DdcsDacClock | DdcsOutputEnable);
```

Visual Basic Style:

```
VBdaqDacWaveSetClockSource (handle, deviceType, chan,
    DdcsDacClock + DdcsOutputEnable);
```

The **DdcsOutputEnable** parameter is defined in the header file in the Daqx.bas module (VB):

' DaqBoard/2000 Output Control Flags

```
Global Const DdcsOutputDisable = 0
            Disables the Dac internal clock output.
Global Const DdcsOutputEnable = &H1000
            Enables the Dac internal clock output.
```

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.

The daqDacWaveSetClockSource function's clockSource parameter specifies the clock source to use for updating all DAC/DIO channels which have been configured for waveform/pattern output. Regardless of the clock source selected, all channels configured for waveform/pattern output (see daqDacSetOutputMode will be synchronously updated upon each update clock pulse.

If using DaqBoard(ISA): If the more than one DAC channel waveform/pattern output is active, the update frequency for each channel is this rate divided by the total number of active DAC waveform output channels.

Prototypes

C/C++

```
daqDacWaveSetClockSource(DaqHandleT handle, DaqDacDeviceType deviceType,
DWORD chan, DaqDacClockSource clockSource);
```

Visual BASIC

```
VBdaqDacWaveSetClockSource&(ByVal handle&, ByVal deviceType&, ByVal chan&,
ByVal clockSource&)
```

Program References

DAQDACEX02.CPP, DAQDACEX03.CPP

daqDacWaveSetDiskFile

Also See: **daqDacWaveSetBuffer,
daqDacSetOutputMode, daqDacTransferGetStat**

Format

daqDacWaveSetDiskFile(handle, deviceType, chan, filename, numUpdateCycles,
offsetBytes, offsetUpdateCycles, dataFormat)

Purpose

daqDacWaveSetDiskFile configures a waveform/pattern output for streaming from a file.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
fileName	LPSTR	String representing the path and filename of the disk file to be output
numUpdateCycles	DWORD	Number of update cycles to read from file
offsetBytes	DWORD	Offset for header in bytes
offsetUpdateCycles	DWORD	Number of update cycles to skip at the start of the file
dataFormat	DaqDacWaveFileDataFormat	Format of data file

Parameter Values

hnadle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType
table below

fileName: pointer to a valid string containing the path and file name of the output data file

numUpdateCycles: the number of usable updates cycles in the file can range from 0 to 4,294,967,295;
however, disk memory limitations may apply. If set to 0, will read all cylces from the
file

offsetBytes: the number of bytes to skip at the beginning of the file range from 0 to 4,294,967,295;
however, disk memory limitations may apply

offsetUpdateCycles: the number of update cycles to skip at the beginning of the file range from 0 to
4,294,967,295, however, disk memory limitations may apply

dataFormat: see table below

Parameter Type Definitions

deviceType-(DaqDacDeviceType)		
Definition	**Chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

dataFormat-(DaqDacWaveFileDataFormat)				
dataFormat Value	**Data Format**	**DAC Output**	**P3 Digital Output**	**Sample Data Termination**
DdwdfBinaryCounts	16-bit word (low/high byte)	Raw D/A Counts (low/high byte)	Word is 16-bit Pattern	None -- fixed 2 bytes
DdwdfBinaryCountsHL	16-bit word (high/low byte)	Raw D/A Counts (high/low byte)	Word is 16-bit Pattern	None – fixed 2 bytes
DdwdfBinaryFloat	Single Precision Float (4 bytes)	Float D/A Volts (-10.0 to +10.0)	Integer portion is 16-bit Pattern	None – fixed 4 bytes
DdwdfBinaryDouble	Double Precision Float (8-bytes)	Double D/A volts (-10.0 to +10.0)	Integer portion is 16-bit Pattern	None – fixed 8 bytes
DdwdfAsciiCountsDec	Decimal ASCII chars (0-65535)	ASCII Decimal D/A Counts	Decimal value is 16-bit Pattern	Any non-Dec ASCII chars
DdwdfAsciiCountsHex	Hex ASCII chars (0-FFFF)	ASCII Hex D/A Counts	Hex value is 16-bit Pattern	Any non-Hex ASCII chars
DdwdfAsciiCountsBin	Binary ASCII chars (0 – 1111111111111111)	ASCII Binary D/A Counts	Decimal value is 16-bit Pattern	Any non-Bin ASCII chars
DdwdfAsciiCountsOct	Octal ASCII chars (0 – 177777)	ASCII Octal D/A Counts	Decimal value is 16-bit Pattern	Any non-Octal ASCII chars
DdwdfAsciiFloat	Floating Point ASCII chars (-10.0 – 65535.0)	ASCII Float Voltage (-10.0 - +10.0)	Float value (integer portion) is 16-bit Pattern	Any non-Floating Point ASCII chars

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* **is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to digital-to-analog converter (DAC) channels and/or digital input/output (DIO) channels that are synchronously updated by a clock source defined by the application.**

The `daqDacWaveSetDiskFile` function has two prerequisites: first, the appropriate channels have been configured for waveform/pattern output using dynamic output mode (see the `daqDacSetOutputMode` function) and second, the `daqDacWaveSetBuffer` function must be set to `DdtmDriverBuffer`. Generally, the waveform/pattern should be configured in the same manner as a standard transfer from a PC memory-based buffer, with the exception that no transfer buffer needs to be allocated. The driver will automatically transfer all data from the specified file to the internal output FIFO on the device while the output transfer is active.

File Location

The `filename` parameter specifies the location of the waveform/pattern output file. This is a string variable that contains the path and file name to open. The path may be any valid local or network path name. However, it should be noted, that a path to a network file may have intrinsic file I/O delays associated with it that would hinder the output process under high throughput configurations. Care should be exercised when specifying a file that is not local to the PC controlling the waveform/pattern output operation.

File Organization

The following sections describe the layout and format of the source file for the waveform/pattern output operation from file. This function supports a number of different data formats for the source file, as well as providing mechanisms to output any contiguous block of update output data within the file.

File Layout

The `offsetBytes` and the `offsetUpdateCycles` parameters allow the start of the update data to be offset from the beginning of the file. The `offsetBytes` parameter indicates that a certain number of bytes appear at the beginning of the file that should be ignored. These bytes may be file header information or other data but are not valid output samples. If there is no file header information or non-output data, then `offsetBytes` should be set to 0. The `offsetUpdateCycles` parameter indicates that a certain amount of output update cycle data should be ignored. The value of this parameter specifies the number of output cycles that should be ignored at the beginning of the file. If the `offsetBytes` is non-zero, then these output update cycles may follow the header or other information in the file. An output cycle consists of an update data block which consists of all channel data necessary for one update clock output for all configured channels. The `numUpdateCycles` parameter indicates the number of output update cycles to output from the file *after* the offset data. This value does not indicate the number of total output cycles to perform during the output from file operation, it simply indicates the number of cycles in the file that should be output for each iteration of the file.

The following table shows how the entire file is organized. The arrows indicate the iterations of the valid output cycle data within the file when the file is being output to the device.

- In the following table, the shaded areas indicate the portion of the file that is to be ignored for the waveform/pattern output from file operation.

- If the **numUpdateCycles** value is set to 0, then all the data *after* the offset data and before the end of the file will be used during the output operation.

- The following table represents a file that has sample data for DAC0, DAC1, DAC2, DAC3 and P3 Digital Output channels. A file with data for a different channel configuration will change accordingly. The number of cycles to ignore will be based upon the waveform/pattern output channels that are currently configured(see **daqDacSetOutputMode** function).

Header Information (`offsetBytes` in bytes)					
Cycle 0	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle 1	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle 2	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle 3	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle ...	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (`offsetUpdateCycles-1`)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (`offsetUpdateCycles`)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (`offsetUpdateCycles+1`)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle ...	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle ...	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle ...	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle ...	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (`offsetUpdateCycles + NumUpdateCycles - 1`)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (`offsetUpdateCycles + NumUpdateCycles`)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (`offsetUpdateCycles + NumUpdateCycles + 1`)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle ...	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data
Cycle (EOF cycle)	DAC0 Data	DAC1 Data	DAC2 Data	DAC3 Data	P3 D0 Data

The driver will automatically calculate the number of complete output update cycles in the file using the offset information, the number of channels currently configured for waveform/pattern output, the size of the data samples and the total file size.

Update Cycle Data Blocks

The waveform/pattern output file is organized into sequences of update data blocks. Each update data block represent the sample data necessary to update all output channels configured for waveform/pattern output for one output cycle. Each output cycle is initiated by an update clock pulse (see `daqDacWaveSetFreq` and `daqDacWaveSetClockSource`). The sample data is ordered according to the output channel for which it will be output. Each update data block is ordered as follows:

DAC0	DAC1	DAC2	DAC3	P3 DO

If any of the above channels is not configured for waveform/pattern output then it will simply not be included in the update data block, however, the channel ordering will not change. If for example a waveform/pattern output is configured for DAC0, DAC2 and P3 DO will have the following update data block:

DAC0	DAC2	P3 DO

Update/Sample Data Format

The `dataFormat` parameter allows the selection of the format of the output sample data that applies to the file. The file must have the its sample data formatted in one of the following valid formats in order to be used for a waveform/pattern output from file operation. The while the layout of the output file is fixed, the format of the output sample data can vary greatly. Several output sample data formats are available in both binary and ASCII data formats.

Output From File Operation

Once the waveform/pattern output has been configured and the output from file has been setup using this function, the waveform/pattern output operation can be armed using the `daqDacWaveArm` function. When the waveform/pattern output operation is armed, the driver will transfer as much output data as possible to the internal output FIFO on the device. Once the waveform/pattern output operation is triggered, the driver will then refill the internal output FIFO as room becomes available from the specified output data file. The cycle data transferred from the output data file to the internal output FIFO is read from the valid output cycle data area defined by the file layout table in the "File Layout" section (unshaded update cylces)
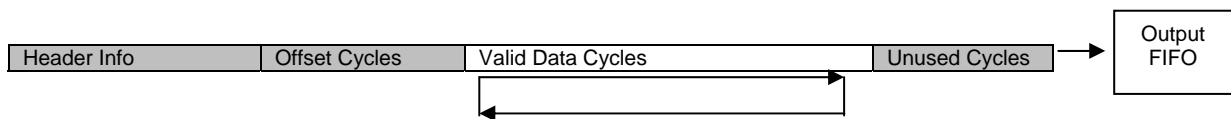
While the waveform/pattern output operation is active, the driver will continue to fill the internal output FIFO with output cycle data until the waveform/pattern output termination condition is satisfied. The waveform/pattern output termination is configured by setting the appropriate update mode using the `daqDacWaveSetMode` function. Once cycle data is transferred to the internal output FIFO on the device, the device is responsible for maintaining the output FIFO and updating the output channels every update clock pulse. When the output update clock "fires", the next available update data block is loaded from the internal waveform/pattern output FIFO on the device to the port, which then presents the values to the DAC or digital port for output.
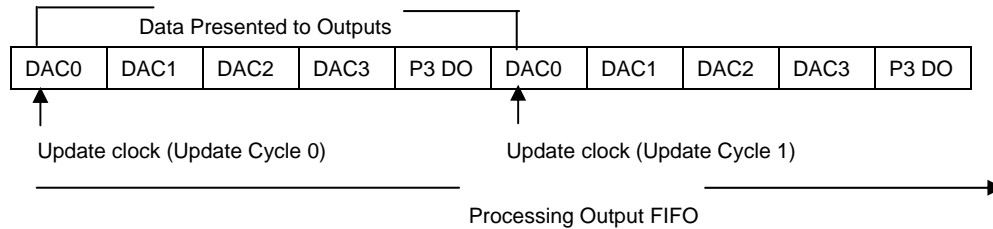
The status of a waveform/pattern output operation can be monitored using the `daqDacTransferGetStat` function. When in waveform/pattern output-from-file mode, the `retCount` parameter *indicates the number of iterations of the file and **not** the total number of update cycles.*

The amount of data transferred of the will automatically calculate the number of complete output update cycles in the file using the offset information, the number of channels currently configured for waveform/pattern output, the size of the data samples and the total file size.

Prototypes

C/C++

```
daqDacWaveSetDiskFile(DaqHandleT handle, DaqDacDeviceType deviceType,
DWORD chan, LPSTR filename, DWORD numUpdateCycles, DWORD OffsetBytes,
DWORD OffsetUpdateCycles, DaqDacWaveFileDataFormat dataFormat);
```

Visual BASIC

```
VBdaqDacWaveSetDiskFile&(ByVal handle&, ByVal deviceType&, ByVal chan&,
ByVal filename$, ByVal numUpdateCycles&, ByVal OffsetBytes&, ByVal
OffsetUpdateCycles&, ByVal dataFormat&)
```

Program References

```
None
```

daqDacWaveSetFreq

Format

```
daqDacWaveSetFreq(handle, deviceType, chan, freq)
```

Purpose

daqDacWaveSetFreq sets the waveform/pattern output update frequency (in Hz) for the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the waveform output update frequency will be set
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
freq	FLOAT	Sets the DAC waveform output frequency to the specified frequency

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table below

freq: single precision floating point value (4 bytes) that contains the update frequency from 0.0 to 500000.0 Hz

Parameter Type Definitions

DeviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

Returns

DerrNoError   No error

Function Usage

> **The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to DAC and/or digital output channels that are synchronously updated by a clock source defined by the application.**

The frequency is set via the `freq` parameter and is dependent upon the clock source chosen for the selected device. The clock source can be configured via the `daqDacWaveSetClockSource` function. The waveform/pattern output update frequency is the rate at which samples are sent from the internal output FIFO to a single output channel.

Setting the Output Update Frequency

The output update frequency is set via the `freq` parameter. This parameter sets the internal waveform/pattern output pacer clock (in Hz) and represents the frequency at which all channels configured for waveform/pattern output will be updated. Each time the configured waveform/pattern output pacer clock fires, a sample for each channel will be loaded from the internal FIFO into the appropriate DAC or digital output device and output.

The frequency programmed using the `freq` function may not be obtainable. The reason for this is that either the frequency specified is outside of the operating update frequency of the output pacer clock, or, due to the resolution of the pacer clock, the frequency is not attainable. If the latter is the case, then the output pacer clock will be programmed to the nearest attainable frequency. If the frequency is outside of the operating range of the pacer clock, then an error will be generated. See the `daqDacWaveGetFreq` function for more information on setting the output pacer clock frequency.

The specifications section of the device user's manual provides the acceptable frequency values for the device.

Prototypes

C/C++

```
daqDacWaveSetFreq(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan, FLOAT freq);
```

Visual BASIC

```
VBdaqDacWaveSetFreq&(ByVal handle&, ByVal deviceType&, ByVal chan&, ByVal
freq!)
```

Program References

DAQDACEX02.CPP, DAQDACEX03.CPP

daqDacWaveSetMode

Also See: **daqDacWaveSetTrig, daqDacWaveSetFreq**

Format

```
daqDacWaveSetMode(handle, deviceType, chan, mode, updateCount)
```

Purpose

daqDacWaveSetMode sets the waveform/pattern update mode for the output operation.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the DAC waveform output mode will be set
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
mode	DaqDacWaveformMode	Specifies the desired DAC waveform output mode
updateCount	DWORD	Sets the total sample update count

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table below

mode: see table below

updateCount: the total number update samples per channel to output can range from 1 to 4,294,967,295; however, memory limitations may apply

Parameter Type Definitions

DeviceType–(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Start the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

mode–(DaqDacWaveformMode)	
Definition	**Description**
DdwmNShot	Write the specified number of DAC points, then disarm
DdwmNFileIterations	Write the specified number of file iterations, then stop (file mode only)?
DdwmInfinite	Cycle through the buffer until the daqDacWaveDisarm function is executed

Returns

DerrNoError   No error

Function Usage

> **The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to DAC and/or digital output channels that are synchronously updated by a clock source defined by the application.**

The `daqDacWaveSetMode` function controls how the waveform/pattern is going to proceed and how the waveform/pattern output will terminate.

Setting the Update Mode

The `mode` parameter defines the state in which the waveform/pattern is to proceed and under what conditions it should terminate. Here, there are two basic modes that can be set with the `mode` parameter. The first, set by the value `DdwmNShot`, will continue the waveform/pattern output until a specified update count has been satisfied. The second, set by the value `DdwmInfinite`, will continue the waveform/pattern output indefinitely until explicitly terminated by the application. The `deviceType` and `chan` parameters must correspond with a channel that has been configured for waveform/pattern output using the `daqDacSetOutputMode` function. The `mode` values are defined as follows:

`DdwmNShot` – Continues generating waveform/pattern output until `updateCount` number of samples have been output to each channel. Upon completion of the specified amount of updates, the waveform/pattern output will automatically terminate and disarm the waveform/pattern output for all channels. For more information on configuring channels for waveform/pattern output operation, see `daqDacSetOutputMode`. Since each output update clock initiates an output on each channel concurrently, the `updateCount` parameter indicates the number of sample updates that are to occur on each channel before the waveform/pattern output operation is terminated.

`DdwmInfinite` – Continues generating waveform/pattern output indefinitely. The waveform/pattern output will continue indefinitely until the application issues a `daqDacWaveDisarm` function or a fatal error condition occurs during the waveform/pattern output operation. In this mode the `updateCount` parameter is ignored.

`DdwmNFileIterations` – Continues generating waveform/pattern output until the the specified number of file interations are written, then stops. This setting only works in file mode. Note that, while in this mode, the `updateCount` parameter of the `daqWaveSetMode` function will return values in terms of file iterations instead of updates (an update consists of all data required to update all channels during a single output clock pulse).

With either mode, the waveform/pattern output will not actually begin until the waveform/pattern output operation has been properly armed (`daqDacWaveArm`) and triggered (`daqDacWaveSetTrig`)

Prototypes

C/C++

```
daqDacWaveSetMode(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan, DaqDacWaveformMode mode, DWORD updateCount);
```

Visual BASIC

```
VBdaqDacWaveSetMode&(ByVal handle&, ByVal deviceType&, ByVal chan&, ByVal
mode&, ByVal updateCount&)
```

Program References

```
DAQDACEX02.CPP, DAQDACEX03.CPP
```

daqDacWaveSetPredefWave

Format

```
daqDacWaveSetPredefWave(handle, deviceType, chan, waveType, amplitude,
offset, dutyCycle, phaseShift)
```

Purpose

`daqDacWaveSetPredefWave` specifies a pre-defined waveform for DAC waveform output on the specified device channel.

> **daqDacWaveSetMode is used to set the update rate and cycling mode for this waveform.**

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to setup a pre-defined waveform output
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
waveType	DaqDacWaveType	Specifies the predefined waveform output type
amplitude	DWORD	Sets the peak-to-peak amplitude for which to generate the pre-defined waveform
offset	DWORD	Sets the offset for the pre-defined waveform
dutyCycle	DWORD	Sets the duty cycle as a percentage of the predefined waveform
phaseShift	DWORD	Set the phase shift (in degrees) of the predefined waveform relative to other DAC channel

Parameter Values

`handle:` obtained from the `daqOpen` function

`deviceType:` see table below

`chan:` valid values depend on the values chosen for the `deviceType` parameter; see the `deviceType` table below

`waveType:` see table below

`amplitude:` valid values range from 0 to 65,535 (in D/A counts)

`offset:` valid values range from 0 to 65,535 (representing the voltage level in D/A counts)

`dutyCycle:` duty cycle is a percentage of the predefined waveform and can range from 1 to 100

`phaseShift:` phase shift is degrees to shift the predefined waveform from 0 to 360

Parameter Type Definitions

deviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Start the transfer of waveform/pattern output data where DAC0 is one of the channels being output.
	1	Start the transfer of waveform/pattern output data where DAC1 is one of the channels being output.
	2	Start the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Start the transfer of waveform/pattern output data where DAC3 is one of the channels being output.

WaveType-(DaqDacWaveType)	
Definition	**Description**
DdwtSine	Output a sine wave
DdwtSquare	Output a square wave
DdwtTriangle	Output a triangle wave

Returns

DerrInvDacChan	The DAC channel number doesn't exist
DerrInvDacParam	Parameters were out-of-range
DerrInvPredefWave	Predefined waveform is not supported
DerrMemAlloc	Not enough memory was available to build the waveform
DerrNotCapable	Hardware is not capable of this function
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The daqDacWaveSetPredefWave function creates the defined waveform on the specified channel buffer as soon as the acquisition is armed—however, this function should only be used in static output mode.

DaqBoard (ISA-Type), DaqBook/2000 Series, DaqBoard/2000 Series, and cPCI DaqBoard/2000c Series

When using the daqDacWaveSetPredefWave function with the DaqBoard(ISA) devices, it is important to note that they use 12-bit DAC's. However, for consistancy, all functions use 16-bit values. The driver will convert 16-bit parameters to appropriate 12-bit values for the DaqBoard (ISA), while the 16-bit value is passed directly to 1000 Series and 2000 Series devices.

Thus, an amplitude or offset value of 0 corresponds to the **minimum output** of the DAC, as follows:

- -10V for DaqBoard/2000 Series
- 0V for DaqBoard (ISA-type)

A value of 65,535 corresponds to the maximum output of the DAC, as follows:

- 10V for DaqBoard/2000 Series
- 5V for DaqBoard (ISA-type)

The specifications section of the device user's manual provides the resolution values for the device; for example: the voltage for a DaqBoard/2000 Series devices has a resolution of approximately 0.305 mV (20 V / 65,535).

Prototypes

C/C++

```
daqDacWaveSetPredefWave(DaqHandleT handle, DaqDacDeviceType deviceType,
DWORD chan, DaqDacWaveType waveType, DWORD amplitude, DWORD offset, DWORD
dutyCycle, DWORD phaseShift);
```

Visual BASIC

```
VBdaqDacWaveSetPredefWave&(ByVal handle&, ByVal deviceType&, ByVal chan&,
ByVal waveType&, ByVal amplitude&, ByVal offset&, ByVal dutyCycle&, ByVal
phaseShift&)
```

Program References

```
DAQDACEX02.CPP, DAQDACEX03.CPP
```

daqDacWaveSetTrig

Also See: **daqDacWaveSetMode**

Format

```
daqDacWaveSetTrig(handle, deviceType, chan, triggerSource, rising)
```

Purpose

`daqDacWaveSetTrig` sets the trigger event for the waveform/pattern output operation.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle of the device for which to set DAC waveform triggering
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	Specifies the DAC channel
triggerSource	DaqDacTriggerSource	Specifies the DAC output trigger source
rising	BOOL	Boolean indicating the trigger source edge

Parameter Values

handle: obtained from the `daqOpen` function

deviceType: see table below

chan: valid values depend on the values chosen for the `deviceType` parameter; see the `deviceType` table below

triggerSource: see table below

rising: this parameter is ignored with the current trigger source definitions available with this command

Parameter Type Definitions

deviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Set the trigger source of waveform/pattern output operation where DAC0 is one of the channels being output.
	1	Get the trigger source of waveform/pattern output operation where DAC1 is one of the channels being output.
	2	Set the transfer of waveform/pattern output data where DAC2 is one of the channels being output.
	3	Set the transfer of waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Set the transfer waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

triggerSource-(DaqDacTriggerSource)	
Definition	**Description**
DdtsImmediate	Output immediately after arming
DdtsSoftware	Output upon calling the `daqDacWaveSoftTrig`function from user or software
DdtsAdcClock	Output on a signal from the ADC clock.

Returns

`DerrNoError`   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to DAC and/or digital output channels that are synchronously updated by a clock source defined by the application.

The `daqDacWaveSetTrig` function is used to setup the trigger event to initiate a waveform/pattern output for all channels which have been configured for waveform/pattern output mode. For more information on configuring channels for waveform/pattern output, see the `daqDacSetOutputMode` function.

Setting the Waveform/pattern Output Trigger Event

The `triggerSource` parameter specifies the source of the event that will trigger the waveform/pattern output operation. Currently, there are only two valid trigger events that can be set with the `triggerSource` parameter. The first, set with the value `DdtsImmediate`, indicates that the waveform/pattern output operation should trigger immediately upon being armed. The second, set with the value `DdtsImmediate`, indicates that the waveform/pattern output operation should be triggered upon the application issuing a command to do so. The `deviceType` and `chan` parameters must correspond with a channel that has been configured for waveform/pattern output using the `daqDacSetOutputMode` function. The trigger event values are defined as follows:

`DdtsImmediate` – Trigger the waveform/pattern output operation immediately upon execution of the `daqDacWaveArm` function. This trigger source is used to trigger the waveform/pattern output immediately upon successfully arming the operation. Since, however, many of the configuration error conditions and conflicts are detected during the arm operation, it is important to handle error conditions properly when using this trigger source.

`DdtsSoftware` – Trigger the waveform/pattern output operation upon execution of the `daqDacWaveSoftTrig` function. This trigger source requires that the `daqDacWaveArm` function be issued before the `daqDacWaveSoftTrig` function. Once armed, the application may trigger the waveform/pattern output operation at any time by issuing `daqDacWaveSoftTrig` function.

`DdtsAdcClock` – Trigger on the ADC clock, useful when trying to synchronize the waveform/pattern output with the ADC clock. This trigger source is valid only for Daq 2000 Series devices.

The rising flag is currently ignored and is reserved for future use.

Prototypes

C/C++

```
daqDacWaveSetTrig(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan, DaqDacTriggerSource triggerSource, BOOL rising);
```

Visual BASIC

```
VBdaqDacWaveSetTrig&(ByVal handle&, ByVal deviceType&, ByVal chan&, ByVal
triggerSource&, ByVal rising&)
```

Program References

`DAQDACEX02.CPP, DAQDACEX03.CPP`

daqDacWaveSetUserWave

Format

```
daqDacWaveSetUserWave(handle, deviceType, chan)
```

Purpose

daqDacWaveSetUserWave configures a user-defined buffer for DAC waveform output.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to which the user-defined waveform will be output
deviceType	DaqDacDeviceType	Specifies the DAC device type
chan	DWORD	Specifies the DAC device channel

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table below

Parameter Type Definitions

deviceType–(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Configure a user defined waveform for waveform/pattern output operation, where DAC0 is one of the channels being output.
	1	Configure a user defined waveform for waveform/pattern output operation, where DAC1 is one of the channels being output.
	2	Configure a user defined waveform for waveform/pattern output operation, where DAC2 is one of the channels being output.
	3	Configure a user defined waveform for waveform/pattern output operation, where DAC3 is one of the channels being output.
DddtLocalDigital	NA	The 16-bit P3 Digital port is not applicable.

Returns

DerrInvDacChan	The DAC channel number doesn't exist
DerrInvBuf	A waveform buffer was not specified
DerrMemAlloc	Not enough memory was available to build the waveform
DerrNotCapable	Hardware is not capable of this function
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to DAC and/or digital output channels that are synchronously updated by a clock source defined by the application.

Any arbitrary waveform can be built in an array. `daqDacWaveSetUserWave` can then be called by specifying pointers to the beginning of the waveform, the size of the array, and the target DAC channel to send the waveform.

The data value should be 16-bit WORD which will be converted to the corresponding voltage by the D/A converter. Reference the device user's manual in regard to D/A range and resolution.

For example; the voltage for a DaqBoard/2000 Series board has a resolution of approximately 0.305 mV (20 V / 65,535).

Prototypes

C/C++

```
daqDacWaveSetUserWave(DaqHandleT handle, DaqDacDeviceType deviceType,
DWORD chan);
```

Visual BASIC

```
VBdaqDacWaveSetUserWave&(ByVal handle&, ByVal deviceType&, ByVal chan&)
```

Program References

```
DAQDACEX03.CPP
```

daqDacWaveSoftTrig

Format

```
daqDacWaveSoftTrig(handle, deviceType, chan)
```

Purpose

daqDacWaveSoftTrig activates a software trigger for the waveform/pattern output operation on the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which to trigger the DAC waveform output
deviceType	DaqDacDeviceType	Specifies the DAC device type
chan	DWORD	Specifies the DAC device channel

Parameter Values

handle: obtained from the daqOpen function

deviceType: see table below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table below

Parameter Type Definitions

DAC Device Type Definitions – **DaqDacDeviceType**		
Definition	**chan Value**	**Description**
DddtLocal	0	Software trigger the waveform/pattern output operation where DAC0 is one of the channels being output.
	1	Software trigger the waveform/pattern output operation where DAC1 is one of the channels being output.
	2	Software trigger the waveform/pattern output data where DAC2 is one of the channels being output.
	3	Software trigger the waveform/pattern output data where DAC3 is one of the channels being output.
DddtLocalDigital	0	Software trigger the waveform/pattern output data where the 16-bit P3 Digital port is one of the channels being output.

Returns

DerrNoError   No error

Function Usage

The term *waveform/pattern output* is used extensively throughout the entries for the functions of the form daqDac…. This refers to an analog waveform output modes and/or digital pattern output modes. These terms describe output to DAC and/or digital output channels that are synchronously updated by a clock source defined by the application.

The trigger event must first have been configured for software triggering with the `daqDacWaveSetTrig` function prior to calling this function, and the trigger source must have been set to `DdtsSoftware`. Once issued, the waveform/pattern output operation will begin; the updating of all channels configured for waveform/pattern output will also begin. The `deviceType` and `chan` parameters must correspond with a channel that has been configured for waveform/pattern output using the `daqDacSetOutputMode` function.

Prototypes

C/C++

```
daqDacWaveSoftTrig(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD
chan);
```

Visual BASIC

```
VBdaqDacWaveSoftTrig&(ByVal handle&, ByVal deviceType&, ByVal chan&)
```

Program References

```
DAQDACEX03.CPP
```

daqDacWt

Format

```
daqDacWt(handle, deviceType, chan, dataVal)
```

Purpose

daqDacWt sets the output value of a local or expansion DAC channel.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device whose channel value will be updated
deviceType	DaqDacDeviceType	Specifies the DAC type
chan	DWORD	D/A channel to output to the selected D/A channel
dataVal	WORD	Value to output to the selected D/A channel

Parameter Values

handle: obtained from the daqOpen function

deviceType: see the table in the "Selecting the Output Channel" section below

chan: valid values depend on the values chosen for the deviceType parameter; see the deviceType table in the "Selecting the Output Channel" section below

dataVal: valid values range from 0 to 65,535

Returns

DerrInvChan	Invalid channel
DerrInvDacVal	Invalid data value
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The channel specified by the daqDacWt function must be an appropriate DAC channel type, and the channel must have been configured as a direct output channel prior to calling this function. See daqDacSetOutputMode for setting the output mode to be direct (using the DdomVoltage value).

Setting the DAC Value

The value of the DAC channel is written using the dataVal parameter. This parameter is an integer value ranging from 0 to 65,535. This is a 16-bit value, and it spans the range over which the output can vary, depending upon the specified range of the port for the device.

Some devices use 16-bit D/A converters, in which case each bit in the 16-bit integer corresponds to one bit programmed to the D/A converter. However, some devices have 12-bit D/A converters. In this case, the low order nibble (4-bits) of the programmed dataVal value will be ignored when the D/A is programmed.

The specifications section of the device user's manual provides the ranges and the associated resolution values for the device.

Selecting the Output Channel

The `chan` and the `deviceType` parameters determine which output channel's voltage will be set. The device type is represented by predefined settings described below. The channel is simply an integer (starting at 0) which represents the channel location relative to the first channel of the same type. The table below describes this relationship:

deviceType-(DaqDacDeviceType)		
Device Type	**chan Value**	**Description**
DddtLocal	0	Output data value to DAC0
	1	Output data value to DAC1
	2	Output data value to DAC2
	3	Output data value to DAC3
DddtDbk	$N*4$ + 0	Set output for channel0 on a DBK2 or DBK5 on bank N
	$N*4$ + 0	Set output for channel1 on a DBK2 or DBK5 on bank N
	$N*4$ + 0	Set output for channel2 on a DBK2 or DBK5 on bank N
	$N*4$ + 0	Set output for channel3 on a DBK2 or DBK5 on bank N
DddtLocalDigital	N/A	The 16-bit P3 Digital port is not applicable for analog output operations

Note! **If using an analog hardware trigger with DaqBook, TempBook or DaqBoard(ISA) products, the DAC channel 1 (deviceType = DddtLocal and chan = 1) is not available to be programmed. The reason for this is that the DAC channel 1 is used to configure the trigger level for the acquisition.**

Prototypes

C/C++

```
daqDacWt(DaqHandleT handle, DaqDacDeviceType deviceType, DWORD chan, WORD
dataVal);
```

Visual BASIC

```
VBdaqDacWt&(ByVal handle&, ByVal deviceType&, ByVal chan&, ByVal dataVal%)
```

Program References

```
DAQEX.FRM (VB)
```

daqDacWtMany

Format

    daqDacWtMany(handle, deviceTypes, chans, dataVals, count)

Purpose

daqDacWtMany sets the output values of multiple local or expansion DAC channels.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device whose channel values will be updated
deviceTypes	DaqDacDeviceType	Pointer to an array which specifies the DAC types
chans	PDWORD	Pointer to any array which specifies the DAC channels
dataVals	PWORD	Pointer to an array which gives a value to output to the D/A channel
count	DWORD	Length of all the arrays

Parameter Values

handle: obtained from the daqOpen function

deviceTypes: see table below

chans: pointer to an array of output channel numbers; for channel number values, refer to the deviceTypes table below

dataVals: valid values range from 0 to 65,535

count: the total number of analog channels to output range from 1 to 512

Parameter Type Definitions

deviceType-(DaqDacDeviceType)		
Definition	**chan Value**	**Description**
DddtLocal	0	Output data value to DAC0
	1	Output data value to DAC1
	2	Output data value to DAC2
	3	Output data value to DAC3
DddtDbk	$N*4 + 0$	Set output for channel0 on a DBK2 or DBK5 on bank N
	$N*4 + 0$	Set output for channel1 on a DBK2 or DBK5 on bank N
	$N*4 + 0$	Set output for channel2 on a DBK2 or DBK5 on bank N
	$N*4 + 0$	Set output for channel3 on a DBK2 or DBK5 on bank N
DddtLocalDigital	NA	The 16-bit P3 Digital port is not applicable for analog output operations

Returns

DerrInvDacVal	Invalid data value
DerrNoError	No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The channels specified by the `daqDacWtMany` function must be the appropriate DAC channel types, and the channels must have been configured as direct output channels prior to calling this function. See `daqDacSetOutputMode` for setting the output mode to be direct (`DdomVoltage`).

The `daqDacWtMany` function operates in a similar manner to the single output version of the function, `daqDacWt`. Here, the `deviceTypes` and `chans` parameters are arrays of DAC channel types and channel numbers. The `dataVals` parameter is an array of output values corresponding to the `deviceTypes` and `chans` arrays. The total number of channels to update (which should be equivalent to the number of entries in each array) is set by the `count` parameter. Since this function is a multichannel version of the `daqDacWt` function, please refer to the `daqDacWt` function for a complete description of the parameters and other settings.

Prototypes

C/C++

```
daqDacWtMany(DaqHandleT handle, DaqDacDeviceType *deviceTypes, PDWORD
chans, PWORD dataVals, DWORD count);
```

Visual BASIC

```
VBdaqDacWtMany&(ByVal handle&, deviceTypes&, chans&, dataVals&, ByVal
count&)
```

Program References

DAQDACEX01.CPP, DBK02Ex.cpp, DBK05Ex.Cpp DAQEX.FRM (VB)

daqDefaultErrorHandler

Format

```
daqDefaultErrorHandler(handle, errCode)
```

Purpose

daqDefaultErrorHandler displays an error message and then exits the application program.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to which will be attached to the default error handler
errCode	DaqError	Error code number of the detected error

Parameter Values

handle: obtianed form the daqOpen function
errCode: valid return values can be found in the Daq Error Table

Returns

None

Function Usage

When the device library is loaded, it invokes the default error handler whenever it encounters an error. The error handler may be changed with daqSetErrorHandler.

For more details on error messages refer to the Daq Error Table.

Prototypes

C/C++

```
daqDefaultErrorHandler(DaqHandleT handle, DaqError errCode);
```

Visual BASIC

```
VBdaqDefaultErrorHandler(ByVal handle&, ByVal errCode&)
```

Program References

DAQADCEX05.CPP, DAQADCEX06.CPP

daqDeleteDevice

Format

```
daqDeleteDevice(AliasName)
```

Purpose

daqDeleteDevice deletes the specified device from the Windows Registry

Parameter Summary

Parameter	Type	Description
AliasName	LPSTR	Pointer to a character string containing the alias name of the device to delete

Parameter Values

AliasName: Pointer to a character string containing the alias name of the device to delete

Returns

DerrNoError if successful or a DaqError error code (see the DaqError table at the end of chapter 4).

Function Usage

This function deletes the specified device from the Windows Registry. Devices created using the daqCreateDevice function or the DaqX control panel applet can be deleted with this function.

Prototypes

C/C++

daqDeleteDevice(LPSTR AliasName);

Visual BASIC

VBdaqDeleteDevice&(ByVal AliasName$)

Program References

```
None
```

daqFormatError

Format

```
daqFormatError (errorNum, msg)
```

Purpose

daqFormatError returns the text-string equivalent for the specified error condition code.

Parameter Summary

Parameter	Type	Description
errorNum	DaqError	Error condition whose text will be returned
msg	PCHAR	Pointer to a character string which will store the returned error text

Parameter Values

errorNum: see the Daq Error Table
msg: pointer to a character string which will hold error text; see the Daq Error Table for more details

Returns

DerrNoError   No error

Function Usage

The error condition is specified by the errorNum parameter. The error text will be returned in the character string pointed to by the msg parameter. The character string space must have been previously allocated by the application before calling this function. The allocated character string should be, at minimum, 64 bytes in length.

For more details on error messages refer to the Daq Error Table.

Prototypes

C/C++

```
daqFormatError(DaqError errorNum, PCHAR msg)
```

Visual BASIC

```
VBdaqFormatError&(ByVal errorNum&, ByRef msg As Byte)
```

Program References

None

daqGetChannelType

Format

    daqGetChannelType(handle, chan, option, pChSubType)

Purpose

daqGetChannelType gets the channel type and sub type of the specified channel of the current device

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the channel type and sub type is to be obtained
chan	DWORD	Specifies which channel
option	BOOL	Selects an option card or main unit if applicable
pChSubType	DaqChannelSubType *	Pointer to a value that will contain the channel sub type

Parameter Values

handle: obtained from the daqOpen function

channel: from 0 to the maximum number of channels available on the system

option: 0 for the main unit or non-zero for an option card

pChSubType: Pointer to a value that will contain the channel sub type or NULL if the channel sub type is not desired
  (see DaqAdcExpSubType of the daqGetHardwareInfo function)

Returns

The channel type of the specified channel of the current device (see DaqAdcExpType of the daqGetHardwareInfo function)

Function Usage

This function gets the channel type and sub type of the current device.

Prototypes

C/C++

daqGetChannelType(DaqHandleT handle, DWORD chan, BOOL option, DaqAdcExpSubType *pChSubType);

Visual BASIC

VBdaqGetChannelType&(ByVal handle&, ByVal chan&, ByVal bOption&, ChSubType&)

Program References

None

daqGetDeviceCount

Format

```
daqGetDeviceCount (deviceCount)
```

Purpose

daqGetDeviceCount returns the number of currently configured devices**.**

Parameter Summary

Parameter	Type	Description
deviceCount	DWORD	Pointer to which the device count is to be returned

Parameter Values

deviceCount: a pointer to a single value ranging from 1 to 4

Returns

DerrNoError   No error

Function Usage

The daqGetDeviceCount function will return the number of devices currently configured in the system. The devices do not need to be opened for this function to properly detect the number of devices configured. If the number returned does not seem appropriate, the device configuration list should be checked via the **Daq* Configuration** applet located in the Control Panel. Refer to the configuration section in your device's user manual for more details (also see daqOpen for the **Daq* Configuration** applet example).

Prototypes

C/C++

```
daqGetDeviceCount(DWORD deviceCount);
```

Visual BASIC

```
VBdaqGetDeviceCount&(devCount As Long)
```

Program References

DAQADCEX01.CPP

daqGetDeviceInfo

Format

```
daqGetDeviceInfo(handle, pDevInfo)
```

Purpose

daqGetDeviceInfo gets the device information for the currently opened device

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the device information is to be obtained
pDevInfo	DaqDevInfoPT	Pointer to the device information structure to return the device information

Parameter Values

handle: obtained from the daqOpen function
pDevInfo: see the DaqDevInfoT table of the daqCreateDevice function

Returns

DerrNoError if successful or a DaqError error code (see the DaqError table at the end of chapter 4)

Function Usage

This function will get the device information of the device actually connected. This information may be slightly different than the information contained in the Windows Registry. For example, if a WaveBook/516A was created using the DaqX control panel applet, but a WaveBook/512A is actually connected, the device information structure returned by this function will reflect the WaveBook/512A.

Prototypes

C/C++

daqGetDeviceInfo(DaqHandleT handle, DaqDevInfoPT pDevInfo);

Visual BASIC

VBdaqGetDeviceInfo&(ByVal handle&, DevInfo As DaqDevInfoT)

Program References

None

daqGetDeviceInventory

Format

daqGetDeviceInventory(pInfoList, pInfoListCount, pMatchInfo, flags)

Purpose

daqGetDeviceInventory builds a device information list of devices that exist in the Windows Registry and/or devices that can be dynamically detected

Parameter Summary

Parameter	Type	Description
pInfoList	DaqDevInfoPT	Pointer an array of device information structures
pInfoListCount	PDWORD	Pointer to the number of elements in the device information list
pMatchInfo	DaqDevInfoPT	Pointer to a device information structure containing device matching information
flags	DWORD	One or more device information flags

Parameter Values

pInfoList: a pointer to an array of DaqDevInfoT structures (see the DaqDevInfoT table of the daqCreateDevice function)

pInfoListCount: upon entry, the value that pInfoListCount points to should be set to the number of elements in the information list array; upon exit, the value that pInfoListCount points to will be set to the number of elements in the information list that were modified

pMatchInfo: if any of the match info flags are set in the flags parameter, pMatchInfo should point to a device information structure containing the appropriate match information, otherwise pMatchInfo can be NULL

flags: see the DaqDevInfoFlagsT table below

Parameter Type Definitions

flags – (DaqDevInfoFlagsT)	
Definition	Description
DaqInfoFlagsCreated	Enables searching the Windows registry for created devices and returns those that are found
DaqInfoFlagsNotCreated	Enables searching the Windows registry for created devices but removes those that are found
DaqInfoFlagsDetected	Enables searching for detectable devices and returns those that are detected
DaqInfoFlagsNotDetected	Enables searching for detectable devices but removed those that are detected
DaqInfoFlagsTestPassed	Enables testing of detected and/or created devices and returns those that pass the communications test
DaqInfoFlagsTestFailed	Enables testing of detected and/or created devices and returns those that pass the communications test
DaqInfoFlagsNew	Returns devices that are detected, but not yet created (DaqInfoFlagsDetected + DaqInfoFlagsNotCreated)
DaqInfoFlagsRemoved	Returns devices that are created, but failed the communications test (DaqInfoFlagsCreated + DaqInfoFlagsTestFailed)
DaqInfoFlagsAvailable	Returns devices that are created, and passed the communications test (DaqInfoFlagsCreated + DaqInfoFlagsTestPassed)
DaqInfoFlagsMatchAlias	Causes any devices whose alias does not match the specified alias to be removed from the list
DaqInfoFlagsMatchParentAlias	Causes any devices whose parent alias does not match the specified alias to be removed from the list
DaqInfoFlagsMatchType	Causes any devices whose device type does not match the specified device type to be removed from the list
DaqInfoFlagsMatchSubType	Causes any devices whose device type and device sub type do not match the specified device type and device sub type to be removed from the list
DaqInfoFlagsMatchProdGroup	Causes any devices whose device product group does not match the product group of the specified device type to be removed from the list
DaqInfoFlagsMatchProdFamily	Causes any devices whose device product family does not match the product family of the specified device type to be removed from the list

Returns

DerrNoError if successful or a DaqError error code (see the DaqError table at the end of chapter 4)

Function Usage

This function will return the subset of created and detected devices specified by the flags parameter. At least one of the DaqInfoFlagsCreated, DaqInfoFlagsNotCreated, DaqInfoFlagsDetected or DaqInfoFlagsNotDetected flags must be specified and no more than one of the DaqInfoFlagsMatch flags can be specified (see preceding table).

The AliasName element of the pMatchInfo structure must be set if the DaqInfoFlagsMatchAlias or DaqInfoFlagsMatchParentAlias flag is set.

The DeviceType element of the pMatchInfo structure must be set if the DaqInfoFlagsMatchType, DaqInfoFlagsMatchSubType, DaqInfoFlagsMatchProdGroup or DaqInfoFlagsMatchProdFamily flag is set.

The DeviceSubType element of the pMatchInfo structure must be set if the DaqInfoFlagsMatchSubType flag is set.

Prototypes

C/C++

daqGetDeviceInventory(DaqDevInfoPT pInfoList, PDWORD pInfoListCount, DaqDevInfoPT pMatchInfo, DWORD flags);

Visual BASIC

VBdaqGetDeviceInventory&(InfoList() As DaqDevInfoT, InfoListCount As Long, ByVal Flags As Long, MatchInfo As DaqDevInfoT)

Program References

None

daqGetDeviceList

Format

```
daqGetDeviceList (deviceList, deviceCount)
```

Purpose

daqGetDeviceList returns a list of currently configured device names.

Parameter Summary

Parameter	Type	Description
deviceList	DaqDeviceListT	Pointer to memory location to which the device list is to be returned
deviceCount	DWORD	Number of devices returned in the device list

Parameter Values

deviceList: pointer to an array of returned device names, each name containing up to 64 characters

deviceCount: valid values range from 1 to 4

Returns

DerrNoError   No error

Function Usage

The daqGetDeviceList function will return the device names in the deviceList parameter for the number of devices returned by the deviceCount parameter. The deviceList entry contains an array of device names each consisting of up to 64 characters. Each device name can then be used with the daqOpen function to open the specific device. The DaqDeviceListT parameter must point to an appropriately sized memory area which can hold all the names for all the configured devices before calling this function. If it is not known how many devices are configured, then call the daqGetDeviceCount function before calling this function.

If the number returned does not seem appropriate, the device configuration list should be checked via the **Daq* Configuration** applet located in the Control Panel. Refer to the configuration section in your device's user manual for more details.

Prototypes

C/C++

```
daqGetDeviceList(DaqDeviceListT *deviceList, DWORD *deviceCount);
```

Visual BASIC

```
VBdaqGetDeviceList&(devList As DaqDeviceListT, devCount As Long)
```

Program References

DAQADCEX01.CPP

daqGetDeviceProperties

Format

daqGetDeviceProperties(daqName, deviceProps)

Purpose

daqGetDeviceProperties returns the properties for a specified device.

Parameter Summary

Parameter	Type	Description
daqName	LPSTR	Pointer to a character string representing the name of the device for which to retrieve properties
deviceProps	DaqDevicePropsT	Pointer to the memory area for which to return the properties of the device specified by daqName

Parameter Values

daqName: a pointer to a string of characters—no effective range of values applies

deviceProps: see table below

Parameter Type Definitions

deviceProps-(DaqDevicePropsT)		
Definition	**Description**	**Format**
deviceType	Main Chassis Device Type Definition	**DWORD**
basePortAddress	Port Address (ISA Address, LPT Port, etc)	**DWORD**
dmaChannel	DMA Channel (if applicable)	**DWORD**
protocol	Interface Protocol	**DWORD**
alias	Device Alias Name	**STRING**
maxAdChannels	Maximum A/D channels (with full expansion)	**DWORD**
maxDaChannels	Maximum D/A channels (with full expansion)	**DWORD**
maxDigInputBits	Maximum Dig. Inputs (with full expansion)	**DWORD**
maxDigOutputBits	Maximum Dig. Outputs (with full expansion)	**DWORD**
maxCtrChannels	Maximum Counter/Timers (with full expansion)	**DWORD**
mainUnitAdChannels	Maximum Main Unit A/D channels (no expansion)	**DWORD**
mainUnitDaChannels	Maximum Main Unit D/A channels (no expansion)	**DWORD**
mainUnitDigInputBits	Maximum Main Unit Digital Inputs (no expansion)	**DWORD**
mainUnitDigOutputBits	Maximum Main Unit Digital Outputs (no expansion)	**DWORD**
mainUnitCtrChannels	Maximum Main Unit Counter/Timer channels (no exp.)	**DWORD**
adFifoSize	A/D on-board FIFO Size	**DWORD**
daFifoSize	D/A on-board FIFO Size	**DWORD**
adResolution	Maximum A/D Converter Resolution	**DWORD**
daResolution	Maximum D/A Converter Resolution	**DWORD**
adMinFreq	Minimum A/D Conversion Scan Frequency (Hz)	**FLOAT**
adMaxFreq	Maximum A/D Conversion Scan Frequency (Hz)	**FLOAT**
daMinFreq	Minimum D/A Output Update Frequency (Hz)	**FLOAT**
daMaxFreq	Maximum D/A Output Update Frequency (Hz)	**FLOAT**

If this function fails, make sure the daqName parameter references a valid device that is currently configured. This can be checked via the Daq* Configuration applet located in the Control Panel. Refer to the configuration section in your device's user manual for more details.

Returns

DerrNoError   No Error

Function Usage

Passing the name of the device in the `daqName` parameter specifies the device. This name should be a valid name of a configured device. The properties for the device are returned in the `deviceProps` parameter. The `deviceProps` parameter is a pointer to an application-allocated memory area which will hold the device-properties structure. This memory must have been allocated before calling this function. Upon return, the memory area pointed to by the `deviceProps` parameter will contain the properties for the device.

Prototypes

C/C++

```
daqGetDeviceProperties(LPSTR daqName, DaqDevicePropsT *deviceProps);
```

Visual BASIC

```
VBdaqGetDeviceProperties(daqName$, deviceProps as DaqDevicePropsT)
```

Program References

```
DAQADCEX01.CPP
```

daqGetDriverVersion

Format

daqGetDriverVersion (version)

Purpose

daqGetDriverVersion retrieves the revision level of the driver currently in use.

Parameter Summary

Parameter	Type	Description
version	PDWORD	Pointer to the version number of the current device driver

Parameter Values

version: a pointer to a value from 100 to 10,000

Returns

DerrNoError   No error

Prototypes

C/C++

daqGetDriverVersion(PDWORD version);

Visual BASIC

VBdaqGetDriverVersion&(version&)

Program References

None

daqGetHardwareInfo

Also See : **daqGetDriverVersion, daqOpen**

Format

    daqGetHardwareInfo(handle, whichInfo, info)

Purpose

daqGetHardwareInfo retrieves hardware information for the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device
whichInfo	DaqInfo	Specifies what type of device information to retrieve
info	VOID	Pointer to the information returned from the selected device; data returned varies according to info type selected

Parameter Values

handle: obtained from the daqOpen function

whichInfo: see table below

info: pointer to a returned value; value ranges depend on information requested

Parameter Type Definitions

whichInfo–(DaqHardwareInfo)	
Definition	**Description**
DhiHardwareVersion	Returns value of type DaqHardwareVersion (see table below)
DhiProtocol	Returns value of type DaqProtocol (see table below)
DhiADmin	ADC Output Low Range
DhiADmax	ADC Output High Range
DhiHardwareSubType	Returns value of type DaqHardwareSubType (see next page)

The following tables give the possible return values when DaqHarwareInfo is set to DhiHardwareVersion or DhiProtocol. DaqSubType information follows the table below.

DhiHardwareVersion		**DhiHardwareVersion**	
Value Returned	**Device**	**Value Returned**	**Device**
0	DaqBook100	23	DaqBook2000 Series Device (A, E, or X)
1	DaqBook112	24, 25, 26, 27, 28	Reserved
2	DaqBook120	29	WaveBook/512A
3	DaqBook200 or DaqBook/260	30	WaveBook/516A, WaveBook/516E
4	DaqBook216	31	WBK25
5	DaqBoard100	32	WBK40
6	DaqBoard112	33	WBK41
7	DaqBoard200	34	DaqBoard/1000
8	DaqBoard216	35	DaqBoard/1005
9	Daq112	36	DaqLab/2000 Series
10	Daq216	37	DaqScan/2000 Series
11	WaveBook512	38	DaqBoard/500
12	WaveBook516	39	DaqBoard/505
13	TempBook66	40	DaqBoard/3000
14	PersonalDaq56	41	DaqBoard/3001
15	WaveBook516_250	42	DaqBoard/3005
16	WaveBook512_10V	43	UsbDaqDevice
17	DaqBoard2000, DaqBoard/2000c	44	PersonalDaq3000
18	DaqBoard2001, DaqBoard/2001c	45	ZonicPod
19	DaqBoard2002, DaqBoard/2002c	46	DaqBoard/3006
20	DaqBoard2003, DaqBoard/2003c	47	DaqBoard/3000USB
21	DaqBoard2004, DaqBoard/2004c	0x100 - DaqTemp7	0x200 – DaqTemp7A
22	DaqBoard2005, DaqBoard/2005c	0x400 - DaqTemp14	0x800 - DaqTemp14A

Hardware Sub Type Definitions

typedef enum {

        // a hardware sub type of DaqSubTypeDefault indicates that the hardware device type

        // is defined by the hardware version

        DaqSubTypeDefault   = 0,

        // sub types for the DaqBook2000 main type

        DaqSubTypeDaqBook2000A = 0, // DaqBook/2000A or DaqBook/2000X

        DaqSubTypeDaqBook2000E = 1, // DaqBook/2000E

        DaqSubTypeDaqBook2001 = 2, // DaqBook/2001

        DaqSubTypeDaqBook2020 = 3, // DaqBook/2020

        DaqSubTypeDaqBook2005 = 4, // DaqBook/2005

        // sub types for the DaqLab2000 main type

        DaqSubTypeDaqLab2001   = 0, // DaqLab/2001

        DaqSubTypeDaqLab2005   = 1, // DaqLab/2005

        // sub types for the DaqScan2000 main type

        DaqSubTypeDaqScan2001 = 0, // DaqScan/2001

        DaqSubTypeDaqScan2002 = 1, // DaqScan/2002

        DaqSubTypeDaqScan2004 = 2, // DaqScan/2004

        DaqSubTypeDaqScan2005 = 3, // DaqScan/2005

        // sub types for the WaveBook516A main type

        DaqSubTypeWaveBook516A = 0, // WaveBook/516A

        DaqSubTypeWaveBook516E = 1, // WaveBook/516E

        DaqSubTypeZonicBook618 = 2, // ZonicBook/618

        // sub types for the DaqBoard2000 main type

        DaqSubTypeDaqBoard2000 = 0, // DaqBoard/2000

        DaqSubTypeDaqTemp7A   = 1, // DaqTemp/7A

        // sub types for the DaqBoard2001 main type

        DaqSubTypeDaqBoard2001 = 0, // DaqBoard/2001

        DaqSubTypeDaqTemp14A   = 1, // DaqTemp/14A

        // sub types for the DaqBoard2005 main type

        DaqSubTypeDaqBoard2005 = 0, // DaqBoard/2005

        DaqSubTypeDaqTemp7   = 1, // DaqTemp/7

        DaqSubTypeDaqTemp14   = 2, // DaqTemp/14

// sub types for the UsbDaqDevice main type

```
DaqSubTypeUnknownUsbDaqDevice          = 0,
DaqSubTypeDevelUsbDaqDevice            = 1,
DaqSubTypeBlankUsbDaqDevice            = 2,
```

// sub types for the PersonalDaq3000 main type

```
DaqSubTypePersonalDaq3001     = 0,
DaqSubTypePersonalDaq3005     = 1,
```

// sub types for the ZonicPod main type

```
DaqSubTypeZonicPod4     = 0,
DaqSubTypeZonicPod8     = 1,
```

// sub types for the DaqBoard3000USB main type

```
DaqSubTypeDaqBoard3000USB     = 0,
DaqSubTypeDaqBoard3001USB     = 1,
DaqSubTypeDaqBoard3005USB     = 2,
```

```
} DaqHardwareSubType;
```

DhiProtocol	
Definition	**Description**
DaqProtocolNone	Communications not established
DaqProtocol4	Standard LPT Port 4-bit mode
DaqProtocol8	Standard LPT Port 8-bit mode
DaqProtocolSMC666	SMC 37C666 EPP mode
DaqProtocolFastEPP	WBK20/21 Fast EPP mode
DaqProtocolECP	Enhanced Capability Port
DaqProtocol8BitEPP	8-bit EPP mode
DaqProtocolISA	ISA bus card DaqBoard 100/200
DaqProtocolPcCard	PCCard for Daq (PCMCIA)
DaqProtocolUSB	USB protocol (PersonalDaq)
DaqProtocolPCI	PCI bus card, DaqBoard/2000 Series
DaqProtocolCPCI	Compact PCI (cPCI) bus card, cPCI DaqBoard/2000c Series
DaqProtocolTCPIP	Ethernet

Returns

DerrNoError   No error

Function Usage

This function has been obsoleted by the `daqGetInfo` function, and his presented here only as a reference. See `daqGetInfo` for more details.

The daqGetHardwareInfo function retrieves hardware information for the device specified by the handle parameter. The device must have been opened previously to calling daqGetHardwareInfo by the daqOpen function.

Prototypes

C/C++

daqGetHardwareInfo(DaqHandleT handle, DaqInfo whichInfo, VOID *info);

Visual BASIC

VBdaqGetHardwareInfo&(ByVal handle&, ByVal whichInfo&, info As Any)

Program References

None

daqGetHardwareType

Format

```
daqGetHardwareType(handle, pHwSubType)
```

Purpose

daqGetHardwareType gets the device type and sub type of the current device

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the device type and sub type is to be obtained
pHwSubType	DaqHardwareSubType *	Pointer to a value that will contain the device sub type

Parameter Values

handle: obtained from the daqOpen function

pHwSubType: Pointer to a value that will contain the device sub type or NULL if the device sub type is not desired (see DaqHardwareSubType of the daqGetHardwareInfo function)

Returns

The device type of the current device (see DaqHardwareVersion of the daqGetHardwareInfo function)

Function Usage

This function gets the device type and sub type of the current device.

Prototypes

C/C++

daqGetHardwareType(DaqHandleT handle, DaqHardwareSubType *pHwSubType);

Visual BASIC

VBdaqGetHardwareType&(ByVal handle&, HwSubType&)

Program References

None

daqGetInfo

Format

    daqGetInfo(handle, chan, whichInfo, info)

Purpose

daqGetInfo retrieves specific information for the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device
chan	DWORD	Specifies which channel.
whichInfo	DaqInfo	Specifies what type of device information to retrieve
info	VOID	Pointer to the information returned from the selected device; data returned varies according to info type selected

Parameter Values

handle: obtained from the daqOpen function

chan: valid values range from 0 to 511

whichInfo: see table below

info: pointer to a returned value; value ranges depend on information requested

Parameter Type Definitions

whichInfo–(DaqInfo)	
Definition	**Description**
DdiHardwareVersionInfo	Returns value of type DaqHardwareVersion (see table: DdiHardwareVersionInfo)
DdiProtocolInfo	Returns value of type DaqProtocol (see table: DdiProtocolInfo)
DdiChTypeInfo	Returns channel type (DaqAdcExpType)
DdiChOptionTypeInfo	Returns channel option type (DaqAdcExpType)
DdiADminInfo	ADC Output Low Range
DdiADmaxInfo	ADC Output High Range
DdiChanCountInfo	Not Used
DdiNVRAMDateInfo	Date String
DdiNVRAMTimeInfo	Time String
DdiDbk4MaxFreqInfo	Returns the current DBK4 LPF setting
DdiDbk4SetBaselineInfo	Returns the current DBK4 baseline setting
DdiDbk4ExcitationInfo	Returns the current DBK4 excitation source setting
DdiDbk4ClockInfo	Returns the current DBK4 clock setting
DdiDbk4GainInfo	internally used by daqAdcSetScan
DdiDbk7SlopeInfo	Returns the current DBK7 slope
DdiDbk7DebounceTimeInfo	Returns the current DBK7 debounce setting
DdiDbk7MinFreqInfo	Returns the current DBK7 min frequency setting
DdiDbk7MaxFreqInfo	Returns the current DBK7 max frequency setting
DdiDbk50GainInfo	internally used by daqAdcSetScan
DdiPreTrigFreqInfo	Returns the pre-trigger sample rate (Hz)
DdiPostTrigFreqInfo	Returns the post-trigger sample rate (Hz)
DdiPreTrigPeriodInfo	Returns the pre-trigger sample period (sec)
DdiPostTrigPeriodInfo	Returns the post-trigger sample period (sec)
DdiOptNVRAMDateInfo	WaveBook parameter
DdiOptNVRAMTimeInfo	WaveBook parameter
DdiExtFeatures	Returns data flag(s) of type DaqHardwareExtFeatures (see table: DdiExtFeatures)
DdipDaqCalibrationTime	Personal Daq initial calibration period
DdiFifoSize	FIFO capacity in WORD's of data
DdiFifoCount	Count of WORD's of data currently in the FIFO

(continued)

Definition	Description
DdiSerialNumber	Serial Number String
DdiAdcClockSource	Current Clock Source
DdiFirmwareVersion	Firmware Version (String)
DdiHardwareVersion	Hardware Version (String)
DdiDriverVersion	Driver Version (String)
DdiAdcTriggerScan	Trigger Scan Number (DWORD)
DdiAdcPreTriggerCount	Amount of Pre-Trigger Scans (DWORD)
DdiAdcPostTriggerCount	Amount of Post-Trigger Scans (DWORD)
DdiSetpointStatusReg	Returns 16-bit value indicating setpoint status
DdiWbk18OpenSensorStatus	Returns opens sensor status for specified WBK18 channel
DdiWbk18OpenSensorStatus	Returns opens sensor status for specified WBK18 channel
DdiWbk18PosOverRangeStatus	Returns positive over-range status for specified WBK18 channel
DdiWbk18NegOverRangeStatus	Returns negative over-range status for specified WBK18 channel
DdiWbk18LowPassMode	Returns current Low Pass mode enumeration value
DdiWbk18LowPassCutOff	Returns current Low Pass cutoff frequency
DdiWbk18HighPassCutOff	Returns current High Pass mode enumeration value
DdiWbk18CurrentSrc	Returns current ICP current state
DdiWbk18OverRangeEnable	Returns current state of over range enable in the form a an 8 bit mask
DdiWbk18esMode	Returns current mode
DdiWbk18esFreq	Returns current frequency setting – register read
DdiWbk18esAmplitude	Returns an enumeration representing the current amplitude setting –register read
DdiWbk18esFreqCycleTime	Returns current cycle time in microseconds
DdiWbk18esFreqDurationTime	Returns current duration in microseconds
DdiWbk18OverRangeLimit	Returns current over range limit setting as a percentage.
DdiWbk18esRelay	Returns current relay position setting – register read
DdiWbk18OpenSensorStatusAll	Returns an 8 bit value indicating status for all 8 WBK18 channels
DdiWbk18PosOverRangeStatusAll	Returns an 8 bit value indicating status for all 8 WBK18 channels
DdiWbk18NegOverRangeStatusAll	Returns an 8 bit value indicating status for all 8 WBK18 channels
DdiWbk18TEDsStatus	Returns one byte of status: 0 = TEDs Transducer Found -- ,Operation complete 1 = TEDS Operation Busy 2 = Operation complete, NO TEDs Found 4 = Channel is shorted
DdiWbk18TEDsData	Returns 34 byte array of TEDs info as follows: info[0] = Status info[1] = Amount of TEDS data available info[2] = First byte of TEDS data if available info[3..33] = Rest of TEDS data
DdiWbk18TEDsDataCount	Returns the Amount of TEDS data available
DdiChSubTypeInfo	Returns Channel sub type
DdiChOptionSubTypeInfo	Returns Option sub type
DdiSetpointStatusReg	Returns a 16 bit value indicating the status for all channels

The following table lists the possible return values when DaqGetInfo is set to `DdiChTypeInfo,` or is set to `DdiChOptionTypeInfo.`

DdiChTypeInfo (Returns channel type (DaqAdcExpType))		
DdiChOptionTypeInfo (Returns channel option type (DaqAdcExpType))		
Definition	**Value**	**Device**
DaetNotDefined	0	Bank is unknown, or undefine the bank
DaetDbk50	1	DBK50/51 option
DaetDbk5	2	DBK5 option
DaetDbk2	3	DBK2 option
DaetDbk4	4	DBK4 option
DaetDbk7	5	DBK7 option
DoctWbk11	6	WBK11 SSH channel
DoctWbk12	7	WBK12 filter card
DoctWbk13	8	WBK13 filter & SSH card
DmctWbk512	9	WaveBook/512 channel
DmctWbk10	10	WBK10 channel
DmctWbk14	11	WBK14 channel
DmctWbk15	12	WBK15 channel
DmctResponseDac	13	Response DAC on WaveBook
DmctWbk16	14	WBK16 channnel
DmctWbk516	15	WaveBook/516
DmctpDaq	16	Personal Daq option
DmctWbk516_250	17	250 kHz WaveBook/516
DoctPga516	18	WaveBook/516 PGA board
DmctWbk512_10V	19	WaveBook/512 10V
DmctWbk10_10V	20	WBK10 10V
DmctWbk16_SSH	21	WBK16 channel with SSH
DmctWbk10A	22	WBK10A channel
DoctWbk12A	23	WBK12A filter card
DoctWbk13A	24	WBK13A filter & SSH card
DmctWbk17	25	WBK17 channel
DmctWbk512A	26	WaveBook/512A channel
DmctWbk516A	27	WaveBook/516A channel
DmctWbk18	28	WBK18 channel
DaetDbk90	29	DBK90 channel

The following tables give the possible return values when `DaqGetInfo` is set to `DdiHardwareVersionInfo`, `DdiProtocolInfo`, or `DdiExtFeatures`.

DdiHardwareVersionInfo	
Value Returned	**Device**
0	DaqBook100
1	DaqBook112
2	DaqBook120
3	DaqBook200 or DaqBook/260
4	DaqBook216
5	DaqBoard100
6	DaqBoard112
7	DaqBoard200
8	DaqBoard216
9	Daq112
10	Daq216
11	WaveBook512
12	WaveBook516
13	TempBook66
14	PersonalDaq56
15	WaveBook516_250
16	WaveBook512_10V
17	DaqBoard2000, DaqBoard/2000c
18	DaqBoard2001, DaqBoard/2001c
19	DaqBoard2002, DaqBoard/2002c
20	DaqBoard2003, DaqBoard/2003c
21	DaqBoard2004, DaqBoard/2004c
22	DaqBoard2005, DaqBoard/2005c
23	DaqBook2000 Series Device (A, E, or X)

DdiHardwareVersionInfo	
Value Returned	**Device**
24	Reserved
25	Reserved
26	Reserved
27	Reserved
28	Reserved
29	WaveBook/512A
30	WaveBook/516A, WaveBook/516E
31	WBK25
32	WBK40
33	WBK41
34	DaqBoard/1000
35	DaqBoard/1005
36	DaqLab/2000 Series
37	DaqScan/2000 Series
38	DaqBoard/500
39	DaqBoard/505
0x100	DaqTemp7
0x200	DaqTemp7A
0x400	DaqTemp14
0x800	DaqTemp14A

DdiProtocolInfo	
Value Returned	**Protocol**
DaqProtocolNone	Communications not established
DaqProtocol4	Standard LPT Port 4-bit mode
DaqProtocol8	Standard LPT Port 8-bit mode
DaqProtocolSMC666	SMC 37C666 EPP mode
DaqProtocolFastEPP	WBK20/21 Fast EPP mode
DaqProtocolECP	Enhanced Capability Port
DaqProtocol8BitEPP	8-bit EPP mode
DaqProtocolISA	ISA bus card DaqBoard 100/200
DaqProtocolPcCard	PCCard for Daq (PCMCIA)
DaqProtocolUSB	USB protocol (PersonalDaq)
DaqProtocolPCI	PCI bus card DaqBoard/2000 Series
DaqProtocolCPCI	Compact PCI (cPCI) bus card, cPCI DaqBoard/2000c Series
DaqProtocolTCPIP	Ethernet

DdiExtFeatures	
Value Returned	**Extended Feature Info**
Wavebook Mega-FIFO features	
DhefFifoOverflowMode	FIFO has Overflow Protection mode
DhefFifoCycleMode	FIFO has Cycle ("Finite") Mode
DhefFifoDataCount	FIFO has readable current-WORD's-of-data count
Wavebook516 features	
DhefTrigDigPattern	Can trigger on a digital pattern
DhefTrigPulseInput	Can trigger on a pulse input
DhefAcqClkExternal	Can pace acquisition to an external clock

DdiChSubTypeInfo	
Value Returned	
DaqSubTypeDefault	Default Sub type
DaqSubTypeDaqBook2000A	DaqBook/2000A or DaqBook/2000X
DaqSubTypeDaqBook2000E	DaqBook/2000E
DaqSubTypeDaqBook2001	DaqBook/2001
DaqSubTypeDaqBook2020	DaqBook/2020
DaqSubTypeDaqBook2005	DaqBook/2005
DaqSubTypeDaqLab2001	DaqLab/2001
DaqSubTypeDaqLab2005	DaqLab/2005
DaqSubTypeDaqScan2001	DaqScan/2001
DaqSubTypeDaqScan2002	DaqScan/2002
DaqSubTypeDaqScan2004	DaqScan/2004
DaqSubTypeDaqScan2005	DaqScan/2005
DaqSubTypeWaveBook516A	WaveBook/516A
DaqSubTypeWaveBook516E	WaveBook/516E
DaqSubTypeZonicBook618	ZonicBook/618
DaqSubTypeDaqBoard2000	DaqBoard/2000
DaqSubTypeDaqTemp7A	DaqTemp/7A
DaqSubTypeDaqBoard2001	DaqBoard/2001
DaqSubTypeDaqTemp14A	DaqTemp/14A
DaqSubTypeDaqBoard2005	DaqBoard/2005
DaqSubTypeDaqTemp7	DaqTemp/7
DaqSubTypeDaqTemp14	DaqTemp/14
DaqSubTypeDaqBoard3000USB	DaqBoard/3000USB
DaqSubTypeDaqBoard3001USB	DaqBoard/3001USB
DaqSubTypeDaqBoard3005USB	DaqBoard/3005USB
DaqSubTypeDaqBoard3006USB	DaqBoard/3006USB
DaqSubTypePersonalDaq3000	PersonalDaq/3000
DaqSubTypePersonalDaq3001	PersonalDaq/3001
DaqSubTypePersonalDaq3005	PersonalDaq/3005
DaqSubTypePersonalDaq3006	PersonalDaq/3006

Returns

DerrNoError   No error

Function Usage

The daqGetInfo function retrieves specific information for the device specified by the handle parameter. The device must have been opened previously to calling daqGetInfo by the daqOpen function. The values returned vary in data type. If it is not specified by the whichInfo table in the "Parameter Type Definitions" section above, the returned data type remains the same as the type it was originally set as.

When the whichInfo parameter is set to DdiHardwareVersionInfo, the result is the same as using the daqGetHardwareInfo function.

The daqGetInfo function should be used instead of daqGetHardwareInfo.

Prototypes

C/C++

daqGetInfo(DaqHandleT handle, DWORD chan, DaqInfo whichInfo, VOID *info);

Visual BASIC

VBdaqGetInfo&(ByVal handle&, ByVal chan&, ByVal whichInfo&, info As Any)

Program References

None

daqGetLastError

Format

```
daqGetLastError (handle, errCode)
```

Purpose

`daqGetLastError` retrieves the last error condition code registered by the driver**.**

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device
errCode	DaqError	Pointer to a value which holds the last returned error code

Parameter Values

handle: obtained from the `daqOpen` function
errCode: pointer to a valid error code ranging from 0 to 1,000

Returns

DerrNoError   No error

Function Usage

This function will return the last error registered by the driver against the device specified by the `handle` parameter. The last error registered against the device will be returned in the memory pointed to by the `errorCode` parameter.

For more details on error messages refer to the Daq Error Table.

Prototypes

C/C++

```
daqGetLastError(DaqHandleT handle, DaqError *errCode);
```

Visual BASIC

```
VBdaqGetLastError&(ByVal handle&, errCode&)
```

Program References

None

daqIOGet8255Conf

Format

        daqIOGet8255Conf(handle, portA, portB, portCHigh, portCLow, config)

Purpose

`daqIOGet8255Conf` sets and retrieves the configuration for the specified 8255 device with the specified port configurations.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device
portA	BOOL	8255 port A value
portB	BOOL	8255 port B value
portCHigh	BOOL	8255 port C high nibble value
portCLow	BOOL	8255 port C low nibble value
config	PDWORD	Pointer to a value representing the 8255's current configuration

Parameter Values

handle: obtained from the `daqOpen` function

portA: valid values are either `true` ($\neq 0$) or `false` ($= 0$)

portB: valid values are either `true` ($\neq 0$) or `false` ($= 0$)

partCHigh: valid values are either `true` ($\neq 0$) or `false` ($= 0$)

portCLow: valid values are either `true` ($\neq 0$) or `false` ($= 0$)

config: a pointer to the configuration value ranging from 0 to 65,535

Returns

DaqError   See Daq Error Table

For more details on error messages refer to the Daq Error Table.

Function Usage

The configuration is returned in the `config` parameter and will indicate the current configuration of the 8255. This configuration must then be written to the control register of the desired 8255 with the `daqIOWrite` function. When set to `true`, the `portA`, `portB`, `portCHigh` and `portCLow` flags will configure the respective port as an input port. If the flag is set to `false`, the port will be configured as an output.

daqSetOption will work for single byte use.

Prototypes

C/C++

```
daqIOGet8255Conf(DaqHandleT handle, BOOL portA, BOOL portB, BOOL portCHigh,
BOOL portCLow, PDWORD config);
```

Visual BASIC

```
VBdaqIOGet8255Conf&(ByVal handle&, ByVal portA&, ByVal portB&, ByVal
portCHigh&, ByVal portCLow&, config&)
```

Program References

DAQDIGIOEX01.CPP, DBK20_21EX.CPP, DBK23_24EX.CPP, DAQEX.FRM (VB)

daqIORead

Format

```
daqIORead(handle, devType, devPort, whichDevice, whichExpPort, value)
```

Purpose

`daqIORead` reads the specified port on the selected device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to perform the IO read
devType	DaqIODeviceType	IO device type
devPort	DaqIODevicePort	IO device port selection
whichDevice	DWORD	IO device instance to read from
whichExpPort	DaqIOExpansionPort	IO device expansion port to read from
value	PDWORD	Pointer to returned IO read value

Parameter Values

handle: obtained from the `daqOpen` function

devType: see table below

devPort: see table below

whichDevice: valid values range from 0 to 171

whichExpPort: see table below

value: pointer to a value ranging from 0 to 65,535

Parameter Type Definitions

devType–(DaqIODeviceType)	
Definition	**Description**
DiodtLocalBitIO	P2 – Local addressing by bit
DiodtLocal8255	P2 – Local addressing on the device
DiodtP2Local8	P2 – Local addressing by byte
DiodtP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodtP3LocalCtr16	P3 – Local addressing of 16-bit counters
DiodtP2Exp8	P2 – Expansion addressing by byte
DiodtExp8255	DBK20, DBK21
DiodtDbk23	DBK23
DiodtDbk24	DBK24
DiodtDbk25	DBK25

devPort–(DaqIODevicePort)	
Definition	**Description**
Local Bit I/O	
DiodpBitIO	P2 – Addressing by bit
P2 Sequential 8-Bit Addressing	
DiodpP2Local8	P2 – Local addressing by byte
DiodpP2LocalIR	P2 – Local Internal register (for configuring P2)
DiodpP2Exp8	P2 – Expansion adressing by byte
P3 Digital Port	
DiodpP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodpP3LocalDigIR	P3 – Local Internal register (for configuring P3)
Local 8255, Dbk20, Dbk21 (Daq Device and DBK)	
Diodp8255A	P2 – Digital byte wide Port A
Diodp8255B	P2 – Digital byte wide Port B
Diodp8255C	P2 – Digital byte wide Port C
Diodp8255IR	P2 –Internal register (for configuring P2)
Diodp8255CHigh	P2 – Digital 4-bit wide Port C
Diodp8255CLow	P2 – Digital 4-bit wide Port C
DiodpP3LocalCtr16	P3 – 16-bit Counter
Dbk23	
DiodpDbk23A	DBK23 – Digital byte wide Port A
DiodpDbk23B	DBK23 – Digital byte wide Port B
DiodpDbk23C	DBK23 – Digital byte wide Port C
DiodpDbk23Unused	Not used
Dbk24	
DiodpDbk24A	DBK24 – Digital byte wide Port A
DiodpDbk24B	DBK24 – Digital byte wide Port B
DiodpDbk24C	DBK24 – Digital byte wide Port C
DiodpDbk24Unused	Not used
DiodpDbk25	DBK25

whichExpPort–(DaqIOExpansionPort)	
Definition	**Description**
DioepP1	Note that DioepP1 is for DigiBook only.
DioepP2	DBK20/21 Port 2
DioepP3	DBK20/21 Port 3

Function Usage

The daqIORead function will return the current state of the port in the value parameter. Normally, if the selected port is a byte-wide port, the port state will occupy the low-order byte of the value parameter. Digital IO channels for the port corresponds to each bit within this low-order byte. If the bit is set, it indicates the channel is in a high state. If the bit is not set, the channel is indicated to be in a low state. This function requires that daqIOGet8255Conf or daqSetOption be called prior to invocation to configure the specifed port as an input port.

Local I/O

Those devices which support the P2 port have built-in Intel 8255C chips which can be used as general purpose I/O. The 8255C has 3 configurable DIO ports (PortA, PortB, PortC). These ports are 8-bit ports which can be individually programmed as either input or output ports. All three of the local P2 ports can be read asynchronously using the daqIORead function.

Additionally, there is a 16-bit Digital port on P3 of the main unit.* If devices allow this port to be used as a high speed digital port which can be scanned synchronously along with other analog channels in a acquisition.

Some devices, such as the DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series, DaqBoard/2000 Series [and 2000c Series] products, can also access this port asynchronously as a general purpose DIO port. If using the DaqBoard/2000 Series [or 2000c Series] products, the daqIORead function can be used to read this port asynchronously.

***Note**: P3 for DaqBoard/2000 Series boards is obtained by connecting an appropriate DBK200 Series board to the DaqBoard/2000 Series board's P4 connector, via cable.

Expansion I/O

There are several expansion options (DBK2x) that allow the DIO to be expanded on the main unit of the device. These expansion units operate off of the P2 port of the main unit (if the main unit supports P2) and can be accessed asynchronously. These expansion cards vary in numbers of DIO as well as DIO connectivity and isolation characteristics (see the User Manual on characterstics of your particular DBK2x card). All of the DBK2x series expansion cards can be accessed asynchronously.

Note! **When using a DBK20 Series expansion card on P2, the Local P2 port becomes inaccessible.**

The following table describes typical port settings. **"n" is the expansion card's position in a system.** In a three card system the first card would have n = 0, the second card n = 1, and the third card n = 2.

Digital I/O Port	devType	devPort	whichDevice	whichExpPort
P2 Local DIO (one 8255 – three 8-bit DIO banks)				
P2 Local 8255 Port A (P2 pins 30-37)	DiodtP2Local8	DiodpP2Local8	Diodp8255A	DioepP2
P2 Local 8255 Port B (P2 pins 3-10)	DiodtP2Local8	DiodpP2Local8	Diodp8255B	DioepP2
P2 Local 8255 Port C (P2 pins 22-28)	DiodtP2Local8	DiodpP2Local8	Diodp8255C	DioepP2
P3 Local DIO/HS Digital IO (one 16-bit DIO bank)				
P3 Local 16-bit Port* (P3 pins 3-10, 22-29)	DiodtP3LocalDig16	DiodpP3LocalDig16	DiodpP3LocalDig16	DioepP3
P2 Expansion DIO with DBK20/21 (dual 8255's – six 8-bit DIO banks)			**"n" is the expansion card's position in a system.**	
P2 Exp 8255-0 Port A (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255A	DioepP2
P2 Exp 8255-0 Port B (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255B	DioepP2
P2 Exp 8255-0 Port C (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255C	DioepP2
P2 Exp 8255-1 Port A (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255A + 4	DioepP2
P2 Exp 8255-1 Port B (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255B + 4	DioepP2
P2 Exp 8255-1 Port C (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255C + 4 (**see example**)	DioepP2
P2 Expansion DIO with DBK23 (three 8-bit DIO banks)			**"n" is the expansion card's position in a system.**	
P2 Exp DBK23 Port A	DiodtDbk23	DiodpP2Exp8	(*n*x4) + DiodpDbk23A	DioepP2
P2 Exp DBK23 Port B	DiodtDbk23	DiodpP2Exp8	(*n*x4) + DiodpDbk23B	DioepP2
P2 Exp DBK23 Port C	DiodtDbk23	DiodpP2Exp8	(*n*x4) + DiodpDbk23C	DioepP2
P2 Expansion DIO with DBK24 (three 8-bit DIO banks)				
P2 Exp DBK24 Port A	DiodtDbk24	DiodpP2Exp8	(*n*x4) + DiodpDbk24A	DioepP2
P2 Exp DBK24 Port B	DiodtDbk24	DiodpP2Exp8	(*n*x4) + DiodpDbk24B	DioepP2
P2 Exp DBK24 Port C	DiodtDbk24	DiodpP2Exp8	(*n*x4) + DiodpDbk24C	DioepP2
P2 Expansion DIO with DBK25 (one 8-bit DIO bank)			**"n" is the expansion card's position in a system.**	
P2 Exp DBK25 Port	DiodtDbk25	DiodpP2Exp8	*n* + DiodpDbk25	DioepP2
WBK17				
WBK17 Digital Output	DiodtWbk17	DiodpWbk17_8Bit	First channel on unit (9, 17, 25 …)	DioeP1

Example of Calculating "whichDevice" for three DBK20 P2 Exp 8255-1 Port C banks.

	whichDevice: (Note 1)	Result:
For the **first** positioned card **n= 0**. Thus for the first card's P2 Exp 8255-1 Port C ….	(*n*x8) + Diodp8255B + 4 = n = 0	Diodp8255B + 4
For the **second** positioned card **n= 1**. Thus for the second card's P2 Exp 8255-1 Port C ….	(*n*x8) + Diodp8255B + 4 = n = 1	Diodp8255B + 12
For the **third** positioned card **n= 2**. Thus for the third card's P2 Exp 8255-1 Port C ….	(*n*x8) + Diodp8255B + 4 = n = 2	Diodp8255B + 20

Note 1: The equation is from the *whichDevice* column and "P2 Exp 8255-1" row (shaded), in the preceding table.

Returns

DerrNoError   No error

Prototypes

C/C++

```
daqIORead(DaqHandleT handle, DaqIODeviceType devType, DaqIODevicePort
devPort, DWORD whichDevice, DaqIOExpansionPort whichExpPort, PDWORD value);
```

Visual BASIC

```
VBdaqIORead&(ByVal handle&, ByVal devType&, ByVal devPort&, ByVal
whichDevice&, ByVal whichExpPort&, value&)
```

Program References

DAQDIGIOEX01.CPP, DAQDIGIOEX02.CPP, DBK20_21EX.CPP, DBK23_24EX.Cpp,
DBK25EX.CPP, DAQEX.FRM (VB)

daqIOReadBit

Also See: **daqIORead, daqIOWrite, daqIOWriteBit**

Format

```
daqIOReadBit(handle, devType, devPort, whichDevice, whichExpPort, bitNum,
bitValue)
```

Purpose

daqIOReadBit reads a specified bit on the selected device and port.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which to perform the IO
devType	DaqIODeviceType	IO device type
devPort	DaqIODevicePort	IO device port selection
whichDevice	DWORD	IO device selection
whichExpPort	DaqIOExpansionPort	IO expansion port address
bitNum	DWORD	IO port bit location to read
bitValue	PBOOL	IO port bit value (true – high, false – low)

Parameter Values

handle: obtained from the daqOpen function

devType: see table below

devPort: see table below

whichDevice: valid values range from 0 to 171

whichExpPort: see table below

bitNum: valid values range from 1 to 16

bitValue: valid values are either true ($\neq 0$) or false ($= 0$)

Parameter Type Definitions

devType-(DaqIODeviceType)	
Definition	**Description**
DiodtLocalBitIO	P2 – Local addressing by bit
DiodtLocal8255	P2 – Local addressing on device
DiodtP2Local8	P2 – Local addressing by byte
DiodtP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodtP3LocalCtr16	P3 – Local addressing of 16-bit counters
DiodtP2Exp8	P2 – Expansion addressing by byte
DiodtExp8255	DBK20, DBK21
DiodtDbk23	DBK23
DiodtDbk24	DBK24
DiodtDbk25	DBK25

devPort-(DaqIODevicePort)	
Definition	**Description**
Local Bit I/O	
`DiodpBitIO`	P2 – Addressing by bit
P2 Sequential 8-Bit Addressing	
`DiodpP2Local8`	P2 – Local addressing by byte
`DiodpP2LocalIR`	P2 – Local Internal register (for configuring P2)
`DiodpP2Exp8`	P2 – Expansion adressing by byte
P3 Digital Port	
`DiodpP3LocalDig16`	P3 – Local addressing for HS 16-bit Dig I/O
`DiodpP3LocalDigIR`	P3 – Local Internal register (for configuring P3)
Local 8255, Dbk20, Dbk21 (Daq device and DBK)	
`Diodp8255A`	P2 – Digital byte wide Port A
`Diodp8255B`	P2 – Digital byte wide Port B
`Diodp8255C`	P2 – Digital byte wide Port C
`Diodp8255IR`	P2 –Internal register (for configuring P2)
`Diodp8255CHigh`	P2 – Digital 4-bit wide Port C
`Diodp8255CLow`	P2 – Digital 4-bit wide Port C
`DiodpP3LocalCtr16`	P3 – 16-bit Counter
Dbk23	
`DiodpDbk23A`	DBK23 – Digital byte wide Port A
`DiodpDbk23B`	DBK23 – Digital byte wide Port B
`DiodpDbk23C`	DBK23 – Digital byte wide Port C
`DiodpDbk23Unused`	Not used
Dbk24	
`DiodpDbk24A`	DBK24 – Digital byte wide Port A
`DiodpDbk24B`	DBK24 – Digital byte wide Port B
`DiodpDbk24C`	DBK24 – Digital byte wide Port C
`DiodpDbk24Unused`	Not used
`DiodpDbk25`	DBK25

whichExpPort-(DaqIOExpansionPort)	
Definition	**Description**
`DioepP1`	Note that `DioepP1` is for DigiBook only.
`DioepP2`	DBK20/21 Port 2
`DioepP3`	DBK20/21 Port 3

Returns

`DaqError` See Daq Error Table

For more details on error messages refer to the Daq Error Table.

Function Usage

The `daqIOReadBit` function will return the current state of the selected bit in the `bitValue` parameter. The selected bit (specified by the `bitNum` parameter) corresponds to the input/output (I/O) channel on the port which is to be read. The `bitValue` will be `true` indicating a high state or `false` indicating a low state. This function requires that `daqIOGet8255Conf` or `daqSetOption` be called prior to invocation to configure the specifed port as an input port.

Local I/O

Those devices which support the P2 port have built-in Intel 8255C chips which can be used as general purpose I/O. The 8255C has 3 configurable digital input/output (DIO) ports (PortA, PortB, PortC). These ports are 8-bit ports which can be individually programmed as either input or output ports. All three of the local P2 ports can be read asynchronously using the `daqIOReadBit` function.

Additionally, there is a 16-bit Digital port on P3 of the main unit.* If devices allow this port to be used as a high speed digital port which can be scanned synchronously along with other analog channels in a acquisition.

Some devices, such as the DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series, DaqBoard/2000 Series [and 2000c Series] products, can also access this port asynchronously as a general purpose DIO port. If using the DaqBoard/2000 Series [or 2000c Series] products, the `daqIORead` function can be used to read this port asynchronously.

*Note: P3 for DaqBoard/2000 Series boards is obtained by connecting an appropriate DBK200 Series board to the DaqBoard/2000 Series board's P4 connector, via cable.

Expansion I/O

There are several expansion options (DBK2x) that allow the DIO to be expanded on the main unit of the device. These expansion units operate off of the P2 port of the main unit (if the main unit supports P2) and can be accessed asynchronously. These expansion cards vary in numbers of DIO as well as DIO connectivity and isolation characteristics (see the User Manual on characterstics of your particular DBK2x card). All of the DBK2x series expansion cards can be accessed asynchronously.

When using a DBK20 Series expansion card on P2, the Local P2 port becomes inaccessible.

The following table describes typical port settings. **"n" is the expansion card's position in a system.** In a three card system the first card would have n = 0, the second card n = 1, and the third card n = 2.

Digital I/O Port	`devType`	`devPort`	`whichDevice`	`whichExpPort`
P2 Local DIO (one 8255 – three 8-bit DIO banks)				
P2 Local 8255 Port A (P2 pins 30-37)	`DiodtP2Local8`	`DiodpP2Local8`	`Diodp8255A`	`DioepP2`
P2 Local 8255 Port B (P2 pins 3-10)	`DiodtP2Local8`	`DiodpP2Local8`	`Diodp8255B`	`DioepP2`
P2 Local 8255 Port C (P2 pins 22-28)	`DiodtP2Local8`	`DiodpP2Local8`	`Diodp8255C`	`DioepP2`
P3 Local DIO/HS Digital IO (one 16-bit DIO bank)				
P3 Local 16-bit Port* (P3 pins 3-10, 22-29)	`DiodtP3LocalDig16`	`DiodpP3LocalDig16`	`DiodpP3LocalDig16`	`DioepP3`
P2 Expansion DIO with DBK20/21 (dual 8255's – six 8-bit DIO banks) **"n" is the expansion card's position in a system.**				
P2 Exp 8255-0 Port A (see DBK20/21 doc)	`DiodtExp8255`	`DiodpP2Exp8`	`(nx8) + Diodp8255A`	`DioepP2`
P2 Exp 8255-0 Port B (see DBK20/21 doc)	`DiodtExp8255`	`DiodpP2Exp8`	`(nx8) + Diodp8255B`	`DioepP2`
P2 Exp 8255-0 Port C (see DBK20/21 doc)	`DiodtExp8255`	`DiodpP2Exp8`	`(nx8) + Diodp8255C`	`DioepP2`
P2 Exp 8255-1 Port A (see DBK20/21 doc)	`DiodtExp8255`	`DiodpP2Exp8`	`(nx8) + Diodp8255A + 4`	`DioepP2`
P2 Exp 8255-1 Port B (see DBK20/21 doc)	`DiodtExp8255`	`DiodpP2Exp8`	`(nx8) + Diodp8255B + 4`	`DioepP2`
P2 Exp 8255-1 Port C (see DBK20/21 doc)	`DiodtExp8255`	`DiodpP2Exp8`	`(nx8) + Diodp8255C + 4` (**see example**)	`DioepP2`
P2 Expansion DIO with DBK23 (three 8-bit DIO banks) **"n" is the expansion card's position in a system.**				
P2 Exp DBK23 Port A	`DiodtDbk23`	`DiodpP2Exp8`	`(nx4) + DiodpDbk23A`	`DioepP2`
P2 Exp DBK23 Port B	`DiodtDbk23`	`DiodpP2Exp8`	`(nx4) + DiodpDbk23B`	`DioepP2`
P2 Exp DBK23 Port C	`DiodtDbk23`	`DiodpP2Exp8`	`(nx4) + DiodpDbk23C`	`DioepP2`
P2 Expansion DIO with DBK24 (three 8-bit DIO banks)				
P2 Exp DBK24 Port A	`DiodtDbk24`	`DiodpP2Exp8`	`(nx4) + DiodpDbk24A`	`DioepP2`
P2 Exp DBK24 Port B	`DiodtDbk24`	`DiodpP2Exp8`	`(nx4) + DiodpDbk24B`	`DioepP2`
P2 Exp DBK24 Port C	`DiodtDbk24`	`DiodpP2Exp8`	`(nx4) + DiodpDbk24C`	`DioepP2`
P2 Expansion DIO with DBK25 (one 8-bit DIO bank) **"n" is the expansion card's position in a system.**				
P2 Exp DBK25 Port	`DiodtDbk25`	`DiodpP2Exp8`	`n + DiodpDbk25`	`DioepP2`
WBK17				
WBK17 Digital Output	`DiodtWbk17`	`DiodpWbk17_8Bit`	First channel on unit (9, 17, 25 …)	`DioeP1`

Example of Calculating "whichDevice" for three DBK20 P2 Exp 8255-1 Port C banks.

	whichDevice: (Note 1)	Result:
For the **first** positioned card **n= 0**. Thus for the first card's P2 Exp 8255-1 Port C ….	$(n \times 8)$ + Diodp8255B + 4 = n = 0	Diodp8255B + 4
For the **second** positioned card **n= 1**. Thus for the second card's P2 Exp 8255-1 Port C ….	$(n \times 8)$ + Diodp8255B + 4 = n = 1	Diodp8255B + 12
For the **third** positioned card **n= 2**. Thus for the third card's P2 Exp 8255-1 Port C ….	$(n \times 8)$ + Diodp8255B + 4 = n = 2	Diodp8255B + 20

Note 1: The equation is from the *whichDevice* column and "P2 Exp 8255-1" row (shaded), in the preceding table.

Prototypes

C/C++

```
daqIOReadBit(DaqHandleT handle, DaqIODeviceType devType, DaqIODevicePort
devPort, DWORD whichDevice, DaqIOExpansionPort whichExpPort, DWORD bitNum,
PBOOL bitValue);
```

Visual BASIC

```
VBdaqIOReadBit&(ByVal handle&, ByVal devType&, ByVal devPort&, ByVal
whichDevice&, ByVal whichExpPort&, ByVal bitNum&, bitValue&)
```

Program References

```
DAQEX.FRM (VB)
```

daqIOWrite

Also See: **daqIORead, daqIOReadBit, daqIOWriteBit**

.

Format

```
daqIOWrite(handle, devType, devPort, whichDevice, whichExpPort, value)
```

Purpose

daqIOwrite writes to the specified port on the selected device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to perform the IO write
devType	DaqIODeviceType	IO device type
devPort	DaqIODevicePort	IO device port selection
whichDevice	DWORD	IO device instance to write from
whichExpPort	DaqIOExpansionPort	IO device expansion port to write from
value	DWORD	Pointer to IO value to write

Parameter Values

handle: obtained from the daqOpen function

devType: see table below

devPort: see table below

whichDevice: valid values range from 0 to 171

whichExpPort: see table below

value: valid values range from 0 to 65,535

Parameter Type Definitions

devType–(DaqIODeviceType)	
Definition	**Description**
DiodtLocalBitIO	P2 – Local addressing by bit
DiodtLocal8255	P2 – Local addressing on DaqBook/2000 Series, DaqBoard/2000 Series and cPCI DaqBoard/2000c Series Devices
DiodtP2Local8	P2 – Local addressing by byte
DiodtP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodtP3LocalCtr16	P3 – Local addressing of 16-bit counters
DiodtP2Exp8	P2 – Expansion addressing by byte
DiodtExp8255	DBK20, DBK21
DiodtDbk23	DBK23
DiodtDbk24	DBK24
DiodtDbk25	DBK25
DiodtWbk17	WBK17

devPort–(DaqIODevicePort)	
Definition	**Description**
Local Bit I/O	
DiodpBitIO	P2 – Addressing by bit
P2 Sequential 8-Bit Addressing	
DiodpP2Local8	P2 – Local addressing by byte
DiodpP2LocalIR	P2 – Local Internal register (for configuring P2)
DiodpP2Exp8	P2 – Expansion adressing by byte
P3 Digital Port	
DiodpP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodpP3LocalDigIR	P3 – Local Internal register (for configuring P3)
Local 8255, Dbk20, Dbk21 (Daq device and DBK only)	
Diodp8255A	P2 – Digital byte wide Port A
Diodp8255B	P2 – Digital byte wide Port B
Diodp8255C	P2 – Digital byte wide Port C
Diodp8255IR	P2 –Internal register (for configuring P2)
Diodp8255CHigh	P2 – Digital 4-bit wide Port C
Diodp8255CLow	P2 – Digital 4-bit wide Port C
DiodpP3LocalCtr16	P3 – 16-bit Counter
Dbk23	
DiodpDbk23A	DBK23 – Digital byte wide Port A
DiodpDbk23B	DBK23 – Digital byte wide Port B
DiodpDbk23C	DBK23 – Digital byte wide Port C
DiodpDbk23Unused	Not used
Dbk24	
DiodpDbk24A	DBK24 – Digital byte wide Port A
DiodpDbk24B	DBK24 – Digital byte wide Port B
DiodpDbk24C	DBK24 – Digital byte wide Port C
DiodpDbk24Unused	Not used
DiodpDbk25	DBK25
Wbk17	
DiodpWbk17_8Bit	WBK17 – 8-bit digital output

whichExpPort–(DaqIOExpansionPort)	
Definition	**Description**
DioepP1	Note that DioepP1 is only for DigiBook and WaveBook applications.
DioepP2	DBK20/21 Port 2
DioepP3	DBK20/21 Port 3

Returns

DaqError   See Daq Error Table

For more details on error messages refer to the Daq Error Table.

Function Usage

The daqIOWrite function will output to the port the bit battern represented by the value parameter. Normally, if the selected port is a byte-wide port, the port state will occupy the low-order byte of the value parameter. Digital I/O channels for the port corresponds to each bit within this low-order byte. If the bit is set, it indicates the channel is in a high state. If the bit is not set, the channel is indicated to be in a low state. This function requires that daqIOGet8255Conf or daqSetOption be called prior to invocation to configure the specifed port as an output port.

Local I/O

Those devices which support the P2 port have built-in Intel 8255C chips which can be used as general purpose I/O. The 8255C has 3 configurable DIO ports (PortA, PortB, PortC). These ports are 8-bit ports which can be individually programmed as either input or output ports. All three of the local P2 ports can be written asynchronously using the daqIOWrite function.

Additionally, there is a 16-bit Digital port on P3 of the main unit.* If P3 is supported, the DaqBook/100 Series, DaqBook/200 Series, Daq PC-Card, ISA-type DaqBoard, DaqBook/2000 Series, DaqBoard/2000 Series*, and cPCI DaqBoard/2000c Series devices allow this port to be used as a high speed digital port which can be scanned synchronously along with other analog channels in a acquisition.

Some devices, such as the DaqBook/2000 Series, DaqBoard/2000 Series [and 2000c Series] products, can also access this port asynchronously as a general purpose DIO port. If using the DaqBoard/2000 Series [or 2000c Series] products, the daqIORead function can be used to read this port asynchronously.

***Note**: P3 for DaqBoard/2000 Series boards is obtained by connecting an appropriate DBK200 Series board to the DaqBoard/2000 Series board's P4 connector, via cable.

Expansion I/O

There are several options that allow for DIO expansion. These expansion options, referred to as the DBK20 Series cards, operate off of the main unit's P2 port and can be accesssed asynchronously. The expansion cards vary in number of DIO, connectivity, and isolation characteristics. Refer to the *DBK Cards and Options User's Manual* (p/n 457-0905) in regard to the specifics of your particular DBK20 Series card.

When using a DBK20 Series expansion card on P2, the Local P2 port becomes inaccessible.

The following table describes typical port settings. **"n" is the expansion card's position in a system.** In a three card system the first card would have n = 0, the second card n = 1, and the third card n = 2.

Digital I/O Port	devType	devPort	whichDevice	whichExpPort
P2 Local DIO (one 8255 – three 8-bit DIO banks)				
P2 Local 8255 Port A (P2 pins 30-37)	DiodtP2Local8	DiodpP2Local8	Diodp8255A	DioepP2
P2 Local 8255 Port B (P2 pins 3-10)	DiodtP2Local8	DiodpP2Local8	Diodp8255B	DioepP2
P2 Local 8255 Port C (P2 pins 22-28)	DiodtP2Local8	DiodpP2Local8	Diodp8255C	DioepP2
P3 Local DIO/HS Digital IO (one 16-bit DIO bank)				
P3 Local 16-bit Port* (P3 pins 3-10, 22-29)	DiodtP3LocalDig16	DiodpP3LocalDig16	DiodpP3LocalDig16	DioepP3
P2 Expansion DIO with DBK20/21 (dual 8255's – six 8-bit DIO banks)			**"n" is the expansion card's position in a system.**	
P2 Exp 8255-0 Port A (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255A	DioepP2
P2 Exp 8255-0 Port B (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255B	DioepP2
P2 Exp 8255-0 Port C (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255C	DioepP2
P2 Exp 8255-1 Port A (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255A + 4	DioepP2
P2 Exp 8255-1 Port B (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255B + 4	DioepP2
P2 Exp 8255-1 Port C (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(*n*x8) + Diodp8255C + 4 (**see example**)	DioepP2
P2 Expansion DIO with DBK23 (three 8-bit DIO banks)			**"n" is the expansion card's position in a system.**	
P2 Exp DBK23 Port A	DiodtDbk23	DiodpP2Exp8	(*n*x4) + DiodpDbk23A	DioepP2
P2 Exp DBK23 Port B	DiodtDbk23	DiodpP2Exp8	(*n*x4) + DiodpDbk23B	DioepP2
P2 Exp DBK23 Port C	DiodtDbk23	DiodpP2Exp8	(*n*x4) + DiodpDbk23C	DioepP2
P2 Expansion DIO with DBK24 (three 8-bit DIO banks)				
P2 Exp DBK24 Port A	DiodtDbk24	DiodpP2Exp8	(*n*x4) + DiodpDbk24A	DioepP2
P2 Exp DBK24 Port B	DiodtDbk24	DiodpP2Exp8	(*n*x4) + DiodpDbk24B	DioepP2
P2 Exp DBK24 Port C	DiodtDbk24	DiodpP2Exp8	(*n*x4) + DiodpDbk24C	DioepP2
P2 Expansion DIO with DBK25 (one 8-bit DIO bank)			**"n" is the expansion card's position in a system.**	
P2 Exp DBK25 Port	DiodtDbk25	DiodpP2Exp8	*n* + DiodpDbk25	DioepP2
WBK17				
WBK17 Digital Output	DiodtWbk17	DiodpWbk17_8Bit	First channel on unit (9, 17, 25 …)	DioeP1

Example of Calculating "whichDevice" for three DBK20 P2 Exp 8255-1 Port C banks.

	whichDevice: (Note 1)	Result:
For the **first** positioned card **n= 0**. Thus for the first card's P2 Exp 8255-1 Port C ….	$(n \times 8)$ + Diodp8255B + 4 = n = 0	Diodp8255B + 4
For the **second** positioned card **n= 1**. Thus for the second card's P2 Exp 8255-1 Port C ….	$(n \times 8)$ + Diodp8255B + 4 = n = 1	Diodp8255B + 12
For the **third** positioned card **n= 2**. Thus for the third card's P2 Exp 8255-1 Port C ….	$(n \times 8)$ + Diodp8255B + 4 = n = 2	Diodp8255B + 20

Note 1: The equation is from the *whichDevice* column and "P2 Exp 8255-1" row (shaded), in the preceding table.

Prototypes

C/C++

```
daqIOWrite(DaqHandleT handle, DaqIODeviceType devType, DaqIODevicePort
devPort, DWORD whichDevice, DaqIOExpansionPort whichExpPort, DWORD value);
```

Visual BASIC

```
VBdaqIOWrite&(ByVal handle&, ByVal devType&, ByVal devPort&, ByVal
whichDevice&, ByVal whichExpPort&, ByVal value&)
```

Program References

```
DAQDIGIOEX01.CPP, DBK20_21EX.CPP, DBK23_24EX.CPP, DAQEX.FRM (VB)
```

daqIOWriteBit

Format

```
daqIOWriteBit(handle, devType, devPort, whichDevice, whichExpPort, bitNum,
bitValue)
```

Purpose

daqIOWriteBit writes a specified bit on the selected device and port.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle of the device to perform an IO write to
devType	DaqIODeviceType	IO device type
devPort	DaqIODevicePort	IO device port selection
whichDevice	DWORD	IO device selection
whichExpPort	DaqIOExpansionPort	IO device expansion port address
bitNum	DWORD	Bit number on port to write
bitValue	BOOL	Bit value to write (true – high, false – low)

Parameter Values

handle: obtained from the daqOpen function

devType: see table below

devPort: see table below

whichDevice: valid values range from 0 to 171

whichExpPort: see table below

bitNum: valid value range from 1 to 16

bitValue: valid values are either true ($\neq 0$) or false ($= 0$)

Parameter Type Definitions

devType-(DaqIODeviceType)	
Definition	**Description**
DiodtLocalBitIO	P2 – Local addressing by bit
DiodtLocal8255	P2 – Local addressing on DaqBook/2000 Series, DaqBoard/2000 Series and cPCI DaqBoard/2000c Series devices
DiodtP2Local8	P2 – Local addressing by byte
DiodtP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodtP3LocalCtr16	P3 – Local addressing of 16-bit counters
DiodtP2Exp8	P2 – Expansion addressing by byte
DiodtExp8255	DBK20, DBK21
DiodtDbk23	DBK23
DiodtDbk24	DBK24
DiodtDbk25	DBK25

devPort-(DaqIODevicePort)	
Definition	**Description**
Local Bit I/O	
DiodpBitIO	P2 – Addressing by bit
P2 Sequential 8-Bit Addressing	
DiodpP2Local8	P2 – Local addressing by byte
DiodpP2LocalIR	P2 – Local Internal register (for configuring P2)
DiodpP2Exp8	P2 – Expansion adressing by byte
P3 Digital Port	
DiodpP3LocalDig16	P3 – Local addressing for HS 16-bit Dig I/O
DiodpP3LocalDigIR	P3 – Local Internal register (for configuring P3)
Local 8255, Dbk20, Dbk21 (Daq device and DBK only)	
Diodp8255A	P2 – Digital byte wide Port A
Diodp8255B	P2 – Digital byte wide Port B
Diodp8255C	P2 – Digital byte wide Port C
Diodp8255IR	P2 –Internal register (for configuring P2)
Diodp8255CHigh	P2 – Digital 4-bit wide Port C
Diodp8255CLow	P2 – Digital 4-bit wide Port C
DiodpP3LocalCtr16	P3 – 16-bit Counter
Dbk23	
DiodpDbk23A	DBK23 – Digital byte wide Port A
DiodpDbk23B	DBK23 – Digital byte wide Port B
DiodpDbk23C	DBK23 – Digital byte wide Port C
DiodpDbk23Unused	Not used
Dbk24	
DiodpDbk24A	DBK24 – Digital byte wide Port A
DiodpDbk24B	DBK24 – Digital byte wide Port B
DiodpDbk24C	DBK24 – Digital byte wide Port C
DiodpDbk24Unused	Not used
DiodpDbk25	DBK25

whichExpPort–(DaqIOExpansionPort)	
Definition	**Description**
DioepP1	Note that DioepP1 is for DigiBook only.
DioepP2	DBK20/21 Port 2
DioepP3	DBK20/21 Port 3

Returns

DaqError   See Daq Error Table

Function Usage

The daqIOWriteBit function will set the current state of the selected bit in the bitValue parameter. The selected bit (specified by the bitNum parameter) corresponds to the I/O channel on the port which is being written to. The bitValue can be set to true indicating a high state or false indicating a low state. This function requires that daqIOGet8255Conf or daqSetOption be called prior to invocation to configure the specifed port as an output port.

Local I/O

Those devices which support the P2 port have built-in Intel 8255C chips which can be used as general purpose I/O. The 8255C has 3 configurable DIO ports (PortA, PortB, PortC). These ports are 8-bit ports which can be individually programmed as either input or output ports. All three of the local P2 ports can be programmed asynchronously using the daqIOWriteBit function.

Additionally, there is a 16-bit Digital port on P3 of the main unit.* If devices allow this port to be used as a high speed digital port which can be scanned synchronously along with other analog channels in a acquisition.

*Note: P3 for DaqBoard/2000 Series boards is obtained by connecting an appropriate DBK200 Series board to the DaqBoard/2000 Series board's P4 connector, via cable.

Some devices, such as the DaqBook/2000 Series, DaqLab/2000 Series, DaqScan/2000 Series, DaqBoard/2000 Series [and 2000c Series] products, can also access this port asynchronously as a general purpose DIO port. If using the DaqBoard/2000 Series [or 2000c Series] products, the `daqIORead` function can be used to read this port asynchronously.

Expansion I/O

There are several expansion options (DBK2x) that allow the DIO to be expanded on the main unit of the device. These expansion units operate off of the P2 port of the main unit (if the main unit supports P2) and can be accessed asynchronously. These expansion cards vary in numbers of DIO as well as DIO connectivity and isolation characteristics (see the User Manual on characterstics of your particular DBK2x card). All of the DBK2x series expansion cards can be accessed asynchronously.

When using a DBK20 Series expansion card on P2, the Local P2 port becomes inaccessible.

The following table describes typical port settings. **"n" is the expansion card's position in a system.** In a three card system the first card would have n = 0, the second card n = 1, and the third card n = 2.

Digital I/O Port	devType	devPort	whichDevice	whichExpPort
P2 Local DIO (one 8255 – three 8-bit DIO banks)				
P2 Local 8255 Port A (P2 pins 30-37)	DiodtP2Local8	DiodpP2Local8	Diodp8255A	DioepP2
P2 Local 8255 Port B (P2 pins 3-10)	DiodtP2Local8	DiodpP2Local8	Diodp8255B	DioepP2
P2 Local 8255 Port C (P2 pins 22-28)	DiodtP2Local8	DiodpP2Local8	Diodp8255C	DioepP2
P3 Local DIO/HS Digital IO (one 16-bit DIO bank)				
P3 Local 16-bit Port* (P3 pins 3-10, 22-29)	DiodtP3LocalDig16	DiodpP3LocalDig16	DiodpP3LocalDig16	DioepP3
P2 Expansion DIO with DBK20/21 (dual 8255's – six 8-bit DIO banks)			**"n" is the expansion card's position in a system.**	
P2 Exp 8255-0 Port A (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(nx8) + Diodp8255A	DioepP2
P2 Exp 8255-0 Port B (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(nx8) + Diodp8255B	DioepP2
P2 Exp 8255-0 Port C (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(nx8) + Diodp8255C	DioepP2
P2 Exp 8255-1 Port A (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(nx8) + Diodp8255A + 4	DioepP2
P2 Exp 8255-1 Port B (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(nx8) + Diodp8255B + 4	DioepP2
P2 Exp 8255-1 Port C (see DBK20/21 doc)	DiodtExp8255	DiodpP2Exp8	(nx8) + Diodp8255C + 4 (**see example**)	DioepP2
P2 Expansion DIO with DBK23 (three 8-bit DIO banks)			**"n" is the expansion card's position in a system.**	
P2 Exp DBK23 Port A	DiodtDbk23	DiodpP2Exp8	(nx4) + DiodpDbk23A	DioepP2
P2 Exp DBK23 Port B	DiodtDbk23	DiodpP2Exp8	(nx4) + DiodpDbk23B	DioepP2
P2 Exp DBK23 Port C	DiodtDbk23	DiodpP2Exp8	(nx4) + DiodpDbk23C	DioepP2
P2 Expansion DIO with DBK24 (three 8-bit DIO banks)				
P2 Exp DBK24 Port A	DiodtDbk24	DiodpP2Exp8	(nx4) + DiodpDbk24A	DioepP2
P2 Exp DBK24 Port B	DiodtDbk24	DiodpP2Exp8	(nx4) + DiodpDbk24B	DioepP2
P2 Exp DBK24 Port C	DiodtDbk24	DiodpP2Exp8	(nx4) + DiodpDbk24C	DioepP2
P2 Expansion DIO with DBK25 (one 8-bit DIO bank)			**"n" is the expansion card's position in a system.**	
P2 Exp DBK25 Port	DiodtDbk25	DiodpP2Exp8	n + DiodpDbk25	DioepP2
WBK17				
WBK17 Digital Output	DiodtWbk17	DiodpWbk17_8Bit	First channel on unit (9, 17, 25 …)	DioeP1

Example of Calculating "whichDevice" for three DBK20 P2 Exp 8255-1 Port C banks.

	whichDevice: (Note 1)	**Result**:
For the **first** positioned card **n= 0**. Thus for the first card's P2 Exp 8255-1 Port C ….	(nx8) + Diodp8255B + 4 = n = 0	Diodp8255B + 4
For the **second** positioned card **n= 1**. Thus for the second card's P2 Exp 8255-1 Port C ….	(nx8) + Diodp8255B + 4 = n = 1	Diodp8255B + 12
For the **third** positioned card **n= 2**. Thus for the third card's P2 Exp 8255-1 Port C ….	(nx8) + Diodp8255B + 4 = n = 2	Diodp8255B + 20

Note 1: The equation is from the *whichDevice* column and "P2 Exp 8255-1" row (shaded), in the preceding table.

Prototypes

C/C++

```
daqIOWriteBit(DaqHandleT handle, DaqIODeviceType devType, DaqIODevicePort
devPort, DWORD whichDevice, DaqIOExpansionPort whichExpPort, DWORD bitNum,
BOOL bitValue);
```

Visual BASIC

```
VBdaqIOWriteBit&(ByVal handle&, ByVal devType&, ByVal devPort&, ByVal
whichDevice&, ByVal whichExpPort&, ByVal bitNum&, ByVal bitValue&)
```

Program References

```
DAQEX.FRM (VB)
```

daqOnline

Format

daqOnline(handle, online)

Purpose

daqOnline determines if a device is online.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle of the device to test for online
online	PBOOL	Boolean indicating whether the device is currently online

Parameter Values

handle: obtained from the daqOpen function

online: a pointer to a boolean value; values are either true ($\neq 0$) or false ($= 0$)

Returns

DerrNoError   No error

Function Usage

The handle parameter for this function must be a valid device handle which has been opened using the daqOpen function. The online parameter indicates the current online state of the device (true = device online; false = device not online).

Prototypes

C/C++

daqOnline(DaqHandleT handle, PBOOL online);

Visual BASIC

VBdaqOnline&(ByVal handle&, online&)

Program References

None

daqOpen

Format

```
daqOpen(LPSTR daqName)
```

Purpose

daqOpen opens an installed device for operation.

Parameter Summary

Parameter	Type	Description
daqName	LPSTR	String representing the name of the device to be opened.

Parameter Values

daqName: a pointer to a string of characters—no effective range of values applies

Returns

handle   A handle to the specified device (-1 if open failed).

For more details on error messages refer to the Daq Error Table.

Function Usage

The daqOpen function will initiate a session for the device name specified by the daqName parameter by opening the device, initializing it, and preparing it for further operation. The daqName specified must reference a currently configured device. Refer to **Daq Configuration** utility sections of your user's manual (on the CD-ROM) if needed. An example of assigning a new Device Name follows shortly.

Example of Accessing and Using the Daq Configuration Control Panel Applet
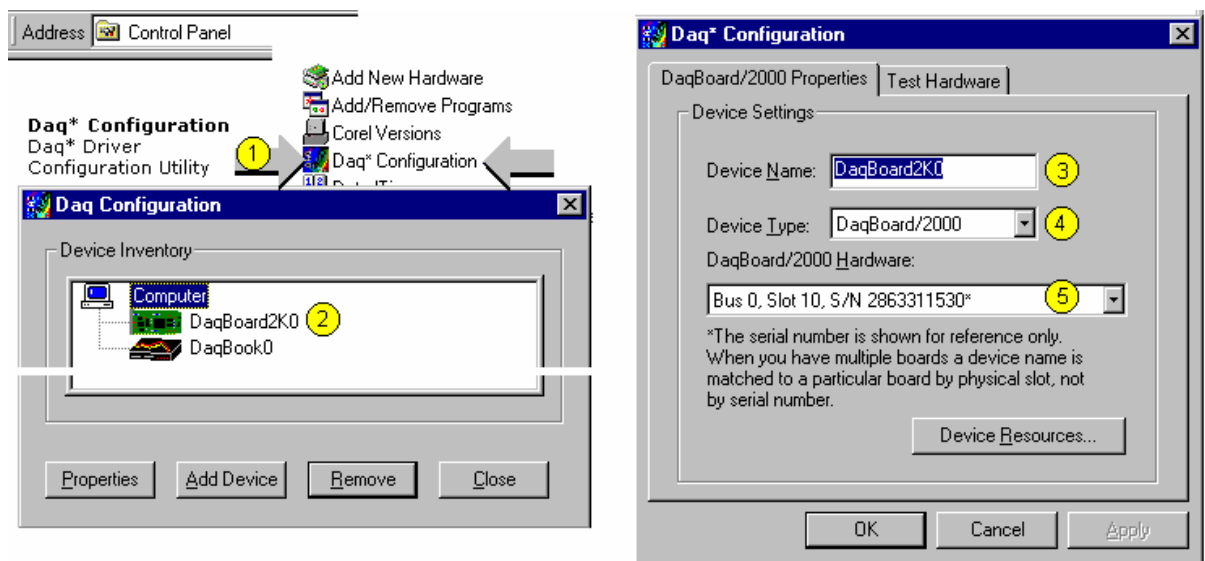
To access the **Daq Configuration** applet and change its device name:

- Run the **Daq Configuration** control panel applet. Navigation from the desktop to the applet is as follows:
  Start ⇒ Settings ⇒ Control Panel ⇒ Daq Configuration
- Double-click on the Device Inventory's DaqBoard2K0 icon. The **DaqBoard/2000 Properties** tab will appear.

- Enter a device name in the text box, or use the default "DaqBoard2K0." **Device Name** is for identifying the specific DaqBoard/2000 Series board [or cPCI DaqBoard/2000c Series board]. Note that **Device Name** actually refers to the PCI[cPCI] slot and not to the actual DaqBoard/2000 Series board[or cPCI DaqBoard/2000c].

> **Programming Note:**
> It should be noted that the daqName parameter of daqOpen should correspond to the name of the device presented here. In this case, "DaqBoard2K0" should be used as the daqName to open the DaqBoard/2000 Series board [or cPCI DaqBoard/2000c] represented here. Notice the device named "DaqBook0" directly under the "DaqBoard2k0" device. To open the this device the daqName parameter of daqOpen should be set to "DaqBook0"

- Verify "DaqBoard/2000**"** is listed as the **Device Type.** Note that available device types can be viewed via the pull-down list.
- Confirm that the DaqBoard/2000 text box shows a Bus #, Slot #, and Serial Number.
- Verify that the DaqBoard/2000's serial number matches the serial number obtained in step 5.

> Steps 5 and 6 apply only to DaqBoard/2000 Series [and 2000c Series] devices.

Obtaining a Device's `handle`

daqOpen should be performed prior to any other operation performed on the device. This function will return a device handle that is used by other functions to reference the device. Once the device has been opened, the device handle should be used to perform subsequent operations on the device. If successful, this function will return a non-negative handle which can then be used in subsequent API calls.

Most functions in this manual require a device handle in order to perform their operation. When the device session is complete, daqClose may be called with the device handle to close the device session.

> If a -1 is returned as the handle then daqOpen function failed to properly open the device. If this occurs, the **returned handle is not valid and cannot be used** in any other handle based function. The -1 handle constitutes a fatal error condition and the device cannot be further accessed.

Prototypes

C/C++

```
daqOpen (LPSTR daqName);
```

Visual BASIC

```
VBdaqOpen& (ByVal daqName$)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQDIGIOEX01.CPP, DAQDIGIOEX02.CPP,
DAQEX.FRM (VB)
```

daqProcessError

Format

```
daqProcessError (handle, errCode)
```

Purpose

`daqProcessError` initiates an error for processing by the driver.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the specified error is to be processed
errCode	DaqError	Pointer to a value which specifies the device error code to process

Parameter Values

handle: obtained from the `daqOpen` function
errCode: a pointer to a value ranging from 0 to 1,000

Returns

For more details on error messages refer to the Daq Error Table.

Function Usage

The `daqProcessError` function can be used to initiate processing for a device-defined error.

Prototypes

C/C++

```
daqProcessError(DaqHandleT handle, DaqError errCode);
```

Visual BASIC

```
VBdaqProcessError&(ByVal handle&, ByVal errCode&)
```

Program References

```
None
```

daqReadCalFile

Format

    daqReadCalFile (handle, calfile)

Purpose

daqReadCalFile is the initialization function for reading in the calibration constants from the calibration text file.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device which will be associated with the calibration file
calfile	LPSTR	File name with optional path information of the calibration file

Parameter Values

handle: obtained from the daqOpen function

calfile: a pointer to a string of characters; if the value of calfile is null or empty (""), the default calibration file DAQBOOK.CAL will be read.

Returns

DerrInvCalfile   Error occurred while opening or reading calibration file

DerrNoError   No error

For more details on error messages refer to the Daq Error Table.

Function Usage

The daqReadCalFile function (usually called once at the beginning of a program) will read all the calibration constants from the specified file. The calfile parameter specifies the path\filename of the calibration file to read. If calibration constants for a specific channel number and gain setting are not contained in the file, ideal calibration constants will be used—essentially performing no calibration for that channel. If an error occurs while trying to open the calibration file, ideal calibration constants will be used for all channels and a non-zero error code will be returned by the daqReadCalFile function.

Prototypes

C/C++

    daqReadCalFile(DaqHandleT handle, LPSTR calfile);

Visual BASIC

    VBdaqReadCalFile&(ByVal handle&, ByVal calfile$)

Program References

    DBK19EX.CPP, DBK52EX.CPP

daqSetDefaultErrorHandler

Format

```
daqSetDefaultErrorHandler(handler)
```

Purpose

`daqSetDefaultErrorHandler` sets the driver to use the default error handler specified for all devices.

Parameter Summary

Parameter	Type	Description
handler	DaqErrorHandlerFPT	Pointer to a user-defined error handler function.

Parameter Values

`handler`: a pointer to a user-defined function

Returns

`DerrNoError`   No error

Function Usage

The `daqSetDefaultErrorHandler` function allows you to set the driver to use a default error handler specified by the `DaqErrorHandleFPT`. The `DaqErrorHandleFPT` parameter should point to the function defined by the application that will be used to process the error codes passed to it. This parameter should be set prior to calling the function. This function can also be used to disable on-screen error reporting by setting the `DaqErrorHandleFPT` to `null`(0).

   For more details on error messages refer to the Daq Error Table.

Prototypes

C/C++

```
daqSetDefaultErrorHandler(DaqErrorHandlerFPT handler);
```

Visual BASIC

```
VBdaqSetDefaultErrorHandler&(ByVal handler&)
```

Program References

```
None
```

daqSetErrorHandler

Format

vdaqSetErrorHandler (handle, handler)

Purpose

daqSetErrorHandler specifies the routine to call when an error occurs in any function for the specified device**.**

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to which to attach the specified error handler
handler	DaqErrorHandlerFPT	Pointer to a user-defined error handler function

Parameter Values

handle: obtained from the daqOpen function
handler: a pointer to a user-defined function

Returns

DerrNoError   No error

Function Usage

The daqSetErrorHandler function specifies an error handler for the device currently open with the handle parameter. It should be used if it is desirable to use a error handler other than the default error handler for a specific device. This function allows the application to specify its own routine to be called when errors occur on processing commands for the device. If it is desirable to have no action occur when a command error is detected on the device, use this function with a null (0) parameter. The default error routine is daqDefaultErrorHandler.

For more details on error messages refer to the Daq Error Table.

Prototypes

C/C++

daqSetErrorHandler(DaqHandleT handle, DaqErrorHandlerFPT handler);

Visual BASIC

VBdaqSetErrorHandler&(ByVal handle&, ByVal handler&)

Program References

DAQADCEX05.CPP, DAQADCEX06.CPP, DAQEX.FRM (VB)

daqSetOption

Format

```
daqSetOption (handle, chan, flags, optionType, optionValue)
```

Purpose

daqSetOption allows the setting of options for a device's channel/signal path configuration.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	The handle to the device for which to set the option
chan	DWORD	The channel number on the device for which the option is to be set
flags	DWORD	Flags specifying the options to use
optionType	DaqOptionType	Specifies the type of option
optionValue	FLOAT	The value of the option to set

Parameter Values

handle: obtained from the daqOpen function.

chan: valid values range from 0 to 271

flags: see table below

optionType: see table below

optionValue: values available depend on optionType parameter --see optionType table below

Parameter Type Definitions

flags	
Definition	**Description**
DcofChannel	Apply option to the channel specified by the chan parameter.
DcofModule	Apply option to the entire module for which chan is located.
DcofSubChannelLow *(For WBK17 Only)*	Counter Low Word. SubChannel identifier for WBK17 Detection Options. Use enum + detection number (1 through 16). Note that WBK17 Detector Options require sub-channel in DaqChanOptionFLagType. Some Option Types have enumerated Option Values (DaqChanOptionValue).
DcofSubChannelHigh *(For WBK17 Only)*	Counter High Word. SubChannel identifier for WBK17 Detection Options. Use enum + detection number (1 through 16). Note that WBK17 Detector Options require sub-channel in DaqChanOptionFLagType. Some Option Types have enumerated Option Values (DaqChanOptionValue).

optionType–DaqOptionType			
DBK 4 Options			
Option Type (optionType)	**Description**	**Option Value (optionValue)**	**Description**
DdcotDbk4MaxFreq	Set the DBK4 Low Pass Filter (LPF) Frequency	DcovDbk4Freq18kHz	LPF 3dB level is 18MHz
		DcovDbk4Freq9000Hz	LPF 3dB level is 9MHz
		DcovDbk4Freq4500Hz	LPF 3dB level is 4.5MHz
		DcovDbk4Freq2250Hz	LPF 3dB level is 2.25MHz
		DcovDbk4Freq1125Hz	LPF 3dB level is 1125Hz
		DcovDbk4Freq563Hz	LPF 3dB level is 563Hz
		DcovDbk4Freq281Hz	LPF 3dB level is 281Hz
		DcovDbk4Freq141Hz	LPF 3dB level is 141Hz
DdcotDbk4SetBaseline	Set the DBK4 Baseline usage	DcovDbk4BaselineNever	Baseline is not set when configuring the scan group (see daqAdcSetScan)
		DcovDbk4BaselineOneShot	Baseline is set when configuring the next scan group (see daqAdcSetScan)

(continued)

(DBK4 Continued)

Option Type (optionType)	Description	Option Value (optionValue)	Description
DdcotDbk4Excitation	Enable/Disable the excitation source on the DBK4	TRUE	Enables (turn on) excitation source
		FALSE	Disables (turn off) excitation source
DdcotDbk4Clock	Enable/Disable the Switched Capacitor Clock on the DBK4	TRUE	Enables Switched Capacitor Clock (must be enabled if using filter)
		FALSE	Disables Switched Capacitor Clock
DdcotDbk4Gain	For Internal Use	N/A	N/A

DBK 7 Options			
DdcotDbk7Slope DdcotDbk55Slope	Set DBK7/DBK5555 channel to count on Rising/Falling edge	TRUE	Count on Rising Edge of Signal
		FALSE	Count on Falling Edge of Signal
DdcotDbk7DebounceTime DdcotDbk55DebounceTime	Set the DBK7/DBK55 signal Debounce Setting	DcovDbk7DeboucneNone	No debouncing
		DcovDbk7Debounce600us	600μs Debounce time
		DcovDbk7Debounce2500us	2500μs Debounce time
		DcovDbk7Debounce10ms	10ms Debounce time
DdcotDbk7MinFreq DdcotDbk55MinFreq	Set the DBK7/DBK55 minimum measured frequency	0-1,000,000	Set the DBK7/DBK55 minimum measured frequency to the value specified
DdcotDbk7MaxFreq DdcotDbk55MaxFreq	Set the DBK7/DBK55 maximum measured frequency	0-1,000,000	Set the DBK7/DBK55 minimum measured frequency to the value specified

DBK 50 Options			
DdcotDbk50Gain	For Internal Use	N/A	N/A

DBK 90 Options			
DdcotDbk90StartChanOn	Set Channel to DBK90		
DdcotDbk90StartChanOffAll	Clear all DBK90 channels		
DdcotDbk90StartChanOff	Clear Channel to DBK90		
Option Type DmotDbk90GlitchReject	= 2 Used to reject high magnitude changes in the data stream.		
Option Values DmovDbk90GlitchRejectOff	= 0 Turns glitch rejection off.		
DmovDbk90GlitchRejectOn	= 1 Turns glitch rejection on.		

DBK 100 Options
Note: DBK100 devices make use of the DBK90 Option Types and Values. See preceding DBK90 section.

optionType-DaqOptionType continued			
Option Type **(optionType)**	**Description**	**Option Value** **(optionValue)**	**Description**
Main Unit Options			
DbotFifoOverflowMode	Enable/Disable FIFO flushing upon overrun of the FIFO (used w/WBK30)	TRUE	Upon FIFO overrun condition the acquisition will terminate but the FIFO will NOT be flushed until all the data is transferred out of the FIFO
		FALSE	Upon FIFO overrun condition the acquisition will terminate and the FIFO will be immediately flushed. Once flushed no data may be retrieved from the FIFO.
DbotFifoCycleMode	Enables/Disables overwriting of data within the FIFO. (used w/WBK30)	TRUE	Allows data to be continuously acquired to the FIFO without transferring data from the FIFO until the acquisition has been completed. This mode is useful when it is desirable to take a pre-defined amount of pre-trigger data.
		FALSE	Will not allow the old FIFO data to be overwritten. This mode requires that either the entire acquisition be acquired within the FIFO or that data in the FIFO be continuously transferred to the PC. This mode is useful when collecting a non-pre-triggered acquisition of unknown or infinite length or a pre-trigger acquisition of unknown or infinite length. This mode requires that the application continuously transfer data from the FIFO.
DbotFifoCycleSize	Cycle buffer length in 16-bit WORDs (used w/WBK30)	1-2,684,354,550	Specifies the amount of the FIFO to use for the specified FIFO operation. This value can never exceed the memory size of the WBK30 module. If using pre-trigger with the DbotFifoCycleMode mode the value should be: pre-trigger size + post-trigger size
DbotFifoFlush	Flush all data in the FIFO now (used w/WBK30)	N/A	Executes an immediate flushing of the WBK30 FIFO. All data will be removed from the FIFO and will no longer be available for transfer
DbotFifoNoFlushOnDisarm	Disable Buffer Flushing upon Disarm (used w/WBK30)	TRUE	Specifies that the data is to remain in the FIFO after the acquisition has been completed or the acquisition has been disarmed. The FIFO data will be available for transfer following a disarm operation.
		FALSE	Specifies that the data is to be flushed from the FIFO after the acquisition has been has been disarmed by the application. The FIFO data will NOT be available for transfer following a disarm operation.

*Option Type and Value Definitions Continued--***DaqOptionType**			
Digital I/O, Counter and Timer Options			
Option Type (`optionType`)	**Description**	**Option Value** (`optionValue`)	**Description**
`DcotP2Local8Mode`	Set Input/Output mode for 8-bit Local P2 port channel	`DcovDigitalInput`	Set the 8-bit local P2 port channel to be an input channel
		`DcovDigitalOutput`	Set the 8-bit local P2 port channel to be an output channel
`DcotP2Exp8Mode`	Set Input/Output mode for 8-bit Expansion P2 port channel	`DcovDigitalInput`	Set the 8-bit expansion P2 port channel to be an input channel
		`DcovDigitalOutput`	Set the 8-bit expansion P2 port channel to be an output channel
`DcotP3Local16Mode`	Set Input/Output mode for 16-bit Local P3 port	`DcovDigitalInput`	Set the 16-bit local P3 port to be an input channel
		`DcovDigitalOutput`	Set the 16-bit local P3 port to be an output channel
`DcotCounterCascade`	Enables/Disables cascading counter channels on DaqBook/2000 Series, DaqBoard/2000 Series, and cPCI DaqBoard/2000c Series.	`DcovCounterCascade`	Enable cascading of 16-bit counters into 32-bit counters
		`DcovCounterSingle`	Use single 16-bit counters – do not cascade counters.
`DcotCounterMode`	Enables/Disables Clear on Read of counter channel on DaqBook/2000 Series, DaqBoard/2000 Series and cPCI DaqBoard/2000c Series.	`DcovCounterClearOnRead`	Enable clear on read of the selected counter(s). When counter channels are included in the acquisition scanning and pulse counting is required this mode must be set
		`DcovCounterTotalize`	Disables clear on read of the selected counter(s). This mode allows counters to free-run in a totalizing mode of operation.
`DcotCounterControl`	Turn Counter Channel On/Off or manually clear counter w/ DaqBook/2000 Series, DaqBoard/2000 Series [and /2000c Series] products. This mode should only be used when reading counters asynchronously	`DcovCounterOn`	Enable counting by turning the counter channel On
		`DcovCounterOff`	Disable counting by turning the counter channel Off
		`DcovCounterManualClear`	Manually clear the counter channel. Counter will continue to count if still enabled.
`DmotCounterControl`	Turn ALL Counter Channels On/Off or manually clear ALL Counter Channels w/ DaqBook/2000 Series, DaqBoard/2000 Series [and /2000c Series] product. This mode should only be used when reading counters asynchronously	`DcovCounterOn`	Enable counting for ALL counter channels by turning the counter channels On
		`DcovCounterOff`	Disable counting for ALL counter channels by turning the counter channels Off
		`DcovCounterManualClear`	Manually clear the ALL counter channels. Counters will continue to count if still enabled.
`DbotOverSampleAmount`	Used to enable and set the sample value for, or disable, oversampling (averaging) of data before data is delivered to the buffer.	`0 thru 13` (for 14 sample settings 2, 4, 8, 16, 32, 64, 128, etc. up to 16384)	0,1 disables oversampling. Number of samples 2, 4, 8, 16, 32, 64 … 16384; where 0 sets 2, 1 sets 4, 2 sets 8 . . . 13 sets 16,384 samples.
`DcotCounterEdge`	Selects Counter Edge Detection (Rising/Falling) for DaqBook/2000 Series, DaqBoard/2000 Series [and /2000c Series] products	`TRUE`	Increment Counter on Rising Edge of input signal
		`FALSE`	Increment Counter on Falling Edge of input signal
`DcotTimerDivisor`	16-bit Number (freq = 1MHz / (Divisor + 1))	`0-65,535`	16-bit value divides the 1MHz clock by 1 to 65535
`DcotTimerControl`	Turn Timer Channel On/Off w/ DaqBook/2000 Series, DaqBoard/2000 Series [and /2000c Series] products.	`DcovTimerOn`	Enables Timer Channel Output by turning the Timer Channel On.
		`DcovTimerOff`	Disables Timer Channel Output by turning the Timer Channel Off.
`DmotTimerControl`	Turn ALL Timer Channel(s) On/Off w/ DaqBook/2000 Series, DaqBoard/2000 Series [and /2000c Series] products.	`DcovTimerOn`	Enables ALL Timer Channel Outputs by turning all the Timer Channels On.
		`DcovTimerOff`	Disables ALL Timer Channel Outputs by turning all the Timer Channels Off.

*Option Type and Value Definitions Continued--***DaqOptionType**		
3000 Series Device Options		
Option Type (`optionType`)	**Option Value** (`optionValue`)	**Description**
DcotCounterEnhDebounceTime Used to bypass the debounce mode, or to set a channel's comparator output to one of 16 debounce times. Debounce is used to eliminate switch-induced transients typically associated with electro-mechanical devices including relays, proximity switches, and encoders. Note that there are two debounce modes, one for "After Stable" and another for "Before Stable." See: `DcotCounterEnhDebounceTrigger`		Sets debounce to:
	`DcovCounterEnhDebounce500ns` = 0	500 ns
	`DcovCounterEnhDebounce1500ns` = 1	1500 ns
	`DcovCounterEnhDebounce3500ns` = 2	3500 ns
	`DcovCounterEnhDebounce7500ns` = 3	7500 ns
	`DcovCounterEnhDebounce15500ns` = 4	15500 ns
	`DcovCounterEnhDebounce31500ns` = 5	31500 ns
	`DcovCounterEnhDebounce63500ns` = 6	63500 ns
	`DcovCounterEnhDebounce127500ns` = 7	127500 ns
	`DcovCounterEnhDebounce100us` = 8	100 µs
	`DcovCounterEnhDebounce300us` = 9	300 µs
	`DcovCounterEnhDebounce700us` = 10	700 µs
	`DcovCounterEnhDebounce1500us` = 11	1500 µs
	`DcovCounterEnhDebounce3100us` = 12	3100 µs
	`DcovCounterEnhDebounce6300us` = 13	6300 µs
	`DcovCounterEnhDebounce12700us` = 14	12700 µs
	`DcovCounterEnhDebounce25500us` = 15	25500 µs
	`DcovCounterEnhDebounceNone` = 16	bypass
DcotCounterEnhEdge Determines whether the rising edge or falling edge is to be detected.		Selects for detection:
	`DcovCounterEnhRisingEdge` = 0	The "Rising Edge"
	`DcovCounterEnhFallingEdge` = 1	The "Falling Edge"
DcotCounterEnhTickSize Determines the ticksize; which is the fundamental unit of time for period, pulsewidth, and timing measurements.		Sets ticksize to:
	`DcovCounterEnhTick20_83ns` = 0	20.83 ns
	`DcovCounterEnhTick208_3ns` = 1	208.3 ns
	`DcovCounterEnhTick2083_3ns` = 2	2 083.3 ns
	`DcovCounterEnhTick20833_3ns` = 3	20833.3 ns
DcotCounterEnhControl	`DcovCounterEnhDisable` = 0	Disables counter
	`DcovCounterEnhEnable` = 1	Enables counter
	`DcovCounterEnhClear` = 2	Clears counter
DmotCounterEnhControl	`DcovCounterEnhDisable` = 0	Disables all counters
	`DcovCounterEnhEnable` = 1	Enables all counters
	`DcovCounterEnhClear` = 2	Clears all counters

Usage Note: Combine the applicable mode with the desired *mode-specific* settings via option values.
The following three option values are available for all Option Types listed above.

```
DcovCounterEnhMode_Counter
DcovCounterEnh_ClearOnRead
DcovCounterEnhModeMask_32Bit
```

3000 Series Device Options *(continued)*
Option Type (`optionType`)

`DcotCounterEnhDebounceTrigger`

Sets the mode of the debounce module to Trigger After Stable, or to Trigger Before Stable.

Option Value (optionValue)

`DcovCounterEnhTriggerAfterStable = 0`

`DcovCounterEnhTriggerAfterStable` selects the "Trigger After Stable" mode. This mode rejects glitches and only passes state transitions after a specified period of stability (the debounce time). This mode is used with electro-mechanical devices like encoders and mechanical switches to reject switch bounce and disturbances due to a vibrating encoder that is not otherwise moving. The debounce time should be set short enough to accept the desired input pulse but longer than the period of the undesired disturbance.

Option Value (optionValue)

`DcovCounterEnhTriggerBeforeStable  = 1`

`DcovCounterEnhTriggerBeforeStable` selects the Trigger Before Stable mode. Use this mode when the input signal has groups of glitches and each group is to be counted as one. The trigger before stable mode will recognize and count the first glitch within a group but reject the subsequent glitches within the group if the debounce time is set accordingly. In this case the debounce time should be set to encompass one entire group of glitches.

3000 Series Device Options		
Option Type (`optionType`)	**Option Value (`optionValue`)**	**Description**

`DcotCounterEnhMapChannel`
 	
Used to select the mapped channel.
 	
For the 3000 Series devices, a mapped channel is one of 4 signals that can get multiplexed into a channel's counter module.
 	
The mapped channel can participate with the channel's input signal by gating the counter, clearing the counter, etc.
	`DcovCounterEnhMap_Channel_0` = 0
	`DcovCounterEnhMap_Channel_1` = 1
	`DcovCounterEnhMap_Channel_2` = 2
	`DcovCounterEnhMap_Channel_3` = 3
		Selects the mapped channel to be one of the counter input channels.
		
		There are 4 post-debounce channel input signals that can be individually selected as mapped channels.
		
		These are:
		    Channel_0
		    Channel_1
		    Channel_2
		    Channel_3

Option Type and Value Definitions Continued--**DaqOptionType**		
3000 Series Device Options *(continued)*		
Option Type **(optionType)**	**Option Value** **(optionValue)**	**Description**
DcotCounterEnhMeasurementMode		

	DcovCounterEnhMode_Counter	**Counter Mode**
Enumeration with Bit-Masking. The options can be combined.	DcovCounterEnhCounter_Totalize	**Totalize Mode** – The counter counts up and rolls over on the 16-bit or 32-bit boundary.
	DcovCounterEnhCounter_ClearOnRead	**Clear On Read Mode -** The counter is cleared at the beginning of every scan; and the final value of the counter [the value just before it was cleared] is latched and returned to the Wavebook/516.
	DcovCounterEnhCounter_RollOver	**Rollover Mode** - The counter continues to count upward, rolling over on the 16-bit or 32-bit boundary.
	DcovCounterEnhCounter_StopOnTop	**Stop at the Top Mode** - The counter will stop at the top of its count. The top of the count is FFFF for the 16-bit option and FFFFFFFF for the 32-bit option.
	DcovCounterEnhCounter_LatchOnSOS	Selects **start of scan**. Latches the counter outputs at the beginning of every scan.
	DcovCounterEnhCounter_LatchOnMap	Selects the **mapped signal** to latch the counter outputs. This allows the user to know the exact counter value when an edge is present on another channel.
	DcovCounterEnhCounter_DecrementOff	Sets the counter decrement option to "off."
	DcovCounterEnhCounter_DecrementOn	Sets the counter decrement option to "on."
	DcovCounterEnhCounter_CountChan	Selects channel for count.
	DcovCounterEnhCounter_CountMap	Selects mapped channel for count.
	DcovCounterEnhModeMask_16Bit	Selects 16-Bit counter.
	DcovCounterEnhModeMask_32Bit	Selects 32-Bit counter.
	DcovCounterEnhModeMaskGatingOff	Selects gating "off."
	DcovCounterEnhModeMaskGatingOn	Selects gating "on." When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).
	DcovCounterEnhMode_OFF	Turns the Enhanced Counter Mode "off."
	DcovCounterEnhMode_Counter	Turns the Enhanced Counter Mode "on."

DcotCounterEnhMeasurementMode   *(continued)*

Option Type and Value Definitions Continued--`DaqOptionType`		
3000 Series Device Options *(continued)*		
Option Type **(`optionType`)**	**Option Value** **(`optionValue`)**	**Description**
`DcotCounterEnhMeasurementMode` *(continued)*		

	`DcovCounterEnhMode_Period`	**Period Mode**

Enumeration with Bit-Masking.
 	
 The options can be combined.
	`DcovCounterEnhPeriod_X1`
	 `DcovCounterEnhPeriod_X10`
	 `DcovCounterEnhPeriod_X100`
	 `DcovCounterEnhPeriod_X1000`
		Used to select the number of periods to time, per measurement. Choices are:
		 1, 10, 100, or 1000

	`DcovCounterEnhPeriod_MeasChan`
	
	 `DcovCounterEnhPeriod_MeasMap`
		Selects to measure input channel's period.
		
		 Selects to measure the mapped channel's period.

	`DcovCounterEnhModeMask_16Bit` `DcovCounterEnhModeMask_32Bit`	Selects 16-Bit counter. Selects 32-Bit counter.

	`DcovCounterEnhModeMaskGatingOff`
	
	 `DcovCounterEnhModeMaskGatingOn`
		Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

	`DcovCounterEnhMode_PulseWidth`	**Pulsewidth Mode**
	`DcovCounterEnhPulseWidth_MeasChan`	Selects to measure input channel's pulsewidth.
	`DcovCounterEnhPulseWidth_MeasMap`	Selects to measure the mapped channel's pulsewidth.
	`DcovCounterEnhModeMask_16Bit`	Selects 16-Bit counter.
	`DcovCounterEnhModeMask_32Bit`	Selects 32-Bit counter.

	`DcovCounterEnhModeMaskGatingOff`
	
	 `DcovCounterEnhModeMaskGatingOn`
		Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

	`DcovCounterEnhMode_Timing`	**Timing Mode**
	`DcovCounterEnhModeMask_16Bit`	Selects 16-Bit counter.
	`DcovCounterEnhModeMask_32Bit`	Selects 32-Bit counter.

	`DcovCounterEnhModeMaskGatingOff`
	
	 `DcovCounterEnhModeMaskGatingOn`
		Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

DcotCounterEnhMeasurementMode *(continued)*

3000 Series Device Options *(continued)*		
Option Type **(optionType)**	**Option Value** **(optionValue)**	**Description**
DcotCounterEnhMeasurementMode *(continued)*		

	DcovCounterEnhMode_Encoder	**Encoder Mode**
Enumeration with Bit-Masking. The options can be combined.	`DcovCounterEnhEncoder_X1`	Determines the encoder measurement mode: 1X, 2X, or 4X.
	`DcovCounterEnhEncoder_X2`	
	`DcovCounterEnhEncoder_X4`	
	`DcovCounterEnhEncoder_LatchOnSOS`	Selects **start of scan**. Latches the counter outputs at the beginning of every scan.
	`DcovCounterEnhEncoder_LatchOnZ`	Selects the Encoder Z **mapped signal** to latch the counter outputs. This allows the user to know the exact counter value when an edge is present on another channel.
	`DcovCounterEnhEncoder_ClearOnZ_Off`	Selects "clear on Z" On or Off. When On, the encoder Z is referenced to clear the counter. The counter is cleared on the rising edge of the mapped (Z) channel.
	`DcovCounterEnhEncoder_ClearOnZ_On`	
	`DcovCounterEnhModeMask_16Bit`	Selects 16-Bit counter.
	`DcovCounterEnhModeMask_32Bit`	Selects 32-Bit counter.
	`DcovCounterEnhModeMaskGatingOff`	Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).
	`DcovCounterEnhModeMaskGatingOn`	

*Option Type and Value Definitions Continued--***DaqOptionType**			
WaveBook/516 and WBK12/13 Options			
Option Type **(optionType)**	**Description**	**Option Value** **(optionValue)**	**Description**
`DcotPga516LowPassMode`	Turn on or bypass the Low Pass Filter on the WaveBook/516 PGA	`DcovPga516LowPassBypass`	Bypass the Low Pass Filter on the WaveBook/516 PGA
		`DcovPga516LowPassOn`	Turn on the Low Pass Filter on the WaveBook/516 PGA
`DcotWbk12FilterCutOff`	Set the cutoff frequency for the WBK12 option	Values range from 400 Hz to 100KHz	Sets the cutoff frequency somewhere between specified range
`DcotWbk12FilterType`	Set the Filter Type for the WBK12 option	`DcovWbk12FilterElliptic`	Set the filter type for the WBK12 to be Elliptic
		`DcovWbk12FilterLinear`	Set the filter type for the WBK12 to be Linear
`DcotWbk12FilterMode`	Set the filter mode on the WBK12 option	`DcovWbk12FilterBypass`	Bypass the WBK12 Filter
		`DcovWbk12FilterOn`	Turn on the WBK12 Filter
`DcotWbk12PreFilterMode`	Set the pre-filter mode for the WBK12 option.	`DcovWbk12PreFilterDefault`	Leave pre-filter mode on
		`DcovWbk12PreFilterOff`	Turn off pre-filter mode
`DcotWbk13FilterCutOff`	Set the cutoff frequency for the WBK13 option	Values range from 400 Hz to 100KHz	Sets the cutoff frequency somewhere between specified range
`DcotWbk13FilterType`	Set the Filter Type for the WBK13 option	`DcovWbk13FilterElliptic`	Set the filter type for the WBK13 to be Elliptic
		`DcovWbk13FilterLinear`	Set the filter type for the WBK13 to be Linear
`DcotWbk13FilterMode`	Set the filter mode on the WBK13 option	`DcovWbk13FilterBypass`	Bypass the WBK13 Filter
		`DcovWbk13FilterOn`	Turn on the WBK13 Filter
`DcotWbk13PreFilterMode`	Set the pre-filter mode for the WBK13 option.	`DcovWbk13PreFilterDefault`	Leave pre-filter mode on
		`DcovWbk13PreFilterOff`	Turn off pre-filter mode

Option Type and Value Definitions Continued--**DaqOptionType**			
WBK14 Options			
Option Type (`optionType`)	**Description**	**Option Value** (`optionValue`)	**Description**
`DcotWbk14LowPassMode`	Turn on, bypass or set to ext clock the Low Pass Filter (LPF) on a WBK14 channel	`DcovWbk14LowPassBypass`	Bypass the Low Pass Filter for the specified WBK14 channel
		`DcovWbk14LowPassOn`	Turn on the Low Pass Filter for the specified WBK14 channel
		`DcovWbk14LowPassExtClk`	Use External Clock to control the cutoff frequency for the Low Pass Filter for the specified WBK14 channel
`DcotWbk14LowPassCutOff`	Set the Low Pass Filter (LPF) cutoff frequency for a WBK14 channel	`1.0 to 100000.0 (Hz)`	Set the Low Pass Filter frequency (when LPF set to `DcovWbk14LowPassOn`) For the specified WBK14 channel
`DcotWbk14HighPassCutOff`	Set the High Pass Filter (HPF) cutoff frequency for a WBK14 channel	`DcovWbk14HighPass0_1Hz`	Set the High Pass Filter (HPF) frequency to 0.1 Hz for the specified WBK14 channel
		`DcovWbk14HighPass10Hz`	Set the High Pass Filter (HPF) frequency to 10.0 Hz for the specified WBK14 channel
`DcotWbk14CurrentSrc`	Set the current source for the WBK14 channel	`DcovWbk14CurrentSrcOff`	Turn off the current source for the specified WBK14 channel
		`DcovWbk14CurrentSrc2mA`	Set the current source to 2mA for the specified WBK14 channel
		`DcovWbk14CurrentSrc4mA`	Set the current source to 4mA for the specified WBK14 channel
`DcotWbk14PreFilterMode`	Set the pre-filter mode for a WBK14 channel	`DcovWbk14PreFilterDefault`	Use the built-in default 65 dB pre-filter stage
		`DcovWbk12PreFilterOff`	Turn the pre-filter off
`DmotWbk14ExcSrcWaveform`	Set the Excitation Source waveform for a WBK14	`DmovWbk14ExcSrcRandom`	Use random signal waveform generation for the excitation waveform
		`DmovWbk14ExcSrcSine`	Use Sine wave signal waveform generation for the excitation waveform
`DmotWbk14ExcSrcFreq`	Set the Excitation Source frequency for a WBK14	`1.0 to 550000.0 (Hz)`	Sets the excitation source waveform frequency
`DmotWbk14ExcSrcAmplitude`	Set the Excitation Source amplitude for a WBK14	`0.0 to 5.0 (Volts)`	Sets the excitation source waveform amplitude
`DmotWbk14ExcSrcOffset`	Set the Excitation Source offset for a WBK14	`-5.0 to 5.0 (Volts)`	Sets the excitation source signal offset
`DmotWbk14ExcSrcApply`		`0.0 to 5.0 (Volts)`	Apply excitation voltage
`DcotWbk14ExtFilterRange`		`DcovWbk14FilterRange_1K`	Sets the filter range for the WBK14 channel to 1Khz
		`DcovWbk14FilterRange_5K`	Sets the filter range for the WBK14 channel to 5Khz
		`DcovWbk14FilterRange_10K`	Sets the filter range for the WBK14 channel to 10Khz
		`DcovWbk14FilterRange_15K`	Sets the filter range for the WBK14 channel to 15Khz
		`DcovWbk14FilterRange_20K`	Sets the filter range for the WBK14 channel to 20Khz

<table>
<tr><td colspan="4">Option Type and Value Definitions -- <code>DaqOptionType</code></td></tr>
<tr><td colspan="4">WBK16 Options</td></tr>
<tr><th>Option Type
(<code>optionType</code>)</th><th>Description</th><th>Option Value
(<code>optionValue</code>)</th><th>Description</th></tr>
<tr><td rowspan="2"><code>DcotWbk16Bridge</code></td><td rowspan="2">Selects the type of bridge configuration.

For diagrams and related text, refer to the WaveBook User's Manual (p/n 481-0901).</td><td><code>DcovWbk16ApplyFull</code></td><td>Selects Full Bridge</td></tr>
<tr><td><code>DcovWbk16ApplyHalfQtrPos</code></td><td>Selects Half-Bridge/ Quarter-Bridge</td></tr>
<tr><td rowspan="5"><code>DcotWbk16ShuntCal</code></td><td rowspan="5">Used to select the shunt value for internal shunt-calibration resistors.   The shunt resistors are not pre-installed.

In regard to shunt setup, refer to the WaveBook User's Manual (p/n 481-0901).</td><td><code>DcovWbk16ApplyNone</code></td><td>normal reading, no shunt</td></tr>
<tr><td><code>DcovWbk16Apply120</code></td><td>Apply 120 ohm shunt – B Shunts are not pre-installed.</td></tr>
<tr><td><code>DcovWbk16Apply350</code></td><td>Apply 350 ohm shunt – D Shunts are not pre-installed.</td></tr>
<tr><td><code>DcovWbk16Apply1K</code></td><td>Apply 1 K ohm shunt – F Shunts are not pre-installed.</td></tr>
<tr><td><code>DcovWbk16AutoZero</code></td><td>apply short (0 ohm)</td></tr>
<tr><td><code>DcotWbk16InDiag</code></td><td>Sets input diagnostics.</td><td><code>DcovWbk16ReadNone</code></td><td>Sets diagnostics for reading input signals.</td></tr>
<tr><td><code>DcotWbk16OffsetDac</code></td><td>Sets the Dac Offset relative to input.</td><td>0.0 to 4095.0   See Note 2.</td><td>See Note 2 at end of WBK16 Table.</td></tr>
<tr><td rowspan="2"><code>DcotWbk16Inv</code></td><td rowspan="2">Used to select normal signal polarity, or to invert the polarity, if desired.</td><td><code>DcovWbk16Normal</code></td><td>Normal signal polarity.</td></tr>
<tr><td><code>DcovWbk16Inverted</code></td><td>Inverts signal polarity.</td></tr>
<tr><td rowspan="3"><code>DcotWbk16FilterType</code></td><td rowspan="3">Used to bypass the single-pole low-pass filter (LPF), or set it to one of two cut-off frequencies: 10 Hz to reduce high frequency noise; or 1K Hz for anti-aliasing & slight noise rejection while maintaining moderate bandwidth.</td><td><code>DcovWbk16FltBypass</code></td><td>Bypasses the filter</td></tr>
<tr><td><code>DcovWbk16Flt10Hz</code></td><td>Enables 10 Hz filter</td></tr>
<tr><td><code>DcovWbk16Flt1Khz</code></td><td>Enables 1 K Hz filter</td></tr>
<tr><td rowspan="2"><code>DcotWbk16Couple</code></td><td rowspan="2">Sets the input coupling. Use AC coupling to reject unwanted DC offsets. Use DC coupling when both AC and DC components are to be presented to the comparator as input.</td><td><code>DcovWbk16CoupleDC</code></td><td>Selects DC Coupling</td></tr>
<tr><td><code>DcovWbk16CoupleAC</code></td><td>Selects AC Coupling</td></tr>
<tr><td rowspan="2"><code>DcotWbk16Sample</code></td><td rowspan="2">Used to bypass or to enable the WBK16's Simultaneous Sample and Hold (SSH) function.</td><td><code>DcovWbk16Bypassed</code></td><td>Bypasses SSH function</td></tr>
<tr><td><code>DcovWbk16Ssh</code></td><td>Enables the SSH function (for WBK16 only)</td></tr>
<tr><td rowspan="6"><code>DcotWbk16ExcDac</code></td><td rowspan="6">Used to select the calibrated excitation DAC values for WBK16.</td><td><code>DcovWbk16Exc0_0</code></td><td>Selects 0.0 V</td></tr>
<tr><td><code>DcovWbk16Exc0_5</code></td><td>Selects 0.5 V</td></tr>
<tr><td><code>DcovWbk16Exc1_0</code></td><td>Selects 1.0 V</td></tr>
<tr><td><code>DcovWbk16Exc2_0</code></td><td>Selects 2.0 V</td></tr>
<tr><td><code>DcovWbk16Exc5_0</code></td><td>Selects 5.0 V</td></tr>
<tr><td><code>DcovWbk16Exc10_0</code></td><td>Selects 10.0 V</td></tr>
<tr><td rowspan="4"><code>DcotWbk16IAG</code></td><td rowspan="4">Provides gain definitions for the WBK16 IAG (Instrumentation Amplifier Gain).

See Note 1 regarding Voltage Out & Total Gain.</td><td><code>DcovWbk16X1</code></td><td>Sets IAG gain to: x 1</td></tr>
<tr><td><code>DcovWbk16X10</code></td><td>Sets IAG gain to: x 10</td></tr>
<tr><td><code>DcovWbk16X100</code></td><td>Sets IAG gain to: x 100</td></tr>
<tr><td><code>DcovWbk16X1000</code></td><td>Sets IAG gain to: x 1000</td></tr>
</table>

Option Type and Value Definitions Continued--`DaqOptionType`			
WBK16 Options			
Option Type (`optionType`)	**Description**	**Option Value** (`optionValue`)	**Description**
`DcotWbk16PGA`	Provides gain definitions for the WBK16 PGA (Programmable Gain Amplifier) See Note 1 regarding Voltage Out & Total Gain.	`DcovWbk16X1_00`	Sets PGA gain to: x 1.00
		`DcovWbk16X1_28`	Sets PGA gain to: x 1.28
		`DcovWbk16X1_65`	Sets PGA gain to: x 1.65
		`DcovWbk16X2_11`	Sets PGA gain to: x 2.11
		`DcovWbk16X2_71`	Sets PGA gain to: x 2.71
		`DcovWbk16X3_48`	Sets PGA gain to: x 3.48
		`DcovWbk16X4_47`	Sets PGA gain to: x 4.47
		`DcovWbk16X5_74`	Sets PGA gain to: x 5.74
		`DcovWbk16X7_37`	Sets PGA gain to: x 7.37
		`DcovWbk16X9_46`	Sets PGA gain to: x 9.46
		`DcovWbk16X12_14`	Sets PGA gain to: x 12.14
		`DcovWbk16X15_58`	Sets PGA gain to: x 15.58
		`DcovWbk16X20_00`	Sets PGA gain to: x 20.00
`DmotWbk16Immediate`	Contains option value for applying Excitation Voltage.	`DmovWbk16ExcSrcApply`	Applies the excitation source voltage that is defined by `DcotWbk16ExcDac`.

Note 1: $V_{OUT} = [(V_{IN} \times IAG) + DacOffset_{SCALED}] \times PGA$

Where: V_{OUT} is Voltage Out, and V_{IN} is Voltage In

Note 2: Software Function for Dac Offset (Bridge Offset)

This note pertains to the `DcotWbk16OffsetDac` option type that is discussed in the preceding table. The Dac Offset, also referred to as *bridge offset*, can be set to values in the range of -3.0 volts to +3.0 volts, divided by the gain of the Instrumentation Amplifier (IAG). To do so, the following conversion is used:

Offset Value = [(Offset Voltage) x IAG x 682.5] + 2047.5

Where: Offset Value is a number in the range of 0.0 to 4095.0.

Example:
In this example we will determine the Offset Value; i.e., an integer in the range of 0 to 4095 that will give us a desired offset of 0.25 volts. We will make use of the following considerations. Actual applications will vary, accordingly.

- the Instrumentation Amplifier Gain (IAG) is set to x 10, i.e., `DcovWbk16X10`
- the desired Offset Voltage is 0.25 volts

Using the above equation we find that our offset value is 3754. The solution follows.

$(0.25 \times 10 \times 682.5) + 2047.5 = 3754$

Correlation of Offset Range with Instrumentation Amplifier Gain	
IAG Value	**Allowable Range for Offset Voltage**
x 1	± 3.0 volts
x 10	± 0.3 volts
x 100	± 0.03 volts
x 1000	± 0.003 volts

*Option Type and Value Definitions Continued--**DaqOptionType***			
WBK17 Options			
Option Type **(optionType)**	**Description**	**Option Value** **(optionValue)**	**Description**
DcotWbk17Level	Sets the comparator threshold level.	- 12.5 to +12.5 Volts	Threshold level.
DcotWbk17Coupling	Sets the input coupling. Use AC coupling to reject unwanted DC offsets. Use DC coupling when both AC and DC components are to be presented to the comparator as input.	DcovWbk17CoupleOff	Turns coupling "off."
		DcovWbk17CoupleAC	Sets coupling to AC.
		DcovWbk17CoupleDC	Sets coupling to DC.
DcotWbk17FilterType	Used to bypass the single-pole low-pass filter (LPF), or set it to one of three cut-off frequencies: 100kHz, 20kHz, or 30 Hz. Used to reject low-level noise.	DcovWbk17FltBypass	Bypasses the analog filter.
		DcovWbk17Flt100KHz	Sets cut-off frequency to 100 kHz.
		DcovWbk17Flt20KHz	Sets cut-off frequency to 20 kHz.
		DcovWbk17Flt30Hz	Sets cut-off frequency to 30 Hz.
DcotWbk17DebounceTime	Used to bypass the debounce mode, or to set a channel's comparator output to one of 16 debounce times. Debounce is used to eliminate switch-induced transients typically associated with electro-mechanical devices including relays, proximity switches, and encoders. Note that there are two debounce modes, "After Stable" and "Before Stable." See DcotWbk17DebounceTrigger.	DcovWbk17Debounce500ns	500 ns.
		DcovWbk17Debounce1500ns	1500 ns.
		DcovWbk17Debounce3500ns	3500 ns.
		DcovWbk17Debounce7500ns	7500 ns.
		DcovWbk17Debounce15500ns	15500 ns.
		DcovWbk17Debounce31500ns	31500 ns.
		DcovWbk17Debounce63500ns	63500 ns.
		DcovWbk17Debounce127500ns	127500 ns.
		DcovWbk17Debounce100μs	100 μs.
		DcovWbk17Debounce300μs	300 μs.
		DcovWbk17Debounce700μs	700 μs.
		DcovWbk17Debounce1500μs	1500 μs.
		DcovWbk17Debounce3100μs	3100 μs.
		DcovWbk17Debounce6300μs	6300 μs.
		DcovWbk17Debounce12700μs	12700 μs.
		DcovWbk17Debounce25500μs	25500 μs.
		DcovWbk17DebounceNone	Selects debounce bypass.
DcotWbk17Edge	Determines whether the rising edge or falling edge is to be detected.	DcovWbk17RisingEdge	Selects the "Rising Edge" for detection.
		DcovWbk17FallingEdge	Selects the "Falling Edge" for detection.
DcotWbk17TickSize	Determines the ticksize. The ticksize is the fundamental unit of time for period, pulsewidth, and timing measurements.	DcovWbk17Tick20ns	Sets ticksize to 20 ns.
		DcovWbk17Tick200ns	Sets ticksize to 200 ns.
		DcovWbk17Tick2000ns	Sets ticksize to 2 μs.
		DcovWbk17Tick20000ns	Sets ticksize to 20 μs.

Option Type and Value Definitions Continued--`DaqOptionType`			
WBK17 Options *(continued)*			
Option Type **(`optionType`)**	**Description**	**Option Value** **(`optionValue`)**	**Description**
`DcotWbk17DebounceTrigger`	Sets the mode of the debounce module to Trigger After Stable, or to Trigger Before Stable.	`DcovWbk17TriggerAfterStable`	Selects the "Trigger After Stable" mode. This mode rejects glitches and only passes state transitions after a specified period of stability (the debounce time). This mode is used with electro-mechanical devices like encoders and mechanical switches to reject switch bounce and disturbances due to a vibrating encoder that is not otherwise moving. The debounce time should be set short enough to accept the desired input pulse but longer than the period of the undesired disturbance.
		`DcovWbk17TriggerBeforeStable`	Selects the Trigger Before Stable mode. Use this mode when the input signal has groups of glitches and each group is to be counted as one. The trigger before stable mode will recognize and count the first glitch within a group but reject the subsequent glitches within the group if the debounce time is set accordingly. In this case the debounce time should be set to encompass one entire group of glitches.

`DcotWbk17MapChannel`	Used to select the mapped channel to be either one of the counter input channels or one of the detection output signals.
	
	A mapped channel is one of 16 signals that can get multiplexed into a channel's counter module. The mapped channel can participate with the channel's input signal by gating the counter, clearing the counter, etc. The 16 possible choices for the mapped channel are the 8 input signals (post debounce) and the 8 detection signals.
		`DcovWbk17Map_Channel_1`
		`DcovWbk17Map_Channel_2`
		`DcovWbk17Map_Channel_3`
		`DcovWbk17Map_Channel_4`
		`DcovWbk17Map_Channel_5`
		`DcovWbk17Map_Channel_6`
		`DcovWbk17Map_Channel_7`
		`DcovWbk17Map_Channel_8`
			Selects the mapped channel to be one of the counter input channels.
			
			There are 8 post-debounce channel input signals that can be individually selected as mapped channels.
			These are indicated as Channel_1 for Channel 1, Channel_2 for Channel 2, etc.

		`DcovWbk17Map_Detect_1`
		`DcovWbk17Map_Detect_2`
		`DcovWbk17Map_Detect_3`
		`DcovWbk17Map_Detect_4`
		`DcovWbk17Map_Detect_5`
		`DcovWbk17Map_Detect_6`
		`DcovWbk17Map_Detect_7`
		`DcovWbk17Map_Detect_8`
			Selects the mapped channel to be one of the detection output signals.
			
			Each input channel has an associated detection signal, e.g., Detect_1 for Channel 1. The detection signal will go active high when the channel's counter value meets the detection module's setpoint criteria (programmed into the pattern detection module).

Option Type and Value Definitions Continued--`DaqOptionType`			
WBK17 Options *(continued)*			
Option Type (`optionType`)	**Description**	**Option Value** (`optionValue`)	**Description**
`DcotWbk17DetectClear`	Clears or resets a channel or a unit (all channels).	`DcovWbk17DetClr_Chan`	Clears a channel.
		`DcovWbk17DetClr_All`	Clears all channels.
`DcotWbk17DetectControl`	Sets the type of comparison to be made. Can also be used to set the detection to "off" or to update the digital output.	`DcovWbk17DetCtrl_Off`	Detection setting "off."
		`DcovWbk17DetCtrl_Below_Low`	Sets count for "below low limit."
		`DcovWbk17DetCtrl_Above_High`	Sets count for "above high limit."
		`DcovWbk17DetCtrl_Outside_Range`	Sets count for "Outside of range."
		`DcovWbk17DetCtrl_Inside_Range`	Sets for "Inside the range."
		`DcovWbk17DetCtrl_Dig_Eq_Dig`	Sets for "Digital comp setpoint equals the Digital Output value." Digcomp equals the value present on the digital output port.
		`DcovWbk17DetCtrl_Update_Dig`	Updates digital output port, using DigMask and DigOut.
`DcotWbk17DetectLowLimit`	Sets low limit.	0 to 65535	Select value from applicable range.
`DcotWbk17DetectHighLimit`	Sets high limit.	0 to 65535	
`DcotWbk17DetectDigComp`	Sets DigComp	0 to 255	
`DcotWbk17DetectDigMask`	Sets DigMask	0 to 255	
`DcotWbk17DetectDigOut`	Sets Digital Port Output	0 to 255	
`DcotWbk17MeasurementMode`	Enumeration with Bit-Masking	**`DcovWbk17Mode_OFF`**	Set Measurement Modes to "Off."
`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking. The options can be combined.	**`DcovWbk17Mode_Counter`**	**Counter Mode**
		`DcovWbk17Counter_Totalize`	**Totalize Mode** – The counter counts up and rolls over on the 16-bit or 32-bit boundary.
		`DcovWbk17Counter_ClearOnRead`	**Clear On Read Mode -** The counter is cleared at the beginning of every scan; and the final value of the counter [the value just before it was cleared] is latched and returned to the Wavebook/516.
		`DcovWbk17Counter_RollOver`	**Rollover Mode** - The counter continues to count upward, rolling over on the 16-bit or 32-bit boundary.
		`DcovWbk17Counter_StopOnTop`	**Stop at the Top Mode** - The counter will stop at the top of its count. The top of the count is FFFF for the 16-bit option and FFFFFFFF for the 32-bit option.

WBK17 Options *(continued)*			
Option Type **(optionType)**	**Description**	**Option Value (optionValue)**	**Description**
`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking. The options can be combined.	`DcovWbk17Counter_LatchOnSOS`	Selects **start of scan**. Latches the counter outputs at the beginning of every scan.
		`DcovWbk17Counter_LatchOnMap`	Selects the **mapped signal** to latch the counter outputs. This allows the user to know the exact counter value when an edge is present on another channel.
		`DcovWbk17Counter_DecrementOff` `DcovWbk17Counter_DecrementOn`	Determines whether the counter decrement option is "off" or "on."
		`DcovWbk17Counter_CountChan` `DcovWbk17Counter_CountMap`	Selects channel for count. Selects mapped channel for count.
		`DcovWbk17ModeMask_16Bit` `DcovWbk17ModeMask_32Bit`	Selects 16-Bit counter. Selects 32-Bit counter.
		`DcovWbk17ModeMaskGatingOff` `DcovWbk17ModeMaskGatingOn`	Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).
`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking. The options can be combined.	**DcovWbk17Mode_Period**	**Period Mode**
		`DcovWbk17Period_X1` `DcovWbk17Period_X10` `DcovWbk17Period_X100` `DcovWbk17Period_X1000`	Used to select the number of periods to time, per measurement. Choices are: 1, 10, 100, or 1000
		`DcovWbk17Period_MeasChan` `DcovWbk17Period_MeasMap`	Selects to measure input channel's period. Selects to measure the mapped channel's period.
		`DcovWbk17ModeMask_16Bit` `DcovWbk17ModeMask_32Bit`	Selects 16-Bit counter. Selects 32-Bit counter.
		`DcovWbk17ModeMaskGatingOff` `DcovWbk17ModeMaskGatingOn`	Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

WBK17 Options *(continued)*			
Option Type **(`optionType`)**	**Description**	**Option Value** **(`optionValue`)**	**Description**
`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking. The options can be combined.	**`DcovWbk17Mode_Pulsewidth`**	**Pulsewidth Mode**

		`DcotWbk17PulseWidth_MeasChan`
		
		`DcotWbk17PulseWidth_MeasMap`
			Selects to measure input channel's pulsewidth.
			Selects to measure the mapped channel's pulsewidth.

		`DcovWbk17ModeMask_16Bit` `DcovWbk17ModeMask_32Bit`	Selects 16-Bit counter. Selects 32-Bit counter.

		`DcovWbk17ModeMaskGatingOff`
		
		`DcovWbk17ModeMaskGatingOn`
			Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking. The options can be combined.	**`DcovWbk17Mode_Timing`**	**Timing Mode**
		`DcovWbk17ModeMask_16Bit` `DcovWbk17ModeMask_32Bit`	Selects 16-Bit counter. Selects 32-Bit counter.

		`DcovWbk17ModeMaskGatingOff`
		
		`DcovWbk17ModeMaskGatingOn`
			Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking. The options can be combined.	**`DcovWbk17Mode_Encoder`**	**Encoder Mode**

		`DcovWbk17Encoder_1X`
		`DcovWbk17Encoder_2X`
		`DcovWbk17Encoder_4X`
			Determines the encoder measurement mode: 1X, 2X, or 4X.

		`DcovWbk17Encoder_LatchOnSOS`	Selects **start of scan**. Latches the counter outputs at the beginning of every scan.
		`DcovWbk17Encoder_LatchOnZ`	Selects the Encoder Z **mapped signal** to latch the counter outputs. This allows the user to know the exact counter value when an edge is present on another channel.

		`DcovWbk17Encoder_ClearOnZ_Off`
		
		`DcovWbk17Encoder_ClearOnZ_On`
			Selects "clear on Z" On or Off. When On, the encoder Z is referenced to clear the counter. The counter is cleared on the rising edge of the mapped (Z) channel.

		`DcovWbk17ModeMask_16Bit` `DcovWbk17ModeMask_32Bit`	Selects 16-Bit counter. Selects 32-Bit counter.

*Option Type and Value Definitions Continued--***DaqOptionType**			
WBK17 Options *(continued)*			
Option Type (`optionType`)	**Description**	**Option Value (`optionValue`)**	**Description**

`DcotWbk17MeasurementMode` *(continued)*	Enumeration with Bit-Masking.
	
	The options can be combined.
		`DcovWbk17ModeMaskGatingOff`
		
		`DcovWbk17ModeMaskGatingOn`
			Gating can be selected On or Off. When "**On**", the counter is enabled when the **mapped channel to gate the counter** is **high**. When the mapped channel is **low**, the counter is disabled (but holds the count value).

*Option Type and Value Definitions Continued--***DaqOptionType**			
WBK18 Options			
Option Type (`optionType`)	**Description**	**Option Value (`optionValue`)**	**Description**
DcotWbk18LowPassMode	Configure Low Pass input filter for Bypass, 2-Pole or 8-Pole on a WBK18	DcovWbk18LowPassBypass	Bypass the Low Pass Filter for the specified WBK18 Channel
		DcovWbk18LowPass_2_Pole	Set the Low Pass Filter to 2-Pole roll off for the specified channel
		DcovWbk18LowPass_8_Pole	Sets the Low Pass Filter to 8-Pole roll off for the specified channel
DcotWbk18LowPassCutOff	Sets the Low Pass input filter cutoff frequency for 2 pole or 8 pole mode	DcovWbk18LPF_Cutoff_10Hz	Sets the Low Pass input filter to 10Hz for the specified channel
		DcovWbk18LPF_Cutoff_20Hz	Sets the Low Pass input filter to 20Hz for the specified channel
		DcovWbk18LPF_Cutoff_50Hz	Sets the Low Pass input filter to 50Hz for the specified channel
		DcovWbk18LPF_Cutoff_100Hz	Sets the Low Pass input filter to 100Hz for the specified channel
		DcovWbk18LPF_Cutoff_200Hz	Sets the Low Pass input filter to 200Hz for the specified channel
		DcovWbk18LPF_Cutoff_500Hz	Sets the Low Pass input filter to 500Hz for the specified channel
		DcovWbk18LPF_Cutoff_1000Hz	Sets the Low Pass input filter to 1000Hz for the specified channel
		DcovWbk18LPF_Cutoff_2000Hz	Sets the Low Pass input filter to 2000Hz for the specified channel
		DcovWbk18LPF_Cutoff_5000Hz	Sets the Low Pass input filter to 5000Hz for the specified channel
		DcovWbk18LPF_Cutoff_10000Hz	Sets the Low Pass input filter to 10000Hz for the specified channel
		DcovWbk18LPF_Cutoff_20000Hz	Sets the Low Pass input filter to 20000Hz for the specified channel
		DcovWbk18LPF_Cutoff_50000Hz	Sets the Low Pass input filter to 50000Hz for the specified channel

WBK18 Options *(continued)*			
Option Type (`optionType`)	**Description**	**Option Value (`optionValue`)**	**Description**
DcotWbk18HighPassCutOff	Configures the High Pass input filter for 0.1Hz, 10Hz or DC coupling	DcovWbk18HighPass0_1Hz	Sets the High Pass input filter to 0.1Hz
		DcovWbk18HighPass10Hz	Sets the High Pass input filter to 10Hz
		DcovWbk18HighPassDC	Sets the High Pass input filter to DC coupling
DcotWbk18CurrentSrc	Enables/disables channel output current to power ICP sensors – active at acquisition arming	DcovWbk18CurrentSrcOff	Turn current off
		DcovWbk18CurrentSrc4mA	Turn current on
DcotWbk18OverRangeEnable	Enables/disables channel over-range detection for the specified channel	DcovWbk18OverRangeOff	Disable
		DcovWbk18OverRangeOn	Enable
DcotWbk18CurrentSrcImmediate	Enables/disables channel output current to power ICP sensors – active immediately – for the specified channel	DcovWbk18CurrentSrcOff	Turn current off
		DcovWbk18CurrentSrc4mA	Turn current on
DmotWbk18OverRangeLimit	Configure over-range detection condition	1.0 to 100.0 (% of range)	Sets the level for the over-range circuitry to detect an over-range condition – applies to all channels
DmotWbk18OverRangeEnable	Enables/disable over-range detection for all eight channels	0 to 255 (bit mask)	Each bit represents a channel. Writing 255 enables all channels
DmotWbk18esMode	Configures the analog output for continuous sine or swept sine mode	DmovWbk18esSine	Continuous sine wave output
		DmovWbk18esSweptSine	Sweep through pre-set frequency amplitude values
DmotWbk18esAmplitude	Set the amplitude for continuous or swept modes	DmovWbk18esAmplitude10pp	10 volts p-p
		DmovWbk18esAmplitude5pp	5 volts p-p
		DmovWbk18esAmplitude2pp	2 volts p-p
		DmovWbk18esAmplitude1pp	1 volt p-p
		DmovWbk18esAmplitude0_5pp	0.5 volt p-p
		DmovWbk18esAmplitude0_2pp	0.2 volt p-p
		DmovWbk18esAmplitude0_1pp	0.1 volt p-p
		DmovWbk18esAmplitude0_0pp	0.0 volt p-p
DmotWbk18esFreq	Set the frequency for continuous or swept modes	1.0 to 5000 (Hz)	Specifies the output frequency
DmotWbk18esRelay	Controls the output relay	DmovWbk18esRelayOpen	Specify relay open
		DmovWbk18esRelayClosed	Specify relay closed
DmotWbk18esFreqCycleTime	Swept mode buffer cycle time. If total RAM segments times the duration time exceeds the cycle time then cycle time is ignored.	50 to 85699000 (microseconds)	Specify time in microseconds
DmotWbk18esFreqDurationTime	Specifies the duration a frequency will be at the output	50 to 85699000 (microseconds)	Specify time in microseconds

<table>
<tr><td colspan="4" align="center"><i>Option Type and Value Definitions Continued--</i>DaqOptionType</td></tr>
<tr><td colspan="4" align="center">WBK18 Options <i>(continued)</i></td></tr>
<tr><td>Option Type (optionType)</td><td>Description</td><td>Option Value (optionValue)</td><td>Description</td></tr>
<tr><td rowspan="4">DmotWbk18esImmediate</td><td rowspan="4">Performs an immediate action on the analog output</td><td>DmovWbk18esClearRAM</td><td>Erases all 1280 RAM segments</td></tr>
<tr><td>DmovWbk18esWriteRAM</td><td>Increments the segment pointer and writes the current frequency, amplitude, relay position, cycle time and duration time to WBK18 RAM segment.</td></tr>
<tr><td>DmovWbk18esStart</td><td>Start output</td></tr>
<tr><td>DmovWbk18esStop</td><td>Disable output – no voltage</td></tr>
<tr><td rowspan="2">DmotWbk18esRelay</td><td rowspan="2">Used to reset, turn on and turn off the excitation source</td><td>DmovWbk18esRelayOpen</td><td>Open relay excitation source</td></tr>
<tr><td>DmovWbk18esRelayClosed</td><td>Close relay excitation source</td></tr>
<tr><td rowspan="4">DmotWbk18esImmediate</td><td rowspan="4"></td><td>DmovWbk18esStop</td><td>Stops excitation output</td></tr>
<tr><td>DmovWbk18esStart</td><td>Starts excitation output</td></tr>
<tr><td>DmovWbk18esWriteRAM</td><td>Writes Amp/Freq/Relay values to excitation source RAM</td></tr>
<tr><td>DmovWbk18esClearRAM</td><td>Clears excitation source RAM</td></tr>
<tr><td rowspan="2">DbotIgnoreFirstScan</td><td rowspan="2">Set to On if using a Wavebook as a Master device and a DaqLab, DaqScan or DaqBook, WBK40/41 as a slave device</td><td>DbovIgnoreFirstScanOff</td><td>Includes first scan</td></tr>
<tr><td>DbovIgnoreFirstScanOn</td><td>Excludes first scan</td></tr>
</table>

Function Usage

The daqSetOption function may be used to set options or configuration settings for a device, module or channel. Generally, this function allows setting states or configuration information for a particular feature for a given device, module or channel that cannot or should not be set using normal scan configuration settings.

When configuring options which relate to module or channel configurtions, this function should be called to set the option prior to arming the acquisition (see daqAdcArm) . The device should have already been opened prior to calling this function (see daqOpen) and the handle parameter indicates the device for which the option is to apply.

The flags parameter indicates if the option applies to a channel or an entire channel bank according to the module used. The flags parameter may be set to the following:

    DcofChannel -- Apply option to the channel specified by the chan parameter.
    DcofModule -- Apply option to the entire module for which chan is located.

The optionType specifies which option to apply. See the table in the "Parameter Type Definitions" section for a complete description of valid option types.

The optionValue parameter specifies the value to set the option specfied by optionType (if applicable). See the table in the "Parameter Type Definitions" section for more detailed description of the valid option values for the desired option type.

Returns

    DerrNoError   No error

Prototypes

C/C++

```
daqSetOption(DaqHandleT handle, DWORD chan, DWORD flags, DaqOptionType
optionType, FLOAT optionValue);
```

Visual BASIC

```
VBdaqSetOption&(ByVal handle&, ByVal chan&, ByVal flags&, ByVal optionType&,
ByVal optionValue!)
```

Program References

```
DAQTMREX01.CPP, DBK04EX.CPP, DBK07EX.CPP
```

daqSetTimeout

Format

```
daqSetTimeout (handle, mSecTimeout)
```

Purpose

daqSetTimeout sets the time-out for waiting on either a single event or multiple events.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the event time-out is to be set
mSecTimeout	DWORD	Specifies time-out (in milliseconds) for events

Parameter Values

handle: obtained from the **DaqOpen** function

mSecTimeout: valid values range from 1 to 4,294,967,295; however, large values will cause timeout to be excessively long

Function Usage

The daqSetTimeout function can be used in conjunction with the daqWaitForEvent and daqWaitForEvents functions to specify a maximum amount of time to wait for the event(s) to be satisfied.

The mSecTimeout parameter specifies the maximum amount of time (in milliseconds) to wait for the event(s) to occur. If the event(s) do not occur within the specified time-out, the daqWaitForEvent and/or daqWaitForEvents will return.

If this function is not called, a default timeout of 10,000 milleseconds (10 seconds) will be used.

The daqSetTimeout function can be used for daqAdcRdN functions.

Returns

DerrNoError   No error

Prototypes

C/C++

```
daqSetTimeout(DaqHandleT handle, DWORD mSecTimeout);
```

Visual BASIC

```
VBdaqSetTimeout&(ByVal handle&, ByVal mSecTimeout&)
```

Program References

DAQADCEX04.CPP, DAQADCEX05.CPP, DAQADCEX06.CPP

daqSetTriggerEvent

Format

```
daqSetTriggerEvent(handle, trigSource, trigSensitivity, channel, gainCode,
flags, channelType, level, variance, event)
```

Purpose

daqSetTriggerEvent sets an acquisition trigger start event or an acquisition stop event.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the trigger event is to be configured
trigSource	DaqAdcTriggerSource	Trigger source
trigSensitivity	DaqEnhTrigSensT	Trigger sense and direction
channel	DWORD	Actual channel number of the trigger channel (NOT the scan list location)
gainCode	DaqAdcGain	Trigger channel gain code
flags	DWORD	Trigger channel flags
channelType	DaqChannelType	Type of channel
level	FLOAT	Trigger level in expected engineering units of the channel type (Volts, Counts, or Deg C)
variance	FLOAT	Variance in the trigger
event	DaqTriggerEvent	Trigger event

Parameter Values

handle: obtained from the **DaqOpen** function

trigSource: see table below

trigSenseitivity: see table below

channel: valid values range from 0 to 271

gainCode: see ADC Gain Definition table for gain parameter values

flags: see ADC Flag Definition table for flags parameter values

channelType: see table below

level: valid values depend on channel type—see channelType table below

variance: valid values depend on channel type—see channelType table below

event: valid values indicate either a start trigger event (daqStartEvent) or a stop trigger event
(daqStopEvent)

Parameter Type Definitions

triggerSource-(DaqAdcTriggerSource)	
Definition	**Description**
DatsImmediate (Trigger event only)	Post-trigger data acquisition begins immediately upon invocation of the daqAdcArm function (no pre-trigger data acquisition is possible with this trigger source)
DatsSoftware	Post-trigger data acquisition begins upon a software command issued by the calling application (see daqAdcSoftTrig)
DatsAdcClock Trigger event only)	Post-trigger data acquisition begins immediately upon invocation detection of the ADC Clock pulse being driven.
DatsGatedAdcClock Trigger event only)	Post-trigger data acquisition begins immediately upon invocation detection of the ADC Clock pulse being driven.
DatsExternalTTL (Trigger event only)	Post-trigger data acquisition begins on the selectable edge of an external TTL signal.
DatsHardwareAnalog (Trigger event only)	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.) Must be defined as the first channel in the channel scan group
DatsSoftwareAnalog	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)
DatsDigPattern	Post-trigger data acquisition beings upon receiving a specified digital pattern on a P2 or P3 digital port.
DatsPulse* (Trigger event only)	Post-trigger data acquisition begins upon detection of a pulse of specified duration and magnitude on an analog input channel.
DatsCounter (P3)	Post-trigger data acquisition begins upon detection of specified counter criteria
DatsScanCount (Stop event only)	Stop collecting post-trigger data when the specified number of post-trigger scans are completed

triggerSensitivity-(DaqEnhTrigSensT)		
Definition	**Trigger Sources**	**Description**
DetsRisingEdge	DatsExternalTTL DatsHardwareAnalog DatsSoftwareAnalog DatsEnhancedTrig	Triggers when the signal goes from *low to high* (TTL trigger) or *rises through* a specified level (Hardware & Software Analog)
DetsFallingEdge	DatsExternalTTL DatsHardwareAnalog DatsSoftwareAnalog DatsEnhancedTrig	Triggers when the signal goes from *high to low* (TTL trigger) or *falls through* a specified level (Hardware & Software Analog)
DetsAboveLevel	DatsExternalTTL DatsHardwareAnalog DatsSoftwareAnalog DatsEnhancedTrig	Triggers when the signal is *above* a specified level (Hardware & Software Analog and Digital Trigger)
DetsBelowLevel	DatsExternalTTL DatsHardwareAnalog DatsSoftwareAnalog DatsEnhancedTrig	Triggers when the signal is *below* a specified level (Hardware & Software Analog and Digital Trigger)
DetsEQLevel	DatsSoftwareAnalog DatsEnhancedTrig DatsDigPattern	Triggers when the signal *equals* a specified level (Hardware & Software Analog and Digital Trigger)
DetsNELevel	DatsSoftwareAnalog DatsEnhancedTrig DatsDigPattern	Triggers when the signal does *not equal* a specified level (Hardware & Software Analog and Digital Trigger)
DetsWindow	DatsPulse	Triggers at a specified pulse width or height

The data ranges described in the following table represent the maximum range over which the level and variance parameters can be set for the channel type selected. The ranges for the particular channel may be actually smaller depending upon the maximum A/D range of main unit, the gain, the polarity and/or other range settings for which the actual input channel is configured.

channelType-(DaqChannelType)		
Definition	**Data Representation**	**Valid Data Ranges (level and variance)**
DaqTypeAnalogLocal	Voltage	DaqBook/100 Series and /200 Series: -5.0 to +10.0 (Volts)
		DaqBoards(ISA): -5.0 to +10.0 (Volts)
		DaqBoard/500 Series: -10.0 to +10.0 (Volts)
		DaqBoard/1000 Series: -10 to +10.0 (Volts)
		DaqBoard/2000 Series: -10.0 to +10.0 (Volts)
		DaqBook/2000 Series: -10.0 to +10.0 (Volts)
		DaqLab/2000 Series: -10.0 to +10.0 (Volts)
		DaqScan/2000 Series: -10.0 to +10.0 (Volts) Daq PC Cards: -10.0 to +10.0 (Volts)
		WaveBook/512: -5.0 to +10.0 (Volts)
DaqTypeDigitalLocal	Digital Pattern	P2: 0 to 255 (Dec) P3: 0 to 65535 (Dec)
DaqTypeDigitalExp	Digital Pattern	P2: 0 to 255 (Dec)
DaqTypeCounterLocal	Counter Value	P3: 0 to 65535 (Dec)
DaqTypeDBK1	Voltage	-10.0 to +10.0 (Volts)
DaqTypeDBK4	Voltage	-5.0 to +10.0 (Volts)
DaqTypeDBK7	Voltage	-5.9 to +5.0 (Volts)
DaqTypeDBK8	Voltage	-5.0 to +5.0 (Volts)
DaqTypeDBK9	Temperature (Degrees C)	-200.0 to 850.0 (Degrees C)
DaqTypeDBK12	Voltage	-5.0 to +10 (Volts)
DaqTypeDBK13	Voltage	-5.0 to +10 (Volts)
DaqTypeDBK14	Voltage	-5.0 to +5.0 (Dependent upon TC type)
DaqTypeDBK15	Voltage	-5.0 to +10 (Volts)
DaqTypeDBK16	Voltage	-5.0 to +5.0 (Volts)
DaqTypeDBK17	Voltage	-5.0 to +10.0 (Volts)
DaqTypeDBK18	Voltage	-5.0 to +10.0 (Volts)
DaqTypeDBK19	Temperature (Degrees C)	-270.0 to 1820.0 (Dependent upon TC type)
DaqTypeDBK20	Digital Pattern	0 – 255 (Dec)
DaqTypeDBK21	Digital Pattern	0 – 255 (Dec)
DaqTypeDBK22	Digital Pattern	0 – 255 (Dec)
DaqTypeDBK23	Digital Pattern	0 – 255 (Dec)
DaqTypeDBK24	Digital Pattern	0 – 255 (Dec)
DaqTypeDBK25	Digital Pattern	0 – 255 (Dec)
DaqTypeDBK42	Voltage	-5.0 to +5.0 (Volts)
DaqTypeDBK50	Voltage	-300.0 to +300.0 (Volts)
DaqTypeDBK51	Voltage	-10.0 to +10.0 (Volts)
DaqTypeDBK52	Temperature (Degrees C)	-270.0 to 1820.0 (Dependent upon TC type)
DaqTypeDBK53	Voltage	-10.0 to +10.0 (Volts)
DaqTypeDBK54	Voltage	-10.0 to +10.0 (Volts)
DaqTypeDBK56	Voltage	-10.0 to +10.0 (Volts)
DaqTypeDBK70	Voltage	0.0 to 5.0 (Volts)

Returns

DaqNoError   No error

Function Usage

Defining the Trigger Channel

The channel parameter selects the trigger channel. The trigger channel must be a configured channel in the scan list (see daqAdcSetScan). The type of channel selected can be set by the channelType parameter. The channel type will be used to properly interpret the value against which the device should be triggered (see the "Setting the Trigger Level" and "Setting the Trigger Variance" sections below) in the context of the channel type selected. For instance, if the channel is an thermocouple channel then the level and variance parameters will be interpreted as values in degrees Celsius. The flags parameter defines the configured settings for the channel and should match the flags setting for the trigger channel configured using the daqAdcSetScan function. Likewise, the gainCode parameter should match the gain code setting when the channel was configured using the daqAdcSetScan function. If the same channel is in the scan multiple times, the 1st occurrence of the channel will be used as the trigger channel.

Error Checking

It is important to note that only basic error checking in done when daqSetTriggerEvent is called. Full verification against the scan list and acquisition mode will take place during the daqAdcArm function.

Selecting the Trigger Event

The event parameter selects the definition of the trigger event to be either a start trigger event (DaqStartEvent) or a stop trigger event (DaqStopEvent). The start trigger event defines the conditions under which post-trigger acquisition data collection should be initiated or triggered. Stop events are events which signal the current data acquisition process to terminate.

Setting the Trigger Source

The trigSource parameter specifies the trigger source to use as the start or stop trigger event. The start and stop trigger event sources are both programmed using selected definitions from the daqAdcTriggerSource data type. As described below, start and stop trigger event definitions may differ from device to device, so it is important to note which start and stop trigger events can be configured for your device.

Setting the Start Trigger Event Source

The start trigger event defines the conditions under which post-trigger acquisition data collection should be initiated or triggered. The start trigger event can vary in complexity from starting immediately, to starting on complex channel value definitions. See the table in the "Setting the Trigger Variance" section for a list of some valid start and stop trigger event definitions.

Setting the Stop Event Source

Stop events are events which signal the current data acquisition process to terminate. Here again, the stop event can be as simple as that of a scan count, or as complex as involving a channel value level condition. Generally, there are fewer options with stop event definitions than start event definitions. However, the DaqBook/2000 Series and DaqBoard/2000 series products do provide a rich set of stop event features based upon software channel value definitions.

Setting the Trigger Level

The `level` parameter is used for those trigger types who depend on an input channel comparison to detect the trigger start or stop event. The `level` parameter is a single precision floating point value which represents, in engineering units, the level at (or around which) the trigger event should be detected. The actual level at which the trigger event is detected depends upon trigger sensing and variability discussed later. Below is a table describing the various ranges of possible values for the level paramter based on the trigger source:

Trigger Level Settings	
Trigger/Stop Sources (daqAdcTriggerSource)	**Meaning of `level` parameter**
`DatsSoftwareAnalog (P1)`	Voltage or temperature (in degrees C) at which the trigger level is to be defined. Trigger detection performed in software.
`DatsHardwareAnalog (P1)`	Voltage or temperature (in degrees C) at which the trigger level is to be defined. Trigger detection performed in hardware.
`DatsDigPattern (P2/P3)`	Defines bit pattern for the digital channel trigger. The valid values are: 0.0 (no bits set) - 255.0 (all bits set) for 8-bit digital channel banks (P2) 0.0 (no bits set) – 65,535.0 (all bits set) for 16-bit channel banks (P3)
`DatsCounter (P3)`	Pulse or Totalize counter values (0.0 – 65,535)

> **Note!** **DaqBooks, TempBooks, DaqBoard (ISA) and Daq PC Cards have a fixed hardware based hysteresis value, so the variance parameter is ignored for those devices when using with `DatsHardwareAnalog` trigger source.**

Setting the Trigger Sensitivity

Some trigger sources require a trigger sensitivity setting. The `trigSensitivity` setting is only required for trigger sources which are based upon an input signal. The trigger sensitivity normally defines the way in which a trigger event is detected based upon the characteristics of the trigger input signal. Often, it defines the way in which the trigger input signal(s) should be compared to the trigger `level` parameter value.

Setting the Trigger Variance

While the `trigSense` parameter indicates the direction of the input signal relative to the `level` parameter, the `variance` parameter specifies the degree to which the input signal can vary relative to `level` parameter. The `variance` parameter is a single-precision floating point value which represents, in engineering units, the amount that the trigger event can vary from the `level` parameter.

The range of trigger values defined by the `variance` and `level` parameters depends also upon the `trigSensitivity` setting and the type of input channel that is configured as the trigger channel. The following table describes how the the `trigSource` and `trigSensitivity` parameters influence the trigger values established by the `level` and `variance` parameters.

Note! **DatsHardwareAnalog can only be used when the trigger channel is first channel in the scan group configuration (see daqAdcSetScan)**

Interaction of Trigger Variance with Trigger Level and Trigger Sensitivities		
Trigger Start/Stop Source	**Trigger Sensitivities**	**Trigger Start/Stop Criteria**
Analog (P1) DatsHardwareAnalog	DetsRisingEdge	Triggers/Stops when: *signal value* $<$ (**level-variance**) Then, *signal value* $>$ **level**
	DetsFallingEdge	Triggers/Stops when: *signal value* $>$ (**level+variance**) Then, *signal value* $<$ **level**
	DetsAboveLevel	Triggers/Stops when: *signal value* $>$ (**level**)
	DetsBelowLevel	Triggers/Stops when: *signal value* $<$ (**level**)
Analog (P1) DatsSoftwareAnalog	DetsRisingEdge	Triggers/Stops when: *signal value* $<$ (**level-variance**) Then, *signal value* $>$ **level**
	DetsFallingEdge	Triggers/Stops when: *signal value* $>$ (**level+variance**) Then, *signal value* $<$ **level**
	DetsAboveLevel	Triggers/Stops when: *signal value* $>$ (**level**)
	DetsBelowLevel	Triggers/Stops when: *signal value* $<$ (**level**)
	DetsEQLevel	Triggers/Stops when: (**level-variance**)$<$ *signal value* $<$ (**level+variance**)
	DetsNELevel	Triggers/Stops when: *signal value* **<** (**level-variance**) or *signal value* **>** (**level+variance**)
Digital (P2/P3) DatsDigPattern	DetsAboveLevel	Triggers/Stops when: *Dig channel bank* $>$ (**level** AND (bitwise)**variance**)
	DetsBelowLevel	Triggers/Stops when: *Dig channel bank* $<$ (**level** AND (bitwise)**variance**)
	DetsEQLevel	Triggers/Stops when: *Dig channel bank* $=$ (**level** AND (bitwise)**variance**)
	DetsNELevel	Triggers/Stops when: *Dig channel bank* *!=* (**level** AND (bitwise)**variance**)
	DetsRisingEdge	Triggers/Stops when: *counter channel* $<$ (**level-variance**) Then, *counter channel* $>$ **level**
Counter (P3) DatsCounter	DetsFallingEdge	Triggers/Stops when: *counter channel* $>$ (**level+variance**) Then, *counter channel* $<$ **level**
	DetsAboveLevel	Triggers/Stops when: *Counter channel* $>$ (**level-variance**)
	DetsBelowLevel	Triggers/Stops when: *Counter channel* $<$ (**level+variance**)
	DetsEQLevel	Triggers/Stops when: (**level-variance**)**<** *counter channel* **<** (**level+variance**)
	DetsNELevel	Triggers/Stops when: *Counter channel* **<** (**level-variance**) Or *counter channel* **>** (**level+variance**)

Prototypes

C/C++

```
daqSetTriggerEvent(DaqHandleT handle, DaqAdcTriggerSource trigSource,
DaqEnhTrigSensT trigSensitivity, DWORD channel, DaqAdcGain gainCode, DWORD
flags, DaqChannelType channelType, FLOAT level, FLOAT variance,
DaqTriggerEvent event)
```

Visual BASIC

```
VBdaqSetTriggerEvent&(ByVal handle&, ByVal trigSource&, ByVal
trigSensitivity&, ByVal channel&, ByVal gainCode&, ByVal flags&, ByVal
channelType&, ByVal level!, ByVal variance!, ByVal trigEvent&)
```

Program References

```
DAQADCEX01.CPP, DAQADCEX02.CPP, DAQADCEX03.CPP, DAQADCEX05.CPP,
DAQADCEX06.CPP, DAQADCEX07.CPP, DBK04EX.CPP, DBK07EX.CPP, DBK08EX.CPP,
DBK09EX.CPP, DBK12_13EX.CPP, DBK15EX.CPP, DBK16EX.CPP, DBK17EX.CPP,
DBK18EX.CPP, DBK19EX.CPP, DBK42EX.CPP, DBK43EX.CPP, DBK44EX.CPP, DBK45EX.CPP,
DBK50EX.CPP, DBK51EX.CPP, DBK52EX.CPP, DBK53_54EX.CPP
```

daqTest

Format

```
daqTest(handle, command, count, cmdAvailable, result)
```

Purpose

daqTest tests a device for specific functionality.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the test is to be performed
command	DaqTestCommand	Specifies the type of test to be run
count	DWORD	Optional parameter which specifies the length of the test
cmdAvailable	PBOOL	Pointer to a return boolean indicating the availability of the test for the device
result	PDWORD	Pointer to the test result field

Parameter Values

handle: obtained from the **DaqOpen** function

command: see table below

count: valid values range from 1 to 4,294,967,295; however, large values will result in excessively long tests

cmdAvailable: pointer to a boolean value; values are either true ($\neq 0$) or false ($= 0$)

result: a pointer to a value ranging from 0 to 4,294,967,295

Parameter Type Definitions

command-(DaqTestCommand)	
Definition	**Description**
DtstBaseAddressValid	Test to determine if communications at currently configured base address are valid.
DtstInterruptLevelValid	Test to determine if conflicts exist at the currently configured IRQ.
DtstDmaChannelValid	Test to determine if conflicts exist at the currently configured DMA channel. This command definition verifies DMA via available Input and Output speed tests, i.e., DtstAdcFifoInputSpeed, DtstDacFifoOutputSpeed, DtstIOInputSpeed, and DtstIOOutputSpeed.
DtstAdcFifoInputSpeed	Test to determine max speed for analog data transfer to computer memory.
DtstDacFifoOutputSpeed	Test to determine max speed for analog data transfer from computer memory.
DtstIOInputSpeed	Test to determine max rate for digital data transfer to computer memory.
DtstIOOutputSpeed	Test to determine max rate for digital data transfer from computer memory.
DtstFifoAddrDataBusValid	WaveBook Only : WBK30 communication check (use DtstFifoMemCellValid instead)
DtstFifoMemCellValid	WaveBook Only : MegaFIFO (WBK30) memory test.
DtstHardwareCompatibility	WaveBook Only : Verifies the DSP Boot Code revision level. 1 = PASS, 0 = FAIL.
DtstFirmwareCompatibility	WaveBook Only : Verifies the FPGA revision level. 1 = PASS, 0 = FAIL.
DtstExpansionCompatibility	WaveBook Only: Verifies that the connected WBK modules are compatible with the main unit. 1 = PASS, 0 = FAIL.
DtstExpUpgradeCompatibility	WaveBook Only: Verifies that the main unit Firmware is compatible with all connected WBK modules. 1 = PASS, 0 = FAIL.

Returns

DerrNoError   No Error

Function Usage

Test types performed by the `daqTest` function vary; test results are based on the type of test requested. Tests can only be performed on valid, opened devices. If there are problems with the test, be sure to check that the device is properly configured, that the device is powered-on, and that it is properly connected.

The `command` parameter specifies the test to run. There are two main types of tests: resource and performance. The `cmdAvailable` parameter is a pointer to a Boolean value that indicates whether or not the specified test is available for the device.

The `count` parameter can be used to indicate the duration or length of the test. For instance, a resource test will be run `count` times; and if any one iteration of the test fails, it will indicate an overall failure of the test. For a performance test, the `count` parameter will indicate the number of times to run the test, and the test result will be an average of all the tests performed.

Resource Tests

Resource tests are pass/fail and are useful in determining if the device has the appropriate resources to function efficiently. If one or more of the resource tests fail, the **Daq Configuration** utility (found in the operating system's Control Panel) may be used to change the resource settings related to the problem. Valid resource test types are defined as follows:

`DtsBaseAddressValid` - This test will indicate if there is a problem communicating with the device at its currently specified base address. A non-zero in the `result` parameter will indicate a failed condition. For DaqBook/2000 Series , DaqBoard/2000 Series and cPCI DaqBoard/2000c Series this command definition returns the bus and slot code of the hardware as "slot + (bus * 256)."

`DtsInterruptLevelValid` - This test is not available for DaqBook/2000 Series, DaqBoard/2000 Series or cPCI DaqBoard/2000c Series boards. For other devices the test will indicate if there is a problem with performing acquisitions using interrupts. A non-zero in the `result` parameter will indicate a failed condition. If this is the case, the interrupts may not be properly configured (if the device is a DaqBook, the LPT interrupts may not be enabled on the system) or an interrupt conflict exists with another device.

`DtsDmaChannelValid` - (DaqBoards only) This test will indicate if there is a problem with performing acquisitions through DMA transfers with the currently configured DMA channel for the device. A non-zero in the `result` parameter will indicate a failed condition. If this is the case, DMA may not be enabled for the device or a conflict may exist with another device. For DaqBook/2000 Series, DaqBoard/2000 Series and cPCI DaqBoard/2000c Series, this command definition verifies DMA via available Input and Output speed tests, i.e., `DtstAdcFifoInputSpeed`, `DtstDacFifoOutputSpeed`, `DtstIOInputSpeed`, and `DtstIOOutputSpeed`. See the following *Performance Tests* section for more information.

Performance Tests

Performance tests measure the speed at which certain operations can be performed on the device. In general, the performance test results indicate the maximum rate at which the operation can be performed on the device. The valid performance test types are defined as follows:

`DtsAdcFifoInputSpeed` – This test will determine the maximum rate at which analog input can be acquired and transferred to system memory. Analog input performance results will be returned in the `result` parameter with units of samples/second. (Note 1).

`DtsDacFifoOutputSpeed` – (DaqBoards only) This test will determine the maximum rate at which analog output data can be read from system memory and transferred to the device's DAC FIFO. Analog output performance results will be returned in the `result` parameter with units of samples/second. (Note 1).

`DtsIOInputSpeed` – This test will determine the maximum rate at which digital input can be read from the device's DIO port and transferred to system memory. Digital input performance results will be returned in the `result` parameter with units of bytes/second. (Note 1).

`DtsIOOutputSpeed` – This test will determine the maximum rate at which digital output can be read from system memory and output to the device's DIO port. Digital output performance results will be returned in the `result` parameter with units of bytes/second. (Note 1).

Note 1: For DaqBook/2000 Series, DaqBoard/2000 Series and cPCI DaqBoard/2000c Series the results are limited to 200 kHz and are only available on the correct hardware.

Prototypes

C/C++

```
daqTest(DaqHandleT handle, DaqTestCommand command, DWORD count, PBOOL
cmdAvailable, PDWORD result);
```

Visual BASIC

```
VBdaqTest&(ByVal handle&, ByVal command&, ByVal count&, cmdAvailable&,
result&)
```

Program References

None

daqWaitForEvent

Format

    daqWaitForEvent(handle, event)

Purpose

daqWaitForEvent waits on a specific event to occur on the specified device.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle of the device for which to wait of the specified event
event	DaqTransferEvent	Specifies the event to wait on

Parameter Values

handle: obtained from the **DaqOpen** function

event: see table below

Parameter Type Definitions

daqEvent-(DaqTransferEvent)	
Definition (daqEvent)	**Description**
DteAdcData	Data is present in the acquisition buffer
DteAdcDone	Acquisition data transfer operation is complete

Returns

DerrNoError   No Error

Function Usage

The daqWaitForEvent function will not return until the specified event (event) has occurred or the wait has timed out— whichever comes first. The event time-out can be set with the function.

Note: The default timeout is 5 seconds.

Prototypes

C/C++

    daqWaitForEvent(DaqHandleT handle, DaqTransferEvent event);

Visual BASIC

    VBdaqWaitForEvent&(ByVal handle&, ByVal daqEvent&)

Program References

    DAQADCEX04.CPP, DAQADCEX05.CPP, DAQADCEX06.CPP, DAQEX.FRM (VB)

daqWaitForEvents

Format

```
daqWaitForEvents(handles, events, eventCount, eventSet, waitMode)
```

Purpose

`daqWaitForEvents` waits on specific device events to occur on the specified devices.

Parameter Summary

Parameter	Type	Description
handles	DaqHandleT	Pointer to an array of handles which represent the list of device on which to wait for the events
daqEvents	DaqTransferEvent	Pointer to an array of events which represents the list of events to wait on
eventCount	DWORD	Number of defined events to wait on
eventSet	BOOL	Pointer to an array of booleans indicating if the events have been satisfied
waitMode	DaqWaitMode	Specifies the mode for the wait

Parameter Values

`handles:` obtained from the **DaqOpen** function

`events:` see table below

`eventCount:` valid values range from 1 to 16

`eventSet:` valid values are either `true` ($\neq 0$) or `false` ($= 0$)

`waitMode:` see table below

Parameter Type Definitions

daqEvents-(DaqTransferEvent)	
Definition	**Description**
DteAdcData	Data is present in the acquisition buffer
DteAdcDone	Acquisition data transfer operation is complete

waitMode-(DaqWaitMode)	
Definition	**Description**
DwmNoWait	Do not wait. Return immediately with the current state of all events
DwmWaitForAny	Wait until any event condition has been satisfied then return current states for all events
DwmWaitForAll	Wait until all event conditions have been satisfied then return current states for all events

Returns

DerrNoError   No Error

Function Usage

This function will wait on the specified events and will return based upon the criteria selected with the `waitMode` parameter. A time-out for all events can be specified using the function. Events to wait on are specified by passing an array of event definitions in the `events` parameter. The number of events is specified with the `eventCount` parameter.

Note: The default timeout is 10 seconds.

Prototypes

C/C++

```
daqWaitForEvents(DaqHandleT *handles, DaqTransferEvent *events, DWORD
eventCount, BOOL *eventSet, DaqWaitMode waitMode);
```

Visual BASIC

```
VBdaqWaitForEvents&(handles&(), events&(), ByVal eventCount&, eventSet&(),
ByVal waitMode&)
```

Program References

None

daqZeroConvert

Format

```
daqZeroConvert (counts, scans)
```

Purpose

`daqZeroConvert` compensates one or more scans according to the previously called `daqZeroSetup` function.

Parameter Summary

Parameter	Type	Description
counts	PWORD	Raw data from one or more scans
scans	DWORD	Number of scans of raw data in the counts array

Parameter Values

`counts`: a pointer to an array ranging from 0 to 65,535

`scans`: valid values range from 1 to 4,295,967,295; however, memory limitiations may apply

Returns

DerrZCInvParam   Invalid parameter value

DerrNoError   No error

For more details on error messages refer to the Daq Error Table.

Function Usage

This function will modify the array of data passed to it.

Prototypes

C/C++

```
daqZeroConvert(PWORD counts, DWORD scans);
```

Visual BASIC

```
VBdaqZeroConvert&(counts%, ByVal scans&)
```

Program References

None

daqZeroSetup

Also See: **daqZeroConvert,daqZeroSetupConvert, daqAutoZeroCompensate**

Format

    daqZeroSetup (nscan, zeroPos, readingsPos, nReadings)

Purpose

daqZeroSetup defines which channels will be zeroed within a scan, the location of the shorted channel, the size of the scan, and the number of readings to zero.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Number of readings in a single scan
zeroPos	DWORD	Position of the zero reading within the scan
readingsPos	DWORD	Position of the readings to be zeroed within the scan
nReadings	DWORD	Number of readings immediately following the zero reading that are sampled at the same gain as the zero reading

Parameter Values

nscan: valid values range from 1 to 272
zeroPos: valid values range from 1 to 272
readingsPos: valid values range from 1 to 272
nReadings: valid values range from 1 to 270

Returns

DerrZCInvParam   Invalid parameter value
DerrNoError   No error

For more details on error messages refer to the Daq Error Table.

Function Usage

This function does not do the actual conversion. A non-zero return value indicates an invalid parameter error.

Prototypes

C/C++

    daqZeroSetup(DWORD nscan, DWORD zeroPos, DWORD readingsPos, DWORD nReadings);

Visual BASIC

    VBdaqZeroSetup&(ByVal nscan&, ByVal zeroPos&, ByVal readingsPos&, ByVal
    nReadings&)

Program References

None

daqZeroSetupConvert

Also See: **daqZeroSetup, daqZeroConvert, daqAutoZeroCompensate**

Format

```
daqZeroSetupConvert (nscan, zerroPos, readingsPos, nReadings, counts, scans)
```

Purpose

`daqZeroSetupConvert` performs both the setup and convert steps with one call.

Parameter Summary

Parameter	Type	Description
nscan	DWORD	Number of readings in a single scan
zeroPos	DWORD	Position of the zero reading within the scan
readingsPos	DWORD	Position of the readings to be zeroed within the scan
nReadings	DWORD	Number of readings immediately following the zero reading that are sampled at the same gain as the zero reading
counts	DWORD	Raw data from one or more scans
scans	DWORD	Number of scans of raw data in the counts array

Parameter Values

nscan: valid values range from 1 to 272
zeroPos: valid values range from 1 to 272
readingsPos: valid values range from 1 to 272
nReadings: valid values range from 1 to 270
counts: a pointer to an array ranging from 0 to 65,535
scans: valid values range from 1 to 4,295,967,295; however, memory limitiations may apply

Returns

DerrZCInvParam   Invalid parameter value
DerrNoError   No error

For more details on error messages refer to the Daq Error Table.

Function Usage

This is useful when the zero compensation needs to be performed multiple times because data was read from channels at different gains or from different boards.

Prototypes

C/C++

```
daqZeroSetupConvert(DWORD nscan, DWORD zeroPos, DWORD readingsPos, DWORD
nReadings, PWORD counts, DWORD scans);
```

Visual BASIC

```
VBdaqZeroSetupConvert&(ByVal nscan&, ByVal zeroPos&, ByVal readingsPos&,
ByVal nReadings&, counts%(), ByVal scans&)
```

Program References

```
None
```

API Error Codes - `daqError`

Error Name	Code # hex - dec	Description
`DerrNoError`	00h – 0	No error – No errors encountered in performing action.
`DerrBadChannel`	01h – 1	Specified LPT channel was out-of-range (Error Code Obsolete)
`DerrNotOnLine`	02h – 2	Requested device is not online - The device cannot be detected. **Corrective Actions:** If using a Book product: • Check power on the device. • Check that cabling is IEEE-1284 compliant and is securely connected to the device and the computer. (if applicable). • If using a plug in PCI or ISA parallel port, check to ensure that the plug in board is properly installed and firmly seated in the bus slot. • If using a PCMCIA (PC Card) parallel port card make sure that the card is firmly and completely inserted into the socket controller. Also make sure that the operating system properly recognizes the card as a parallel port device and that its interrupt setting has no conflicts. • Check that the device is properly configured in the Daq Configuration Applet in the Control Panel. Make sure that the device is connected to the parallel port for which it is configured. If using a Daq PC card product: • Check to ensure that the card is properly and fully inserted into the socket. • Check to make sure that the operating system recognizes the card in the Device Manager of the Control Panel; and make sure that its interrupt setting has no conflicts. • Check to ensure that the socket in which the card is installed corresponds to the socket configured in the Daq Configuration Applet in the Control Panel If using an ISA-board: • Check to ensure that the product is firmly seated into the ISA bus slot. • Check the Base Address and Interrupt settings on the board match the settings in the Daq Configuration Applet in the Control Panel If using a PCI-board: • Check to ensure that the product is firmly seated into the PCI bus slot. • Check that the operating system properly recognizes the device in the Device Manager of the Control Panel. • Check that the Serial Number on the device matches that reported by the Daq Configuration Applet in the Control Panel. • Check that the Bus and Slot number reported in the Daq Configuration Applet in the Control Panel match the physical bus and slot number in which the device is installed.
`DerrNoDaqbook`	03h – 3	Reserved for future use
`DerrBadAddress`	04h – 4	Reserved for future use
`DerrFIFOFull`	05h – 5	FIFO Full detected, possible data corruption. Input FIFO on device has oveflowed – data loss or data corruption is possible under these conditions. **Corrective Actions:** • Reduce scan rate or channel count. • If Book product select a parallel port protocol capable of higher throughput speeds.
`DerrBadDma`	06h – 6	DMA is currently being used (DaqBoard(ISA) only) by another device **Corrective Actions:** • Change DMA channel for the DaqBoard in the Daq configuration Applet in the system Control Panel to another setting not used by other devices.
`DerrBadInterrupt`	07h – 7	Bad or illegal INTERRUPT level specified for device - Interrupt could not be acquired. **Corrective Actions:** • Check system for devices configured with the same interrupt or interrupt conflicts. • If using DaqBook check system to make sure that the interrupt is enabled on the LPT port connected to DaqBook (BIOS and system settings configuration in the Control Panel)
`DerrDmaBusy`	08h – 8	DMA is currently being used (DaqBoard(ISA) only) **Corrective Actions:** • Change DMA channel for the DaqBoard in the Daq configuration Applet in the system Control Panel to another setting not used by other devices.
`DerrInvChan`	10h – 16	Invalid analog input channel - Channel number selected cannot be included in the scan group because it is an invalid channel. **Corrective Actions:** • Check channel setting to ensure that it is correlated to a real physical channel on the main unit or expansion device. • Check that the channel is being configured on the desired device. Device Handle should correspond to the device for which the channel is being configured

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex - dec	Description
`DerrInvCount`	11h - 17	Invalid count parameter – An invalid number was specified. This can be related to number of scans requested, number of channels in a scan, number of updates or scans in a buffer, number of scans requested to transfer and so on. **Corrective Actions:** • Check any API's where any channel, scan or buffer allocation is performed. • Check API's that request transferred or updated data status requests. • Check that passed count values are valid.
`DerrInvTrigSource`	12h - 18	Invalid trigger source parameter Trigger source selected is not a valid trigger source for the given device. **Corrective Actions:** • Check that the trigger source selected is a valid trigger source (see trigger source table) • Check device documentation to see if the device is capable of using the trigger source.
`DerrInvLevel`	13h - 19	Invalid trigger level parameter – Trigger level programmed is invalid. **Corrective Actions:** • Check that the trigger value programmed is appropriate for the input range of the selected device and channel.
`DerrInvGain`	14h - 20	Invalid channel gain parameter - Gain level programmed is invalid. **Corrective Actions:** • Check that the device and channel for the specified gain are capable of utilizing the gain programmed. (see gain tables)
`DerrInvDacVal`	15h - 21	Invalid DAC output parameter **Corrective Actions:** • Make sure that the programmed DAC values are within the specified range of the DAC output for the device and channel.
`DerrInvExpCard`	16h - 22	Invalid expansion card parameter – parameter cannot be used with specified expansion card or channel: **Corrective Actions:** • Do not use the illegal parameter
`DerrInvPort`	17h - 23	Invalid port parameter – Invalid DIO port reference. **Corrective Actions:** • Check that the port number programmed is valid DIO port for the device or DIO expansion card.
`DerrInvChip`	18h - 24	Invalid chip parameter (no ref)
`DerrInvDigVal`	19h - 25	Invalid digital output parameter – Digital output value not valid for output to digital port **Corrective Actions:** • Inspect all daqIO output parameters for inappropriate values for the given DIO port and device.
`DerrInvBitNum`	1Ah - 26	Invalid bit number parameter – Bit number specified is not valid for specified DIO port. **Corrective Actions:** • Check that bit number programmed is valid for specified DIO port and channel
`DerrInvClock`	1Bh - 27	Invalid clock parameter – Input or Output clock parameter or combination of clock parameters programmed is not valid. **Corrective Actions:** • Check clock parameter values • Check clock parameter flags
`DerrInvTod`	1Ch - 28	Invalid time-of-day parameter – Time of Day programmed on the 9513 chip is invalid **Corrective Actions:** • Check Daq9513TimeOfDay (tod) parameter used with daq9513SetMasterMode API.
`DerrInvCtrNum`	1Dh - 29	Invalid counter number – The counter number specified is not a valid counter number for the 9513 device. **Corrective Actions:** • Use a counter number between 1 and 5 for the 9513.
`DerrInvCntSource`	1Eh - 30	Invalid counter source parameter –The 9513 Counter Source parameter is not valid. **Corrective Actions:** • Select a defined Counter Source (see 9513 Counter Source tables)
`DerrInvCtrCmd`	1Fh - 31	Invalid counter command parameter – The 9513 Counter Command specified is not valid. **Corrective Actions:** • Specify a valid 9513 Counter Command. (see 9513 counter command tables)
`DerrInvGateCtrl`	20h - 32	Invalid counter gate parameter – The specified gate is not valid when programming counter mode. **Corrective Actions:** • Specify a valid 9513 Gate Control value. (see 9513 counter gate tables)

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex - dec	Description
`DerrInvOutputCtrl`	21h - 33	Invalid output control parameter – Bad Output Control specified when programming the 9513 for timer output. **Corrective Actions:** • Specify a valid 9513 Gate Control value.
`DerrInvInterval`	22h - 34	Invalid interval parameter – Bad interval specified when programming the 9513 for frequency measurement over the interval. **Corrective Actions:** • Specify an interval for the 9513 that can properly be used to measure a frequency over.
`DerrTypeConflict`	23h - 35	Obsolete Error Code
`DerrMultBackXfer`	24h - 36	A second background transfer was requested – Another acquisition transfer has been requested when one is already active. Only one transfer can be active on a device at any given time. **Corrective Actions:** • Wait for the first transfer to complete then start the new transfer • Disarm (daqDisarm) the first transfer so that a new transfer can be started.
`DerrInvDiv`	25h - 37	Invalid Fout divisor – foutDiv parameter for the daq9513SetMasterMode API is out of range (>15) **Corrective Actions:** • Set the foutDiv parameter ≤15
Temperature Conversion Errors		
Error Name	**Code # hex - dec**	**Description**
`DerrTCE_TYPE`	26h - 38	TC type out-of-range – An undefined TC type value has been specified. **Corrective Actions:** • Check that a valid TC type has been specified (see TC Types Table)
`DerrTCE_TRANGE`	27h - 39	Temperature out-of-CJC-range – Temperature at CJC is outside of the specified operating range of the CJC. Invalid CJC readings and corresponding temperature channel readings may be inaccurate due to CJC out of range. **Corrective Actions:** • Operate device and/or expansion units under temperature conditions which are not outside the specified range of the CJC.
`DerrTCE_VRANGE`	28h - 40	Voltage out-of-TC-range – The voltage level of the TC input on the channel is outside of the specified range of the TC configured for the channel. Temperature readings may be inaccurate due to voltage levels being outside of the specified range of the TC. **Corrective Actions:** • Check to ensure that the correct TC type is physically connected to the specified channel and is connected securely. • Check to ensure that the physical channel has been configured to the correct TC value in the application. • Check that the temperature level at the TC juncture is within the operating range of the TC.
`DerrTCE_PARAM`	29h - 41	Unspecified parameter value error – bad parameter was passed to daqTCSetup and/or daqTCConvert **Corrective Actions:** • Check to ensure that the daqTCSetup and daqTCConvert have been called with the proper parameters for the operation. • Check to ensure that daqTCSetup and daqTCConvert have not returned an error
`DerrTCE_NOSETUP`	2Ah - 42	`DaqTCConvert` called before `daqTCSetup` – TC setup has not been properly initialized – possibly due to order of precedence for these API's has not been performed correctly. `Corrective Actions:` • Check to ensure that the daqTCSetup is called before daqTCConvert. • Check to ensure that the daqTCSetup has been called with the proper parameters for the operation. • Check to ensure that daqTCSetup has not returned an error.
Device Capabilities		
Error Name	**Code # hex - dec**	**Description**
`DerrNotCapable`	2Bh - 43	Device is incapable of function – The device is not capable of performing the operation or function specified. **Corrective Actions:** • Refer to product documentation to ensure that the device is capable of performing the specified operation or function. • Check to ensure that the application is referencing the proper handle for the device and not using a handle opened for another device.

API Error Codes (Cont.) – `daqError`

Scanned Input Transfers		
Error Name	**Code # hex - dec**	**Description**
`DerrOverrun`	2Ch - 44	A buffer overrun occurred – The acquisition transfer buffer located in PC memory has overrun. When this occurs data loss and/or corruption is possible. **Corrective Actions:** • Make sure that the application has enough time to return to processing/removing data from the buffer before it overruns. • If using a Driver allocated buffer make sure that daqAdcTransferGetStat is called often to manage the transfer. • Increase the buffer size so that more processing/removing latencies can be tolerated without overrunning. • Decrease the acquisition rate and/or acquisition channel scan count.

Zero and Cal Conversion Errors		
Error Name	**Code # hex - dec**	**Description**
`DerrZCInvParam`	2Dh - 45	Unspecified parameter value error – An invalid parameter was specified during daqZeroSetup or daqZeroConvert. **Corrective Actions:** • Check the parameter settings for daqZeroSetup and daqZeroConvert.
`DerrZCNoSetup`	2Eh - 46	`Zero compensation has not been properly setup. `**`daq…Convert`**` was called before `**`daq…Setup`**` or daqZeroSetup was not properly performed` **Corrective Actions:** • Make sure that daqZeroSetup was called before daqZeroConvert. • Make sure that daqZeroSetup was called with the proper parameter definitions and settings. • Check any error codes returned from daqZeroSetup.
`DerrInvCalFile`	2Fh - 47	Cannot open the specified cal file – the calibration file specified by the calfile parameter when executing the daqReadCalFile API could not be opened. **Corrective Actions:** • Make sure that the calibration file name and path have been properly defined in the *calfile* string parameter. • Make sure that the application has the appropriate system permissions to read the file. This can be an issue under Windows/NT/2000 systems.

Environmental Errors		
Error Name	**Code # hex - dec**	**Description**
`DerrMemLock`	30h - 48	Cannot lock allocated memory from operating system. This is a rare error condition and is an operating system resource issue. **Corrective Actions:** • Shut down any applications that may have PC memory intense operations. • Increase system memory size.
`DerrMemHandle`	31h - 49	Cannot get a memory handle from operating system This is a rare error condition and is an operating system resource issue. **Corrective Actions:** • Shut down any applications which may have PC memory intense operations.

Pre-trigger acquisition Errors		
Error Name	**Code # hex - dec**	**Description**
`DerrNoPreTActive`	32h - 50	No pre-trigger configured – pre-trigger operation could not be performed since no pre-trigger has been defined. **Corrective Actions:** • Setup the acquisition for pre-trigger operation.

DAC FIFO Errors (DaqBoard only)		
Error Name	**Code # hex - dec**	**Description**
`DerrInvDacChan`	33h - 51	DAC channel does not exist. The DAC channel specified does not exist for the specified device. Channel number selected cannot be programmed because it is an invalid DAC channel. **Corrective Actions:** • Check channel setting to ensure that it is correlated to a real physical DAC channel on the main unit or expansion device. • Check that the channel is being configured on the desired device. Device Handle should correspond to the device for which the channel is being configured

Error Name	Code # hex - dec	Description
`DerrInvDacParam`	34h – 52	DAC parameter is invalid – A parameter passed to one of the DaqDac… API's was invalid.
`DerrInvBuf`	35h – 53	Buffer points to NULL or buffer size is zero when in User Buffer mode–The buffer passed by the application is un-allocated or has a bad pointer address. **Corrective Actions:** • Properly allocate the memory pointed to by the buffer address. • Check to ensure that the pointer to the buffer is properly passed to the API.
`DerrMemAlloc`	36h – 54	Could not allocate the needed memory – Memory could not be allocated by the driver. **Corrective Actions:** • Shut down other applications that may be utilizing PC system memory. • Free any unneeded dynamically allocated memory used by the application. • Increase PC system memory.
`DerrUpdateRate`	37h – 55	Could not achieve the specified update rate – could not program the unit to scan or update at the requested rate. In most cases the rate will automatically be set to the nearest achievable rate. **Corrective Actions:** • Lower the rate requested – normally, the rate requested is not achievable due to the rate being to high for the unit. • Decrease channel count – the aggregate rate can be increased by reducing the number of channels to scan. • If using a DaqBoard/2000 Series board the rate can be increased by ensuring that Compatibility Mode (100Khz) is disabled.
`DerrInvDacWave`	38h – 56	Could not start waveforms because of missing or invalid parameters - One or more of the waveform parameters is incorrect. This error will occur upon daqDacWaveArm and could indicate that one or more waveform parameters is not set properly. **Corrective Actions:** • Check parameters for API's like daqDacSetOutputMode and all the appropriate daqDaqWave… API's called to ensure that each parameter for each API is configured or set properly.
`DerrInvBackDac`	39h – 57	Could not start waveforms with background transfers – DAC waveform output transfer already active. While multiple DAC [and P3 DIO] channels can be updated concurrently during a transfer – no more than one transfer may be active at any given time for a single main unit device. **Corrective Actions:** • Wait for the first transfer to complete then start the new transfer • Disarm (daqDisarm) the first transfer so that a new transfer can be started.
`DerrInvPredWave`	3Ah – 58	Predefined waveform not supported. Waveform type passed to daqDacWaveSetPredefWave is not supported. **Corrective Actions:** • Check WaveForm Type parameter in daqDacWaveSetPredefWave to ensure that value represents a valid waveform type definition defined by the DaqDacWaveType enumeration.
RTD Conversion Errors		
Error Name	**Code # hex - dec**	**Description**
`DerrRtdValue`	3Bh – 59	RTD Value out-of-range – Error code generated if RTD value is outside of the maximum 1k range. This error code can be generated by the daqCvtRtdSetup API and indicates that the passed *rtdValue* parameter is out of range. **Corrective Actions:** • Inspect the *rtdValue* parameter for the daqCvtRtdSetup API for proper values.
`DerrRtdNoSetup`	3Ch – 60	No setup. This error can occur if the daqCvtRtdConvert API is called before the daqCvtRtdSetup API. The error indicates that there is no setup information from which a conversion can be performed. **Corrective Actions:** • Make sure that daqCvtRtdSetup API is called before the daqCvtRtdConvert API. • Make sure that the prior call to daqCvtRtdSetup returns without error.
`DerrRtdArraySize`	3Dh – 61	Obsolete Error Code
`DerrRtdParam`	3Eh – 62	Incorrect RTD parameter – This error code is generated if a bad parameter was passed to the daqCvtRtdSetup API. The error code can be generated if either the *nScan* or *nRtd* parameters are zero. **Corrective Actions:** • Check the *nScan* and *nRtd* parameters of the daqCvtRtdSetup API for inappropriate settings.

Channel Bank/Option Errors		
Error Name	**Code # hex - dec**	**Description**
`DerrInvBankType`	`3Fh - 63`	Invalid bank-type specified. This error code is returned from daqAdcExpSetChanOption and daqAdcExpSetBank if the channel type passed is either undefined or inappropriate for the bank. **Corrective Actions:** • Check the *chanType* parameter for the daqAdcExpSetBank API and the *optionType* parameter for the daqAdcExpSetChanOption API
`DerrBankBoundary`	`40h - 64`	Simultaneous writes to DBK cards in different banks not allowed.
Error Name	**Code # hex - dec**	**Description**
`DerrInvFreq`	`41h - 65`	Invalid scan frequency specified. Could not achieve the specified input scan frequency – could not program the unit to scan or update at the requested rate. In most cases the scan rate will automatically be set to the nearest achievable rate. **Corrective Actions:** • Lower the rate requested – normally, the rate requested is not achievable due to the rate being too high for the unit. • Decrease channel count – the aggregate rate can be increased by reducing the number of channels to scan. • If using a DaqBoard/2000 Series board the rate can be increased by ensuring that Compatibility Mode (100Khz) is disabled.
`DerrNoDaq`	`42h - 66`	Daq/112/216 not found (obsolete error code)
`DerrInvOptionType`	`43h - 67`	Invalid option-type parameter - An invalid option type has been specified to the daqSetOption or daqAdcExpSetModuleOption API's. The *optionType* parameter needs to be a valid value supported by the DaqOptionType definitions. **Corrective Actions:** • Check that the *optionType* parameter specified is a defined DaqOptionType. • Check documentation for your device to determine if the device is capable of setting the specified option.
`DerrInvOptionValue`	`44h - 68`	Invalid option-value parameter - An invalid option value has been specified to the daqSetOption or daqAdcExpSetModuleOption API's. The *optionValue* parameter needs to be valid for the associated *optionType* parameter specified. The *optionValue* parameter is a single precision floating point number that represents the value to be set for the option. The most common occurrence of this error is due to a value being passed that is outside the minimum and maximum allowable settings for the option being set. **Corrective Actions:** • Check that the *optionType* parameter specified is the desired option to set. • Check that the value being passed does not exceed the minimum or maximum allowable settings for the option specified. • Check documentation for your device to determine if the device is capable of setting the specified option.
`DerrTooManyHandles`	`60h - 96`	No more handles available to open. Too many open sessions are active. Many daqOpen API's have been issued without closing enough sessions (daqClose). There are no more handles available. **Corrective Actions:** • Close one or more devices (daqClose) so that a handle can become available.
`DerrInvLockMask`	`61h - 97`	Error code reserved for future use
`DerrAlreadyLocked`	`62h - 98`	Error code reserved for future use
`DerrAcqArmed`	`63h - 99`	Operation not available while an acquisition is armed. The operation must be performed when there are no current acquisitions active or pending. **Corrective Actions:** • Wait until the acquisition terminates normally • Terminate the acquisition with daqDisarm.
`DerrParamConflict`	`64h - 100`	Each parameter is valid, but the combination is invalid.
`DerrInvMode`	`65h - 101`	Invalid acquisition/wait/dac mode
`DerrInvOpenMode`	`66h - 102`	Invalid file-open mode – The file open mode *openMode* (DaqAdcOpenMode) specified in the daqAdcSetDiskFile is invalid. **Corrective Action:** Check that the value passed for the *openMode* parameter is a valid DaqAdcOpenMode type.

Error Name	Code # hex - dec	Description
`DerrFileOpenError`	67h – 103	Cannot open the specified file – the file specified by the *filename* parameter in the daqAdcSetDiskFile or the daqDacWaveSetDiskFile APIs could not be opened. **Corrective Actions:** • Make sure that the file name and path have been properly defined in the *filename* string parameter. • Make sure that the application has the appropriate system permissions to read/open the file. This can be an issue, especially under Windows/NT/2000 systems.
`DerrFileWriteError`	68h – 104	Unable to write file. The file specified by the daqAdcSetDiskFile API command could not be written or pre-written. **Corrective Actions:** • Check that the application has called daqAdcSetDiskFile API prior to arming the acquisition (daqAdcArm). • Check the return codes from daqAdcSetDiskFile API to ensure that the file was opened properly. • Make sure that the application has the appropriate system permissions to write the file. This can be an issue, especially under Windows/NT/2000 systems. • Lack of adequate space on the target drive could cause a failure to write acquired data during the acquisition or the pre-write operation (if requested). Make sure that there is enough space on the target drive and path to write the entirety of the acquisition.
`DerrFileReadError`	69h – 105	Unable to read file. The file specified by the daqDacWaveSetDiskFile API command could not be read. **Corrective Actions:** • Check that the application has called daqDacWaveSetDiskFile API prior to arming waveform output (daqDacWaveArm). • Check the return codes from daqDacWaveSetDiskFile API to ensure that the file was opened properly. • Make sure that the application has the appropriate system permissions to read the file. This can be an issue, especially under Windows/NT/2000 systems.
`DerrInvClockSource`	6Ah – 106	Invalid clock source selected – The clock source for the input or output operation is not properly set for the device. The *clockSource* parameter (set in the daqAdcSetClockSource or daqDacWaveSetClockSource APIs is not set to a valid clock source for the given device. **Corrective Actions:** • If performing a clocked input operation with daqAdcSetClockSource make sure that the *clockSource* selected is a valid clock source type (DaqAdcClockSource) for the given device. • If performing a clocked output operation with daqDacWaveSetClockSource make sure that the *clockSource* selected is a valid clock source type (DaqDacClockSource) for the given device.
`DerrInvEvent`	6Bh – 107	Invalid transfer event. The events specified by the *events* parameter in the daqWaitForEvents or daqWaitForEvent API's was not a valid transfer event type (DaqTransferEvent). **Corrective Action:** Check the *events* parameter to ensure that it is a valid DaqTransferEvent event type.
`DerrTimeout`	6Ch – 108	Time-out on wait – The wait on a single event or wait on multiple events timed-out. The event(s) specified by the daqWaitForEvents or daqWaitForEvent API's did not occur before the time-out for the events expired. The time-out for the event(s) is set via the daqSetTimeout API and is based in ms. **Corrective Actions:** • If the event(s) are expected to occur within a given finite period that can be determined then use the daqSetTimeout API to set the maximum time to wait for the event(s) to occur. • If the event(s) cannot be guaranteed to occur within a finite period then set the time-out to be infinite by setting the parameter *mSecTimeout*=0 in the daqSetTimeout API.
`DerrInitFailure`	6Dh – 109	Unexpected result occurred while initializing the hardware. This error is a general failure indicating that the driver cannot properly communicate and/or initialize the device. **Corrective Action:** Follow the procedures set forth in DerrNotOnline error code for your particular device to make sure that the device is properly powered, connected and configured.

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex - dec	Description
`DerrBufTooSmall`	`6Eh - 110`	Buffer specified is too small for the requested operation. This is a general purpose error code for operations that require a minimum buffer size. This error code is device and operation specific. **Corrective Action:** Refer to device documentation for specific buffer size requirements for the operation and device.
`DerrInvType`	`6Fh - 111`	Invalid Port/Channel type – This error indicates that an invalid request has been made to perform an operation on a port or channel which is not capable of performing the operation. **Corrective Operation:** Check the port/channel number and type to determine if the operation is appropriate for that channel/port.
`DerrArraySize`	`70h - 112`	Used as a catch all for arrays not large enough. Returned if the operation cannot be performed because the array is too small. This error code can be generated when calling daqCvtLinearConvert when the number of (*scans* x *nReadings*) > *nValues* when using moving average or when (*nReadings* > *nValues* when using block averaging. **Corrective Actions:** • Check array size definitions and parameters • If using daqCvtLinearConvert check that the *nReadings* are properly defined and that daqCvtLinearSetup has been called prior to daqCvtLinearConvert. Also make sure that *scans, nReadings* and *nValues* are properly set for the averaging mode selected.
`DerrBadAlias`	`71h - 113`	Invalid Alias Name – The device alias name does not exist or is corrupt. The name supplied or its associated device configuration is not properly stored in the operating system registry. This is normally encountered when trying to open the device(daqOpen) using the string defined by *daqName* or get the device properties (daqDevicePropsT). This problem is generally due to the device either not being configured at all or a typo in the name of the device. **Corrective Actions:** • Run the Daq Configuration Control Panel Applet. Check to see that the device appears in the device list. • If the device does not appear in the device list then Click <Add Device> and follow instructions for configuring the device. • If the device does appear in the list double-check spelling and other readable characters such as _.-><)(. Remember that the device alias names are case dependent so that the string used should match **exactly** as it appears in the device list in the Daq Configuration Applet.
`DerrInvCommand`	`72h - 114`	Invalid test command – This error occurs when calling the daqTest API with an invalid or undefined test. The *command* parameter is not a valid DaqTestCommand type. **Corrective Action:** Check the *command* parameter of the daqTest API to ensure that its value represents a valid DaqTestCommand type.
`DerrInvHandle`	`73h - 115`	Invalid device handle. The handle passed to the DaqX API is not a valid handle. The handle is created by calling the daqOpen API with the appropriately defined device name. **Corrective Actions:** • Make sure that daqOpen for the device has been invoked before calling any handle based API's for the device. • Check the returned handle value from daqOpen. If the handle is < 0 then the handle is invalid indicating that the device associated with the *daqName* parameter could not be successfully opened. (See DerrBadAlias for more details) • Check that the device has not been inadvertently closed (daqClose) prior to calling a handle based API.
`DerrNoTransferActive`	`74h - 116`	Transfer not active – An operation that requires an input or output transfer to be active was called when no transfer was currently active. This normally occurs if daqAdcTransferBufData is called without a transfer currently being active. **Corrective Action:** Make sure that daqAdcTransferStart API has been called and the transfer is currently active or was active before calling daqAdcTransferBufData.
`DerrNoAcqActive`	`75h - 117`	Acquisition not active – An operation that requires an acquisition to be active was called when no acquisition was currently active.
`DerrInvOpstr`	`76h - 118`	Invalid operation string used for enhanced triggering (WaveBook only)
`DerrDspCommFailure`	`77h - 119`	Device with DSP failed communication (WaveBook only)
`DerrEepromCommFailure`	`78h - 120`	Device with EEPROM failed communication (WaveBook only)

Error Name	Code # hex - dec	Description
`DerrInvEnhTrig`	79h - 121	Device using enhanced trigger detected invalid trigger type – The *triggerSources* array parameter of the daqAdcSetTrigEnhanced API contained an invalid enhanced trigger source type (DaqAdcTriggerSource) for the device. **Corrective Actions:** • Make sure that all trigger sources passed in the *triggerSources* array parameter are valid enhanced trigger types defined by DaqAdcTriggerSource types. • Triggering capabilities vary from device to device. Make sure that the device is capable of performing the trigger source as it has been defined.
`DerrInvCalConstant`	7Ah - 122	User calibration constant out of range
`DerrInvErrorCode`	7Bh - 123	Invalid error code – Invalid/Undefined error code encountered when *errorNum* parameter contains an undefined error code when calling daqFormatError. **Corrective Action:** Make sure that the error code number passed represents a defined error code
`DerrInvAdcRange`	7Ch - 124	Invalid analog input voltage range parameter – Indicates that an invalid *adcRange* parameter was passed. A number of APIs use the *adcRange* parameter that has the type of DaqAdcRangeT. This error indicates that at least one of these API's was passed an invalid range parameter. **Corrective Action:** Check all API's that pass a parameter of type DaqAdcRangeT. Make sure that all are passing a valid range of the type DaqAdcRangeT.
`DerrInvCalTableType`	7Dh - 125	Invalid calibration table type – An invalid calibration table type has been specified when using the daqCal… API's. The *tableType* parameter does not specify a valid table type (DaqCalTableTypeT) for specified device. **Corrective Actions:** • Inspect all daqCal… API's for invalid or undefined *tableType* parameters. Each *tableType* parameter needs to take on values defined by DaqCalTableTypeT. • Check documentation for your device to make sure that the table type requested is valid for your device.
`DerrInvCalInput`	7Eh - 126	Invalid calibration input signal selection – The calibration input signal selection was invalid. The *input* parameter did not represent a valid calibration input selection setting defined by DaqCalInputT when calling the daqCalSelectInputSignal or daqConfCalConstants API's **Corrective Actions:** • Inspect the daqCalSelectInputSignal and the daqConfCalConstants API's to ensure that only valid DaqCalInputT types are passed to the *input* parameter. • Check documentation for your device to make sure that the daqCalSelectInputSignal and the daqConfCalConstants API's are valid for your device.
`DerrInvRawDataFormat`	7Fh - 127	Invalid raw-data format selection – Indicates an invalid raw data type selected if the *flags* settings in daqSetTriggerEvent indicate an invalid raw type for the trigger channel. This will be the case if the trigger channel is digital and the *flags* indicate Signed raw data. This error may also occur if setting the *rawFormat* in the daqAdcSetDataFormat to a type not defined (DaqAdcRawDataFormatT) **Corrective Actions:** • If using the daqSetTriggerEvent API and setting the trigger channel to be digital channel then set the *flags* parameter to use Unsigned raw data. • If using the daqAdcSetDataFormat API then make sure that the value passed in the *rawFormat* parameter is a valid type defined by DaqAdcRawDataFormatT
`DerrNotImplemented`	80h - 128	Feature/function not implemented yet – The requested feature or function is not yet implemented. No corrective action available.
`DerrInvDioDeviceType`	81h - 129	Invalid digital I/O device type – An invalid Digital I/O device type was passed to a daqIO… API. The *devType* parameter for a digital IO API is not properly defined as a DaqIODeviceType type. **Corrective Action:** Inspect all daqIO… API's and digital I/O API's that use the *devType* parameter for values that are not defined by the DaqIODeviceType type.
`DerrInvPostDataFormat`	82h - 130	Invalid Post Data Format – Indicates an invalid post processing data type when using the daqAdcSetDataForma API. The *postProcFormat* parameter in the daqAdcSetDataFormat is set to a type not defined by DaqAdcPostProcDataFormatT **Corrective Action:** If using the daqAdcSetDataFormat API then make sure that the value passed in the *postProcFormat* parameter is a valid type defined by DaqAdcPostProcDataFormatT

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex-dec	Description
`DerrDaqStalled`	83h – 131	*PersonalDaq Only*. The low level driver has stalled in an attempt to continue collecting data. **Corrective Actions:** • If using the PersonalDaq check that the unit is still properly connected to the PC and that all USB cables and hubs are properly connected.
`DerrDaqLostPower`	84h – 132	*PersonalDaq Only*. The PersonalDaq does not have enough power to operate properly. The unit either does not have enough power supplied to it by the host pc or the auxiliary power is not connected. Some PC's and notebooks may not supply the appropriate power necessary for the PersonalDaq to operate properly. **Corrective Actions:** • Some PC's and notebooks may not supply the appropriate power necessary for the PersonalDaq to operate properly and you may need to operate the PersonalDaq with the auxiliary power connection attached. Check the manual for instructions on using the auxiliary power input. • Another way to circumvent this problem is to place an external USB hub between the host computer and the PersonalDaq. The placement of the USB hub allows the PersonalDaq to run off power supplied by the hub rather than the host PC.
`DerrDaqMissing`	85h – 133	*PersonalDaq Only*. The PersonalDaq is missing. This occurs when a session is open with a PersonalDaq but the PersonalDaq has been found to be missing. **Corrective Actions:** • Check the USB cable connections on the PC, PersonalDaq as well as any USB hubs in use.
`DerrScanConfig`	86h – 134	The scan configuration programmed is not legal. One or more channel configuration parameters have been found to be in error. **Corrective Actions:** • Check that all channels have been configured using legal parameters and settings for that channel type. Inspect channel parameter settings and flags for each channel being configured using the daqAdcSetScan or daqAdcRd… API functions.
`DerrInvTrigSense`	87h – 135	Invalid trigger sense specified. The trigger sense specified for a hardware based trigger (either TTL or Analog) is not allowed for the current device. **Corrective Actions:** • Check that your device is capable of the using the trigger sense provided. For instance, some devices can use TTL Above and Below Level while not being available on other devices.
`DerrInvTrigEvent`	88h – 136	Invalid trigger event specified. The trigger event specified is not available for the device being configured. • Check that the value being passed is a legal value based upon trigger sense values in the header file for the programming language which you are using. **Corrective Actions:** • Check that your device is capable of the using the trigger sense provided. For instance, some devices can use Software Analog Level triggers while not being available on other devices. • Check that the value being passed is a legal trigger Event which is defined in the header file for the programming language which you are using.
`DerrInvTrigChannel`	89h – 137	Invalid trigger channel specified. The trigger channel specified for a trigger Event is not valid. **Corrective Actions:** • Check that the trigger channel provided is a valid channel number. • Check that the trigger channel provided is currently configured in the scan list. See daqAdcSetScan for details on channel scan list configuration
`DerrDacWaveformNotActive`	8Ah – 138	An invalid waveform operation has been issued while a waveform operation is not currently active. This may occur if the application issues an API which is only valid while waveform output is currently active. **Corrective Actions:** • Typically this error will be issued when a waveform API such as *daqDacArm* function is issued while a waveform output is currently active. The application should refrain from making such calls during an active waveform output operation.

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex-dec	Description
`DerrDacWaveformActive`	8B – 139	An invalid waveform operation has been issued while a waveform operation is currently active. This may occur if the application issues an API which is only valid while waveform output is **not** currently active. **Corrective Actions:** Typically this error will be issued when a waveform API such as *daqDacTransferGetStat* function is issued while a waveform output is not currently active. The application should refrain from making such calls when a waveform is not active.
`DerrDacNotEnoughMemory`	8Ch – 140	There is not enough memory to download the requested static waveform. **Corrective Actions:** • Static waveforms are limited to 256,000 total update samples across all output channels. Change your static waveform to use a smaller number of updates or reduce the number of DAC channels for which you are outputting waveforms.
`DerrDacBufferNotEqual`	8Dh – 141	The specified dynamic waveform buffers for each output channel are not of equal size. In dynamic waveform mode the waveforms for each output channel must be exactly the same size. **Corrective Actions:** • Check that the output for each channel of waveform output is exactly the same size.
`DerrDacBufferTooSmall`	8Eh – 142	The specified dynamic waveform buffer is too small to output. The waveform cannot be updated with a buffer less than 4096 update samples. **Corrective Actions:** • Check that the output buffer is greater than 4096 when using dynamic waveform output mode.
`DerrDacBufferUnderrun`	8Fh – 143	The specified dynamic waveform output operation has underrun. A waveform output underrun occurs when the DAC is able to output data faster than the controlling application can update the dynamic waveform buffers. **Corrective Actions:** • Update the waveform buffer in a more timely manner. • Increase the size of the waveform buffer to allow greater latency if the application cannot update the buffer quickly enough • Decrease the DAC output update rate.
`DerrDacPacerOverrun`	90h – 144	The specified dynamic waveform output operation has experienced a pacer clock overrun. A pacer clock overrun will occur if the clock pulses pacing the outputs from the DAC's occur at less than 10us intervals. Normally, this should only occur if clocking the outputs from an external source or the ADC pacer clock since the DAC pacer clock is limited to 10us intervals. **Corrective Actions:** • The only way to avoid this condition is to make sure that the clock pacing the DAC outputs never runs at less than 10us intervals.
`DerrDacPacerOverrun`	91h – 145	The specified acquisition operation has experienced a pacer clock overrun. A pacer clock overrun will occur if the clock pulses pacing the acquisition occur at less than 5us intervals. Normally, this should only occur if clocking the acquisition from an external source since the internal ADC pacer clock is limited to 5us intervals. **Corrective Actions:** • The only way to avoid this condition is to make sure that the clock pacing the acquisition never runs at less than 5us intervals.
`DerrAdcNotReady`	92h – 146	N/A
`DerrArbitrationFailure`	93h– 147	The board could not be communicated with properly on the PCI bus. This error indicates a critical condition that may cause the board to function improperly. **Corrective Actions:** • Move the board to be another bus or slot number. • Check BIOS settings to ensure that DMA is enabled on the bus and slot number in which the board is installed.
`DerrDacWaveFileTooSmall`	94h – 148	The specified dynamic waveform output file is too small to perform the desired waveform output. **Corrective Actions:** • Increase the size of the waveform output file.

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex-dec	Description
`DerrDacBufferUnderrun`	95h – 149	The specified dynamic waveform output operation has underrun. A waveform output underrun occurs when the DAC is able to output data faster than the controlling application can update the dynamic waveform buffers. **Corrective Actions:** • Update the waveform buffer in a more timely manner. • Increase the size of the waveform buffer to allow greater latency if the application cannot update the buffer quickly enough • Decrease the DAC output update rate.
`DerrDacWaveModeConflict`	96h – 150	Conflicting waveform output modes have been specified. Mixing of static and dynamic waveform modes is not allowed for concurrent waveform output from different DAC channels. **Corrective Actions:** • Change all DAC waveform channels to be either dynamic or static waveform output modes.
`DerrTedsInfoInvStatus`	97h – 151	*WaveBook Series Only* - Invalid TEDS status operation. No TEDS support can be found on this channel. **Corrective Actions:** • Check that a TEDS capable transducer is connected to the channel input • Check that the application intends to use a TEDS capable channel.
`DerrAlreadyCreated`	98h – 152	*Device already created* – Attempting to create a device which already exists. Duplicate device entries are not permitted. **Corrective Actions:** • Check the device name for which you are creating. Make sure device name does not already exist. Device names and settings can be checked via the DaqX control panel applet.
`DerrParentNotCreated`	99h – 153	*Device Parent Not Created* – Attempting to create a device whose specified parent device does not exist. **Corrective Actions:** • Create the parent device for the device you are attempting to create. • Check the parent device name for the device for which you are creating. Make sure that the parent device name already exists and is the device name which you are attempting to create a child from.. Device names and settings can be checked via the DaqX control panel applet.
`DerrUnableToCreateDevice`	9Ah – 154	*Unable to create specified device.* Unable to create the specified device due to registry path or access issues. **Corrective Actions:** • Make sure that the application and User privileges for the PC has access rights to write registry locations in the HKLM registry key.
`DerrUnableToDeleteDevice`	9Bh – 155	*Unable to Delete specified device.* Unable to delete the specified device due to registry path or access issues. **Corrective Actions:** • Make sure that the application and User privileges for the PC has access rights to write registry locations in the HKLM registry key.
`DerrInvIPAddress`	9Ch – 156	*Invalid IP Address* – IP Address specified was zero which is an invalid IP address setting. **Corrective Actions:** • Specify a proper, non-zero, IP address.
`DerrOutputEnableConflict`	9Dh – 157	*Output Enable Conflict* – A DaqBoard/500 device has an output conflict. Timer1 and AdcClockOutput are in conflict. **Corrective Actions:** • Disable Timer1 or disable AdcClockOutput. Both cannot be enabled at the same time.
`DerrCalDataTypeMismatch`	9Eh – 158	*Calibration Data Type Mismatch* – Calibration points do not match the data type. **Corrective Actions:** • Verify the data type and enter new calibration points of the same data type.
`DerrDacFifoUnderrun`	9Fh – 159	*DAC Fifo Underrun* – Data is not being sent fast enough. **Corrective Actions:** • Decrease the waveform playback speed • Increase the buffer size

API Error Codes (Cont.) – `daqError`

Error Name	Code # hex-dec	Description
`DerrSetpointCountMismatch`	A0 - 160	*Setpoint Count Mismatch* – The number of Setpoints does not equal the number of channel setpoint flags set. **Corrective Actions:** • Correct the mismatch situation. Each Setpoint must have one associated channel setpoint flag set.
`DerrSetpointLevelInvalid`	A1 - 161	*Setpoint Level Invalid* – The Setpoint Level is outside the channel's range. **Corrective Actions:** • Change the Setpoint Level and/or the associated channel's range such that the Setpoint Level falls within the channel's range.
`DerrSetpointOutputType`	A2 - 162	*Setpoint Output Type Invalid* – The Setpoint Output Type is invalid. **Corrective Actions:** • Set an appropriate output type for the setpoint.
`DerrSetpointOutputValue`	A3 - 163	*Setpoint Output Value Invalid* – The Setpoint Output Value is outside the channel's range. **Corrective Actions:** • Change the Setpoint Output value and/or the associated channel's range such that the setpoint output value falls within the channel's range.
`DerrSetpointLimits`	A3 - 164	*Setpoint Comparison* – Error flags when Limit B is greater than Limit A. For outside or inside window limits (`DsfOutsideLimits`, `DsfInsideLimits`) and for hysteresis (`DsfHysteresis`) Limit A must be greater than Limit B. **Corrective Actions:** • Ensure that the value for Limit A is greater than the value for Limit B.

This page is intentionally blank.

Appendix A was removed due to outdated material.

Appendix B – Using Borland C++   B

This appendix illustrates beginning a project with 32-bit Borland C++ Builder V5.0. Subsequent or previous versions of Borland C++ may require changes or modifications to these procedures. In this case, please consult Borland C++ documentation. However, though the project build procedures may differ, other 32-bit versions of Borland C++ (4.0, 6.0 or later) should function as well.

Note: The **DaqX.DLL** has been developed and built using Microsoft Visual C++. However, a Borland C++ compatible export library **BCB5DaqX.LIB** is available.

Note: BCB5DaqX.LIB and other language support files and examples are located in:
        <InstallDirectory>\DaqX\Programming Language Support\daq\C\32-bit Enhanced API\...

To begin your first project, perform the following**:**

1)   Launch Borland C++ IDE.

2)   Under the **File** menu, select **New**.

3)   Select the project type that best meets your needs; or if building an existing DaqX example, select **Console Wizard** and use the following settings:

- **Source Type**: C++
- **Use VCL**: NO
- **Multi Threaded**: NO
- **Console Application**: YES
- **Specify project source**: YES
- Select the DaqX example file of interest.

4)   Under the **Project** menu, select **Add to Project.**

5)   Add **..\lib\BCB5DaqX.LIB** to the project by browsing to the DaqX Programming Language Support directory described above.

6)   If using an existing DaqX example, include the file **..\include\DaqRoutines.CPP**

     Note: _kbhit() may not be defined; use **kbhit()** instead.

7)   If creating a new **.cpp** file, place an include statement for **DaqX.H** before any references to the DaqX API functions.

8)   Under **Compiler Options**, set the **"Pre-compiled header"** option to **"None"**

9)   **IMPORTANT!** Under **Compiler Options**, set the "**Treat enum types as ints**" option to **true**.

10) **Save** the project.

Notes

Custom OEM Installation   C

This appendix consists of two parts. Both outline the procedures required for custom reseller hardware and driver installation and distribution; and both include the names of driver files and their locations.

- o   Part 1 applies to USB devices operating under a Windows 2000 or Windows XP operating system.

- o   Part 2 applies to non-USB devices operating under a Windows 9x, Me operating system, or under a Windows NT/2000/XP system, with device exceptions as noted.

Part 1 – USB Devices Operating under Windows 2000 or XP

Applies to:

Personal Daq/3000 Series devices

DaqBoard/3000USB Series devices

The following files are required for installation by OEM.

```
PDaq3K.inf    (requested when the hardware is detected)
DaqClIn.dll
DaqCoIn.dll
PDaq3KLD.sys
PDaq3K.sys
DaqX.dll
```

File Installation is automated for the Personal Daq/3000 and DaqBoard/3000USB Series devices on plug-and-play operating systems.

Part 2 – Non-USB Devices Operating under Windows 9x, Me, or Windows NT, 2000, XP Systems

Installation by OEM excludes TempBook, Daq PC-Card, Personal Daq, WaveBook, WBK20, and DBK35.

This section applies to non-USB devices with exclusions as noted above. The following files are required for installation by OEM.

WIN 9x/Me	WIN NT/2000/XP	WIN 9x/Me and WIN NT/2000/XP

For All Devices
`daqRes.vxd`
 	
For DaqBoard/2000 Series and DaqBook/2000 Series devices
`DaqFind.vxd`
	For All Devices
	`daqRes.sys`
		For All Devices
		`DaqX.cpl`
		`DaqX.dll`
		`DaqComp.dll`
		(if using compatibility layer)

Device Series	Files for Windows System Drivers		

DaqBoard/500	**WIN 2000/XP**
	`DaqBd50x.inf`
	`DaqBd50x.sys`		

DaqBoard/1000
DaqBoard/2000
	WIN 9x/Me
	`DaqX.inf`
	`Daq2k0.vxd`
	`Daq2k1.vxd`
	`Daq2k2.vxd`
	`Daq2k3.vxd`
		WIN 2000/XP
		`DaqXNT.inf`
		`DaqBrd2k.sys`
			WIN NT
			`DaqBrd2k.sys`

DaqBoard/3000		**WIN 2000/XP**
		`DaqXNT.inf`
		`DaqBrd2k.sys`	

DaqBook/2000 **DaqLab/2000** **DaqScan/2000**	**WIN NT/2000/XP** `DaqBk2k.sys`		

DaqBook/2000A	**WIN NT/2000/XP**
	`DaqBk2k.sys`
		WIN 9x/Me
		`DaqBk2k0.vdx`
		`DaqBk2k1.vdx`
		`DaqBk2k2.vdx`
		`DaqBk2k3.vdx`	

ISA-Type DaqBoards	**WIN 9x**
	`DaqBrd0.vxd`
	`DaqBrd1.vxd`
	`DaqBrd2.vxd`
	`DaqBrd3.vxd`
		WIN NT/2000/XP
		`DaqBrd.sys`	

DaqBook/100
DaqBook/200
	WIN 9x
	`DaqBk0.vxd`
	`DaqBk1.vxd`
	`DaqBk2.vxd`
	`DaqBk3.vxd`
		WIN NT/2000/XP
		`DaqBk.sys`	

Installation Locations:

Place all .DLLs and .CPLs into C:\Windows\System (9X/Me) or C:\Windows\System32 (NT/2000/XP)

Place all .VxDs into C:\Windows\System (9X/Me Only)

Place all .SYSs into C:\Windows\System32\Drivers (NT/2000/XP Only)

You must reboot the computer and then configure the device via the Daq Configuration applet in the Control Panel. Refer to the applicable device user's manual for more information.

PDF versions of the manuals are included on the data acquisition CD and can be accessed from the CD's intro-screen using the <View PDFs> button.

File Installation is automated via the .INF files for the DaqBoard/500, /1000, and /2000 Series Boards under plug-and-play operating systems.

File Installation is automated for the Personal Daq/3000 and DaqBoard/3000USB Series devices on plug-and-play operating systems.

This page is intentionally blank.

Appendix D   daq9513… Commands   D

> **Note!**
>
> This appendix only applies to product versions that make use of a 9513 counter-timer chip.
>
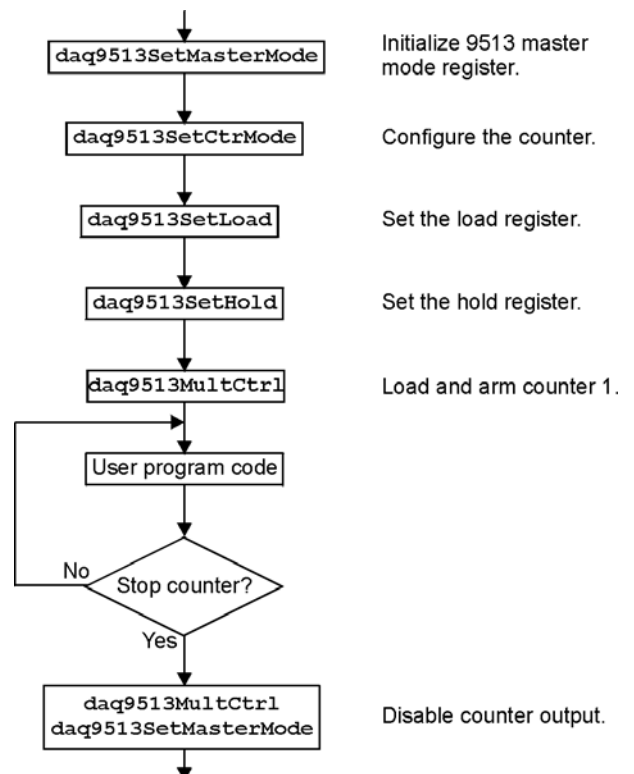> The 9513 counter-timer chip devices include the following:
> - **DaqBook/100 Series**
> - **DaqBook/200 Series**
> - **DaqBoard/100 Series**
> - **DaqBoard/200 Series**

API Programming Model for 9513 Counter-Timer Chip Devices

Variable Rate, Variable Duty-Cycle Square-Wave Output

This section demonstrates the use of the counter/timer section of a DaqBook/100/200 or of a DaqBoard/100A/200A with the P3 port. After configuring the counter and setting the load and hold registers, the counter is armed. At this point, program execution continues while the counter outputs the signal. This example generates a variable rate, variable duty-cycle square wave. Functions used include:

- `Vbdaq9513SetMasterMode&(handle&,deviceType& ,whichDevice&,foutDiv&, cntSource&, comp1&, comp2&, tod&)`

- `Vbdaq9513SetCtrMode&(handle&,deviceType&,wh ichDevice&, ctrNum&,gayeCtrl&, cntEdge&, cntSource&, specGate&, reload&, cntRepeat&, cntType&, cntDir&, outputCtl&)`

- `Vbdaq9513SetHold&(handle&, deviceType&,whichDevice&, ctrNum&, ctrVal%)`

- `Vbdaq9513SetLoad&(handle&,deviceType&,which Device&, ctrNum&, ctrVal%)`

- `Vbdaq9513MultCtrl&(handle&,deviceType&,whic hDevice&, ctrCmd&, ctr1&, ctr2&, ctr3&, ctr4&, ctr5&)`

`daq9513SetMasterMode`	Initialize 9513 master mode register.
`daq9513SetCtrMode`	Configure the counter.
`daq9513SetLoad`	Set the load register.
`daq9513SetHold`	Set the hold register.
`daq9513MultCtrl`	Load and arm counter 1.
User program code	
Stop counter? No / Yes	
`daq9513MultCtrl` `daq9513SetMasterMode`	Disable counter output.

Initialize the 9513 master mode register fout divider: 10, fout source: DcsF2 (100 kHz), compare1: no, compare 2: no, time of day disabled. This will place a 10 kHz pulse on the oscillator output. The **daq9513SetMasterMode** function will initialize the counter/timer section and configure several of its parameters. This is a system-wide function that affects all 5 counter timers.

Aside from initializing the counter/timer section, this application does not use most of the capabilities of the **daq9513SetMasterMode** function. The first two arguments in this function select a clock source for the fout signal found on connector P3, then select a divider for that signal. F2 in this application is a fixed, internal frequency source of 100 kHz. Our example divides this fixed frequency by 10 yielding a signal on fout of 10 kHz.

```
ret& = VBdaq9513SetMasterMode&(handle&, DiodtLocal9513&, 0, 10, DcsF2&, 0, 0,
DtodDisabled&)
```

The **daq9513SetCtrMode** function configures an individual counter in the 9513. The first argument specifies the counter to be configured; the second argument specifies the internal operation of the gate control. Our application does not use the gate, so it is disabled. The fixed 100 kHz internal clock (F1) is used as the source. By setting the reload parameter to 1, the counter will use the 'load' register and the 'hold' register to generate the pulse train. When the counter is armed, the 'load' register value is loaded then decremented on every edge of the F1 clock. The output signal will be high during this phase. When the terminal count is reached, the 'hold' register is loaded then decremented on every edge of the F1 clock. The output signal is low during this phase. If the reload argument is set to 0, only the 'load' register is used, always yielding a 50% duty-cycle pulse train. The **cntRepeat** argument specifies whether the pulse train should execute once or repeat continuously. The counter interprets the load and load register as either binary or BCD, depending on the value of the **cntType** argument. The **cntDir** specifies whether the internal counter should count up or down to reach the terminal count. A value of 5 counted down has the same effect as a value of 65,530 counted up.

```
ret& = VBdaq9513SetCtrMode&(handle&, DiodtLocal9513&, 0, 1, DgcNoGating&, 1,
  DcsF1&, 0, 1, 1, 0, 0, DocTCToggled&)
```

Set the load register to 75 and the hold register to 25. This produces a high duty-cycle of 75% and (with 100 total counts to count down) a frequency of 10 kHz.

```
' Load the load register: 75 low counts & hold register with 25 counts
ret& = VBdaq9513SetLoad&(handle&, DiodtLocal9513, 0, 1, 75)
ret& = VBdaq9513SetHold&(handle&, DiodtLocal9513, 0, 1, 25)
```

The daq9513MultCtrl function will arm counter 1.

```
ret& = VBdaq9513MultCtrl&(handle&, DiodtLocal9513&, 0, DmccLoadArm&, 1, 0,
  0, 0, 0)
```

Continue the pulse train until user terminates it.

```
Print "A 10Khz 25% duty cycle square wave is on the counter 1 output.":
 Print
MsgBox "Click to halt counter 1 output.", , "Counter 1"
    ' Halt all output
ret& = VBdaq9513MultCtrl&(handle&, DiodtLocal9513&, 0, DmccDisarm&, 1, 0,
  0, 0, 0)
ret& = VBdaq9513SetMasterMode&(handle&, DiodtLocal9513&, 0, 0, DcsF2&, 0, 0,
  DtodDisabled&)
Print "Outputs disabled."
```

daq9513GetHold

Format

```
daq9513GetHold(handle, deviceType, whichDevice, ctrNum, ctrVal)
```

Purpose

`daq9513GetHold` reads the hold register of the specified counter.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which to get the 9513 hold register
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
ctrNum	DWORD	The counter number
ctrVal	PWORD	Variable which stores the value read from the hold register of the selected counter

Parameter Values

handle: obtained from the `daqOpen` function.

deviceType: must be set to value `DiodtLocal9513`

whichDevice: valid value for all current devices is 0

ctrNum: valid values range from 1 to 5

ctrVal: pointer to a varible in which the contents of the hold register will be placed; valid values for the hold register range from 0 to 65,535

Returns

DerrInvCtrNum	Invalid counter
DerrNotCapable	No 9513 available
DerrNoError	No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

The hold register read by `daq9513GetHold` is used in event-counting applications to store counter values accumulated by the `daq9513MultCtrl` function. `daq9513GetHold` can read the hold register while the count process is running without interrupting the process.

Prototypes

C/C++

```
daq9513GetHold(DaqHandleT handle, DaqIODeviceType deviceType, DWORD
whichDevice, DWORD ctrNum, PWORD ctrVal);
```

Visual BASIC

```
VBdaq9513GetHold&(ByVal handle&, ByVal deviceType&, ByVal whichDevice&,
ByVal ctrNum&, ctrVal%)
```

Program References

None

---

daq9513MultCtrl

Format

```
daq9513MultCtrl(handle, deviceType, whichDevice, ctrCmd, ctr1, ctr2, ctr3,
ctr4, ctr5)
```

Purpose

daq9513MultCtrl can arm or disarm specified counters, can save data from the specified counters to the load and/or hold register, or can load data from the load and/or hold register to the specified counters; each parameter can be activated for multiple counters simultaneously.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which multiple counter commands will be activated
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
ctrCmd	Daq9513MultCtrCommand	The counter command
ctr1	BOOL	A flag that, if true, enables the counter command to be executed on counter 1; if false, it does nothing to counter 1
ctr2	BOOL	A flag that, if true, enables the counter command to be executed on counter 2; if false, it does nothing to counter 2
ctr3	BOOL	A flag that, if true, enables the counter command to be executed on counter 3; if false, it does nothing to counter 3
ctr4	BOOL	A flag that, if true, enables the counter command to be executed on counter 4; if false, it does nothing to counter 4
ctr5	BOOL	A flag that, if true, enables the counter command to be executed on counter 5; if false, it does nothing to counter 5

Parameter Values

handle: obtained from the daqOpen command

deviceType: must be set to value DiodtLocal9513

whichDevice: valid value for all devices is 0

ctrCmd: see table below

ctr1-ctr5: valid values are either true ($\neq 0$) or false ($= 0$).

Parameter Type Definitions

ctrCmd—(Daq9513MultCtrCommand)	
Definition	**Description**
DmccArm	Enable counters to start
DmccLoad	Load initial counter values from either load or hold register
DmccLoadArm	Perform initial loading and enable counting
DmccDisarmSave	Disable counters and save counter value
DmccSave	Transfer current counter value to hold register
DmccDisarm	Halt counting

Returns

DerrInvCtrCmd	Invalid counter command
DerrNotCapable	No 9513 available
DerrNoError	No error

For more details on error messages, please refer to the Daq Error Table.

Prototypes

C/C++

```
daq9513MultCtrl(DaqHandleT handle, DaqIODeviceType deviceType, DWORD
whichDevice, Daq9513MultCtrCommand ctrCmd, BOOL ctr1, BOOL ctr2, BOOL
ctr3, BOOL ctr4, BOOL ctr5);
```

Visual BASIC

```
VBdaq9513MultCtrl&(ByVal handle&, ByVal deviceType&, ByVal whichDevice&,
ByVal ctrCmd&, ByVal ctr1&, ByVal ctr2&, ByVal ctr3&, ByVal ctr4&, ByVal
ctr5&)
```

Program References

```
9513EX01.CPP, 9513EX01.FRM (VB)
```

daq9513SetAlarm

Format

daq9513SetAlarm(handle, deviceType, whichDevice, alarmNum, alarmVal);

Purpose

daq9513SetAlarm sets the specified alarm register.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device from which to get 9513 frequency
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
alarmNum	DWORD	The alarm register number
alarmVal	DWORD	The value to write to the selected alarm register

Parameter Values

handle: obtained from the daqOpen function
deviceType: must be set to value DiodtLocal9513
whichDevice: valid value for all devices is 0
alarmNum: valid values range from 1 to 2
alarmVal: valid values range from 0 to 65,535

Returns

DerrInvCtrNum   Invalid counter number
DerrNotCapable   No 9513 available
DerrNoError   No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

The alarm register set by daq9513SetAlarm is only used if the corresponding comparator has been enabled using the daq9513SetMasterMode function. The alarm register can then be used with the comparators described in the entry for daq9513SetMasterMode.

Prototypes

C/C++

daq9513SetAlarm(DaqHandleT handle, DaqIODeviceType deviceType, DWORD whichDevice, DWORD alarmNum, DWORD alarmVal);

Visual BASIC

VBdaq9513SetAlarm&(ByVal handle&, ByVal deviceType&, ByVal whichDevice&, ByVal alarmNum&, ByVal alarmVal&)

Program References

None

daq9513SetCtrMode

Also See: **daq9513SetHold,daq9513MultCtrl**

Format

```
daq9513SetCtrMode (handle, deviceType, whichDevice, ctrNum, gateCtr1,
cntEdge, cntSource, specGate, reload, cntRepeat, cntType, cntDir, outputCtrl)
```

Purpose

daq9513SetCtrMode sets the 9513's mode register for a specified counter. Setting this register defines how the specific counter works for a variety of square waves, pulse generation, and event counting.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device for which the 9513 counter mode will be set
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
ctrNum	DWORD	The counter number
gateCtrl	Daq9513GatingControl	The gating control mode
cntEdge	BOOL	If true, will count on a rising count edge; if false, it will count on a falling count edge
cntSource	Daq9513CountSource	Count source
specGate	BOOL	If true, will enable the special gate; if false, it will disable the special gate
reload	BOOL	If true, will reload from load or hold registers; if false, it will reload only from load
cntRepeat	BOOL	If true, will count repetitively; if false, it will count once
cntType	BOOL	If true, will select a BCD count; if false, it will select a binary count
cntDir	BOOL	If true, will count up; if false, it will count down
outputCtrl	Daq9513OutputControl	Output control mode

Parameter Values

handle: obtained from the daqOpen function
deviceType: must be set to value DiodtLocal9513
whcihDevice: valid value for all devices is 0
ctrNum: valid values range from 1 to 5.
gateCtrl: see table below
cntEdge: valid values are true ($\neq 0$) or false ($= 0$)
cntSource: see table below
specGate: valid values are true ($\neq 0$) or false ($= 0$)
reload: valid values are true ($\neq 0$) or false ($= 0$)
cntRepeat: valid values are true ($\neq 0$) or false ($= 0$)
cntType: valid values are true ($\neq 0$) or false ($= 0$)
cntDir: valid values are true ($\neq 0$) or false ($= 0$)
outputCtrl: see table below

Parameter Type Definitions

gateCtrl-(Daq9513GatingControl)	
Definition	**Description**
DgcNoGating	Gating disabled
DgcHighTCNM1	Active level high of TC toggled output of previous (N-1) counter
DgcHighLevelGateNP1	Active level high of gate next (N+1) counter
DgcHighLevelGateNM1	Active level high of gate previous (N-1) counter
DgcHighLevelGateN	Active level high of gate of selected (N) counter
DgcLowLevelGateN	Active level low of gate of selected (N) counter
DgcHighEdgeGateN	Active rising edge of gate of selected (N) counter
DgcLowEdgeGateN	Active falling of gate of selected (N) counter

cntSource-(Daq9513CountSource)	
Definition	**Description**
DcsTcnM1*	TC toggled output of previous (N-1) counter
DcsSrc1	Counter 1 input (pin36 of P3)
DcsSrc2	Counter 2 input (pin19 of P3)
DcsSrc3	Counter 3 input (pin17 of P3)
DcsSrc4	Counter 4 input (pin15 of P3)
DcsSrc5	Counter 5 input (pin13 of P3)
DcsGate1	Counter 1 gate (pin37 of P3)
DcsGate2	Counter 2 gate (pin18 of P3)
DcsGate3	Counter 3 gate (pin16 of P3)
DcsGate4	Counter 4 gate (pin14 of P3)
DcsGate5**	Counter 5 gate (pin12 of P3)
DcsF1**	Onboard 1 MHz Clock
DcsF2**	Onboard 100 kHz Clock
DcsF3**	Onboard 10 kHz Clock
DcsF4**	Onboard 1 kHz Clock
DcsF5**	Onboard 100 Hz Clock
*invalid with daq9513SetMasterMode or daq9513RdFreq	
**invalid with daq9513RdFreq	

outputCtrl-(Daq9513OutputControl)	
Definition	**Description**
DocInactiveLow	Inactive – Always low
DocHighTermCntPulse	High impulse on terminal count
DocTCToggled	Toggled on terminal count
DocInactiveHighImp	Inactive w/ high impedance
DocLowTermCntPulse	Low impulse on terminal count

Returns

DerrInvCtrNum	Invalid channel
DerrInvGateCtrl	Invalid gate
DerrInvCntSource	Invalid source
DerrInvOutputCtrl	Invalid output
DerrNotCapable	No 9513 available
DerrNoError	No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

Input/Output Parameters

The gate control (`gateCtrl`) parameter dictates how the counter will use its gate input (P3 pins 37, 18, 16, 14 and 12) or another counter's gate input. Possible settings are as follows:

- If the gate is disabled using the `DgcNoGating` definition, it will be ignored and the counter will run as long as it is armed.
- If a level gate control is selected (using the `DgcHighLevelGateNPI`, `DgcHighLevelGateNMI`, `DgcHighLevelGateN`, or `DgcLowLevelGateN` definitions), the counter will operate only while armed and the selected high or low level is applied to the gate.
- If an edge-sensitive gate control is selected using the `DgcHighEdgeGate` or `DgcHighEdgeGateN` definitions, the counter will operate after a rising or falling edge is detected on the gate input.

Most gate control modes select gate N (gate of the selected counter) or gate inputs of the previous (N-1) and next (N+1) counters. Thus, counter 3 could use the gate input of counter 2 by selecting N-1, counter 4 by selecting N+1, or its own gate input by selecting N. Counter 1 and counter 5 are considered adjacent when selecting gate input N+1 or N-1. The final gate control mode allows the TC-toggled output (see the following description of the output control parameter) of the previous counter (N-1) to be the gate. The selected counter will operate only when the previous counter's TC-toggled output is high.

The output control (`outputCtrl`) parameter controls the state of the counter output (P3 pins 35, 34, 33, 32, 31). There are 2 inactive and 3 active output modes. If inactive, the output can be driven to low impedance, or increased to high impedance. The active modes are all associated with the terminal count (TC) which is the moment in time when the counter reaches 0. This can happen by counting above 65,535 in binary count mode (9,999 in BCD count mode) or counting down below 1. The output can be either driven high during the TC and low otherwise, driven low during the TC and high otherwise, or toggled every time a TC occurs. The TC-toggled mode is used to generate variable duty-cycle square waves.

Counter Parameters

The count source (`cntSource`) parameter selects the source used as input to the specified counter. The count source can be any one of the following:

- the counter inputs--`Src1` to `Src5` (P3 pins 36, 19, 17, 15 or 13)
- the counter gates--`Gate1` to `Gate5` (P3 pins 37, 18, 17, 16 or 14)
- an internal frequency--`F1` to `F5`
- the TC-toggled output of the previous counter (N-1)

The internal frequencies are divide-by-10 divisions of the onboard oscillator which is by default 1 MHz, but can be jumpered to 10 MHz. The sources `F1` through `F5` correspond to the frequencies 1 MHz, 100 kHz, 10 kHz, 1 kHz and 100 Hz. The TC-toggled output of the previous counter can be used as a source—allowing counters to be cascaded without external connections.

The `cntEdge`, `cntDir`, `cntType`, `cntRepeat`, `reload`, and `specGate` parameters all take boolean value types. For the follwoing discussions, if any of these parameters has a non-zero value, it is understood to be `true`; if a parameter has a value of zero, it is understood to be `false`.

The count edge (`cntEdge`) parameter selects whether the counter will count when it receives a rising or falling edge on its count source (see the count source parameter description above).When set to `true`, a rising count edge will be used; if the value of `cntEdge` is `false`, and falling count edge will be used.

The count direction (`cntDir`) parameter selects whether the counter will count up or down. If `cntDir` is set to `true`, the count will go up; if the value of `cntDir` is `false`, the count will go down. The counter is normally configured for down counting when generating a pulse or square wave. The load register would be set to a positive value that would descent in decrements to zero, defining the duration or width of the waveform. In event counting, the counter would initially be set to zero and configured to count up (in this case, the hold register would contain the number of events received).

The count type (cntType) parameter selects either binary or binary-coded decimal (BCD) counting. A value of true for this parameter selects a BCD count, while a value of false will select a binary count. Binary format accepts a 16-bit number ranging from 0 to 65,535. BCD format accepts four 8-bit numbers representing 0 to 9, ranging from 0 to 9,999. In this format, each of the 8-bit numbers represents a placeholder in a base-10 system—for instance, if the thousands bit is 2, the hundreds bit is 5, the tens bit is 7, and the ones bit is 9, then the value of the four bits together is 2,579.

The count repeat (cntRepeat) parameter causes the counter to re-arm after TC occurs if true, and does not re-arm the counter after TC if false. Applications such as software re-triggerable 1-shots would disable the repeat flag so the 1-shot occurs only after the counter arm command is sent. Other applications (such as rate generators, square waves and hardware re-triggerable 1-shots) would enable the count repeat so that the counter will run until disarmed.

The reload (reload), special gate(specGate)and gate control (gateCtrl)parameters can be used together to configure the counter. Using these three commands, the counter can be configured in one of four ways:

- If the reload parameter is set to false, then the counter will only use the contents of the load register for counting.
- If the reload parameter is true and the special gate parameter is false, then the counter will alternate between registers.
- If both the reload and the special gate parameters are true, and the gate control parameter is inactive, then the counter will use the hold register for counting if the counter's gate is high, or to use the load register if the gate is low.
- If both the reload and the special gate parameters are true, and the gate control parameter is active, then the operation is dependent on the gateCtrl parameter value.

The chart below summarizes the various configurations of counter mode operation.

Counter Mode Operating Summary

Counter Mode	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Special Gate (CM7)	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
Reload Source (CM6)	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1
Repetition (CM5)	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
Gate Control (CM15-CM-13); N=no gating; L=level; E=edge	N	L	E	N	L	E	N	L	E	N	L	E	N	L	E	N	L	E	N	L	E	N	L	E
Count to TC once, then disarm	X	X	X										X	X										
Count to TC twice, then disarm							X	X	X										X					
Count to TC repeatedly without disarming				X	X	X				X	X	X				X	X					X		X
Gate input does not gate counter input	X			X			X			X									X			X		
Count only during active gate level		X			X			X			X			X			X							
Start count on active gate edge and stop count on next TC			X			X									X			X						X
Start count on active gate edge and stop count on second TC									X			X												
No hardware re-triggering	X	X	X	X	X	X	X	X	X	X	X	X							X			X		X
Reload counter from Load Register on TC	X	X	X	X	X	X							X	X			X	X						X
Reload counter on each TC, alternating reload source between Load and Hold Registers							X	X	X	X	X	X												
Transfer Load Register into counter on each TC that gate is LOW, transfer Hold Register into counter on each TC that gate is HIGH																			X		X			
On active gate edge transfer counter into Hold Register and then reload counter from Load Register													X	X			X	X						
On active gate edge transfer counter into Hold Register, but counting continues																								X

Prototypes

C/C++

```
daq9513SetCtrMode(DaqHandleT handle, DaqIODeviceType deviceType, DWORD
whichDevice, DWORD ctrNum, Daq9513GatingControl gateCtrl, BOOL cntEdge,
Daq9513CountSource cntSource, BOOL specGate, BOOL reload, BOOL cntRepeat,
BOOL cntType, BOOL cntDir, Daq9513OutputControl outputCtrl);
```

Visual BASIC

```
VBdaq9513SetCtrMode&(ByVal handle&, ByVal deviceType&, ByVal whichDevice&,
ByVal ctrNum&, ByVal gateCtrl&, ByVal cntEdge&, ByVal cntSource&, ByVal
specGate&, ByVal reload&, ByVal cntRepeat&, ByVal cntType&, ByVal cntDir&,
ByVal outputCtrl&)
```

Program References

```
DAQEX.FRM (VB)
```

daq9513SetHold

Format

daq9513SetHold (handle, deviceType, whichDevice, ctrNum, ctrVal)

Purpose

daq9513SetHold outputs a value to the hold register of the specified counter.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to set the 9513 hold register
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
ctrNum	DWORD	The counter number
ctrVal	WORD	Variable which stores the value read from the hold register of the selected counter

Parameter Values

handle: obtained from the daqOpen function

deviceType: must be set to value DiodtLocal9513

whichDevice: valid value for all current devices is 0

ctrNum: valid values range from 1 to 5

crtVal: pointer to a variable from which the hold register will be set. Valid values for the hold register range from 0 to 65,535

Returns

DerrInvCtrNum	Invalid channel
DerrInvGateCtrl	Invalid gate
DerrInvCntSource	Invalid source
DerrInvOutputCtrl	Invalid output
DerrNotCapable	No 9513 available
DerrNoError	No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

The hold register can be used to set the counter's initial value using the daq9513MultCtrl function. Please see the daq9513SetMasterMode and daq9513SetCtrMode function entries for a description of various uses for the hold register.

Prototypes

C/C++

daq9513SetHold(DaqHandleT handle, DaqIODeviceType deviceType, DWORD whichDevice, DWORD ctrNum, WORD ctrVal);

Visual BASIC

VBdaq9513SetHold&(ByVal handle&, ByVal deviceType&, ByVal whichDevice&, ByVal ctrNum&, ByVal ctrVal%)

Program References

9513EX01.CPP, 9513EX01.FRM (VB)

daq9513SetLoad

Format

```
daq9513SetLoad (handle, deviceType, whichDevice, ctrNum, ctrVal)
```

Purpose

daq9513SetLoad outputs a value to the load register of the specified counter.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device to set the 9513 load register
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
ctrNum	DWORD	The counter number
ctrVal	WORD	Variable which stores the value read from the load register of the selected counter

Parameter Values

handle: obtained from the daqOpen function

deviceType: must be set to value DiodtLocal9513

whichDevice: valid value for all devices is 0

ctrNum: valid values range from 1 to 5

ctrVal: pointer to a variable from which the load register will be set. Valid values for the load register range from 0 to 65,535

Returns

DerrInvCtrNum	Invalid channel
DerrInvTod	Invalid time of day mode
DerrInvDiv	Invalid divisor
DerrNotCapable	No 9513 available
DerrNoError	No error

For more details on error messages, please refer to the Daq Error Table.

Function Usage

The load register can be used to set the counter's initial value using the daq9513MultCtrl. Please see the daq9513SetMasterMode and daq9513SetCtrMode function entries for a description of various uses for the load register.

Prototypes

C/C++

```
daq9513SetLoad(DaqHandleT handle, DaqIODeviceType deviceType, DWORD
whichDevice, DWORD ctrNum, WORD ctrVal);
```

Visual BASIC

```
Vbdaq9513SetLoad&(ByVal handle&, ByVal deviceType&, ByVal whichDevice&,
ByVal ctrNum&, ByVal ctrVal%)
```

Program References

9513EX01.CPP, 9513EX01.FRM (VB)

---

daq9513SetMasterMode

Also See: **daq9513SetLoad, daq9513MultCtrl, daq9513GetHold, daq9513SetCtrMode**

Format

```
daq9513SetMasterMode (handle, deviceType, whichDevice, foutDiv,
cntSource, comp1, comp2, tod)
```

Purpose

daq9513SetMasterMode is used to set the counter's master mode register, which is used to configure the frequency output pin (P3 pin 30), the comparators of counter 1 and 2, and the time-of-day operation of the 9513 chip.

Parameter Summary

Parameter	Type	Description
handle	DaqHandleT	Handle to the device in which to set the 9513 master mode
deviceType	DaqIODeviceType	Specifies the 9513 device type
whichDevice	DWORD	Specifies which 9513
foutDiv	DWORD	The frequency output (fout) divider
cntSource	Daq9513CountSource	The frequency output (fout) source
comp1	BOOL	A flag that, if true, will enable the compare 1 operation; if false, it will be disabled
comp2	BOOL	A flag that, if true, will enable the compare 2 operation; if false, it will be disabled
tod	Daq9513TimeOfDay	The time-of-day mode

Parameter Values

handle: obtained from the daqOpen function
deviceType: must be set to value DiodtLocal9513
whichDevice: valid value for all devices is 0
foutDiv: valid values range from 1 to 16; 0 selects divider of 16
cntSource: see table below
comp1: valid values are either true ($\neq 0$) and false ($= 0$)
comp2: valid values are either true ($\neq 0$) and false ($= 0$)
tod: see table below

Parameter Type Definitions

cntSource-(Daq9513CountSource)	
Definition	**Description**
DcsTcnM1*	TC toggled output of previous (N-1) counter
DcsSrc1	Counter 1 input (pin36 of P3)
DcsSrc2	Counter 2 input (pin19 of P3)
DcsSrc3	Counter 3 input (pin17 of P3)
DcsSrc4	Counter 4 input (pin15 of P3)
DcsSrc5	Counter 5 input (pin13 of P3)
DcsGate1	Counter 1 gate (pin37 of P3)
DcsGate2	Counter 2 gate (pin18 of P3)
DcsGate3	Counter 3 gate (pin16 of P3)
DcsGate4	Counter 4 gate (pin14 of P3)
DcsGate5	Counter 5 gate (pin12 of P3)
DcsF1	Onboard 1 MHz Clock
DcsF2	Onboard 100 kHz Clock
DcsF3	Onboard 10 kHz Clock
DcsF4	Onboard 1 kHz Clock
DcsF5	Onboard 100 Hz Clock
*invalid with daq9513SetMasterMode	

tod-(Daq9513TimeOfDay)	
Definition	**Description**
DtodDisabled	Time of day function is not used
DtodDivideBy5	A 50Hz signal is being applied to pin 36 of P3 to generate time of day input
DtodDivideBy6	A 60Hz signal is being applied to pin 36 of P3 to generate time of day input
DtodDivideBy10	A 100Hz signal is being applied to pin 36 of P3 to generate time of day input

Returns

DerrInvCntSource	Invalid source
DerrInvTod	Invalid time of day mode
DerrInvDiv	Invalid divisor
DerrNotCapable	No 9513 available
DerrNoError	No error

For more details on error messages refer to the Daq Error Table. The table precedes Appendix A.

Function Usage

All daq9513SetMasterMode parameters default to zero after daqOpen.

Configuring the Frequency Output Pin and Comparators

The frequency output source (cntSource) parameter selects what signal will be output on the frequency output (fout) pin. The fout source can be any one of the following:

- the counter inputs--Src1 to Src5 (P3 pins 36, 19, 17, 15 or 13)
- the counter gates--Gate1 to Gate5 (P3 pins 37, 18, 17, 16 or 14)
- an internal frequency--F1 to F5

The sources F1 through F5 correspond to the frequencies 1 MHz, 100 kHz, 10 kHz, 1 kHz and 100 Hz. The fout divider will divide the selected source by 1 to 16 before outputting the signal on the fout pin.

The 2 comparator flags (comp1 and comp2) control the comparators associated with counters 1 and 2. If a comparator is set to true, the value in the corresponding alarm register (set with the daq9513SetAlarm function) will be compared with the value in the counter. The output of the corresponding counter will become true when the value in the counter reaches the value in the alarm register; the output remains true until the counter value changes. The polarity of the output depends on the output control (set with the

`daq9513SetCtrMode` function). When either the output control is high, the terminal count pulsed, or the terminal count toggled, then the output will be high while the comparator is `true`. When the output control is low and terminal count pulsed, the output will be low while the comparator is `true`.

Using the Time-Of-Day Parameter

Time-of-Day Configuration
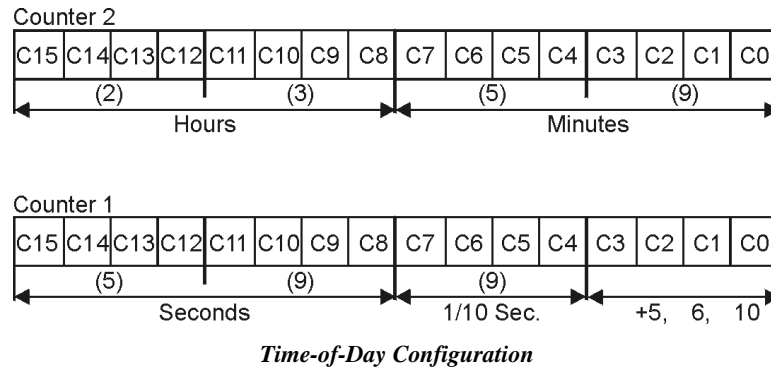
The time-of-day (`tod`) parameter is used to enable or disable the time-of-day operation. The time-of-day operation is a special mode which causes counters 1 and 2 to turn over at counts that generate 24-hour time-of-day accumulations. A 10 Hz clock source is needed to drive the time-of-day clock. Therefore, if a 100 Hz, 60 Hz or 50 Hz signal is applied to the input of counter 1 (P3 pin 36), the appropriate divide-by mode (`DtodDivideBy10`, `DtodDivideBy6` and `DtodDivideBy5`, respectively) must be activated. So, if a 60 Hz signal is being used, the `tod` parameter must be set to divide by 6 (`DtodDivideBy6`). The resolution of the time-of-day operation is 0.1 seconds. The hold registers of counters 1 and 2 will hold the 24-hour time.

The following steps must be performed to use the time-of-day operation:

1. Set the master mode register as described above.

2. For general-purpose time keeping, configure counter 1 using `daq9513SetCtrMode` with the following parameters: no gating, count on rising edge, special gating disabled, reload from hold only, count repetitively, BCD counting and count up. The count source can be any of the available sources. The output control does not affect time-of-day operation.

3. Set the mode of counter 2 with the same settings as counter 1, except that the count source should be TC toggled of the previous (N-1) counter. This allows internal concatenation of counter 1 to counter 2.

4. Set the load registers of counter 1 and 2 to zero, using the `daq9513SetLoad` function.

5. Initialize the current 24-hour time-of-day by setting the load registers of counters 1 and 2, using the format shown in the figure above (again using `daq9513SetLoad`).

6. Repeat step 4.

Prototypes

C/C++

```
daq9513SetMasterMode(DaqHandleT handle, DaqIODeviceType deviceType, DWORD
whichDevice, DWORD foutDiv, Daq9513CountSource cntSource, BOOL comp1, BOOL
comp2, Daq9513TimeOfDay tod);
```

Visual BASIC

```
VBdaq9513SetMasterMode&(ByVal handle&, ByVal deviceType&, ByVal
whichDevice&, ByVal foutDiv&, ByVal cntSource&, ByVal comp1&, ByVal
comp2&, ByVal tod&)
```

Program References

```
9513EX01.CPP, 9513EX01.FRM
```

This page is intentionally blank.

Glossary

This list of terms is intended to give a brief background on some of the terms used throughout the Daq Programmers Manual. These definitions should prove a highly informative primer for those unfamiliar with the data acquisition terminology.

8255
Refers to Intel 8255 chip. This chip (or emulation of the chip) is used in Daq products for P2 Digital I/O support.

9513
Refers to the Texas Instruments 9513 chip. This chip is used by DaqBook/100 Series, DaqBook/200 Series, and ISA-type DaqBoard products for Counter/Timer operations.

A/D
A/D refers to an "analog-to-digital" converter. A/D's convert a voltage over a specific range to a digitized reading. The value of the digitized reading depends upon the resolution of the A/D device. Most A/D devices have resolution of 12 or 16 bits. The range over which the conversion is performed depends upon the gain and polarity selected.

A/D Resolution
With 12-bit A/D's, the significant values of the converted reading can range from 0 to 4,095 over the specified voltage range. With 16-bit A/D's, the significant values of the converted reading can range from 0 to 65,535 over the specified voltage range.

A/D Data Representation
At times, the presentation of A/D values may differ, depending on the device or the current mode of the device. For instance, a 12-bit reading may be normalized to a 16-bit ranging from 0 to 65,535, where the 4 least significant bits are not relevant to the actual value of the reading. Also, a reading may be signed if the range specified is a bipolar voltage, where the digitized reading will range from –32,768 to +32,767 rather than from 0 to 65,535.

Acquisition
An acquisition is the collection of analog, digital or counter input based upon a common input synchronization event. The common synchronization event can be an internal time-base generated by an on-board clock, or it can be an external signal. The start event (also referred to as the "trigger event") for an acquisition may take one of several forms—triggers can be based on input channel values, external events, or programmatically defined events. Likewise, the end of an acquisition (also referred to as the "stop event") may be based on similar criteria.

Acquisition Frequency
The rate at which an acquisition takes place, measured in terms of frequency (Hz).

Acquisition Period
The rate at which an acquisition takes place, measured in terms of period (nanoseconds).

Acquisition Rate
The rate (in frequency or period) at which channel scans are to be taken, if using an internal clock source for the acquisition.

ADC acquisition
See **Acquisition**.

Analog
A signal of varying voltage or current that communicates data (compare with **Digital**).

API (Application Program Interface)
The interface program within the Daq system's driver that includes function calls specific to Daq hardware and can be used with user-written programs (several languages supported)

Arm
An action that enables a device to detect the trigger/start event specified.

Array

A collection of quantities [of the same data type] that are located in contiguous memory.

Asynchronous

Describes an event or action that is not temporally related to other events or actions. Normally, this describes an event or action that uses no synchronization method (such as an external or internal clock) to coordinate the event or action with other events or actions.

BCD (Binary Coded Decimal)

Refers to a data format where each byte represents a digit from 0 to 9. This is used mostly in reference to the counter mode selection for the 9513 chip. If used, 4 bytes are available, each byte representing a digit between 0 and 9 multiplied by subsequent powers of 10, from 0 to 3. Thus, the range for this number would be from 0 to 9,999 (dec).

Bit Mask

A collection of bits (usually 8 to 32-bits long) that is used to configure devices, channels or operations. A bit mask can also represents the state of certain operations, and events detected by the device. In a bit mask, each bit has special meaning that can be interpreted using the bit mask enumerations located in each specific programming language's API header files.

BOOL

A 32-bit Boolean (4 bytes) quantity that can take on either true ($\neq 0$) or false ($= 0$) values. This parameter type should be *passed by value* according to the dictates of the programming language used.

Boolean

A value that indicates a binary state of either true (see **true**) or false (see **false**).

Buffer (circular)

A buffer that will continue the reading or writing operation upon reaching the end of the buffer by starting again at the beginning of the buffer. This style of buffer is normally useful when there is not enough memory available to hold the entire acquisition input or waveform/pattern output data.

Buffer (linear)

A buffer that will terminate the reading or writing operation upon reaching the end of the buffer. Once the end of the buffer has been reached, no more data may be read from or written to the buffer. This style of buffer is normally useful when there is enough memory available to hold the entire acquisition input or waveform/pattern output data.

Buffer Position

The current position of the read or write operation of the buffer. If the buffer is being used to hold acquisition input data, then this position refers to the write (or head) position within the buffer at which the driver will store the next available data block. If the buffer is being used to buffer waveform/pattern output data, then this position refers to the read (or tail) position within the buffer from which the driver will retrieve the next available data block.

Channel

In reference to Daq products, a channel refers to a single *input*, or *output* entity. In a broader sense, an *input channel* is a signal path between the transducer at the point of measurement and the data acquisition system. A channel can go through various stages (buffers, multiplexers, or signal conditioning amplifiers and filters). Input channels are periodically sampled for readings. An *output channel* from a device can be digital or analog. Outputs can vary (as dictated by a program) in response to an input channel signal.

Channel Scan Configuration

See **Channel Scan Group**.

Channel Scan Group

A list of input channel configurations that fully define each individual channel's configuration for a particular data acquisition operation.

CJC (Cold Junction Compensation) Channel

A Cold Junction Compensation channel is used to correct Cold Junction offset during temperature correction when using thermocouple channels.

CTR (Counter) channel

A 16-bit or cascaded 32-bit counter input channel on a device.

Command

A DaqX API command. Commands are accessible through the DaqX API support files.

Comparator

A device that can compare an input value to a specified programmed value. Often used in reference to 9513 counter operations.

D/A

A digital-to-analog converter. D/A's convert digital values (binary bits) into analog signals, manifested as a voltage output. The voltage level generated depends upon the voltage range of D/A converter, the resolution of the D/A converter, and the digital value passed to the D/A converter. Most D/A devices have resolution of 12 or 16 bits.

D/A Data Representation

At times, the presentation of digital values may differ according to the device, or the current mode of the device. For instance, a 12-bit digital quantity may be normalized to a 16-bit digital quantity ranging from 0 to 65,535, where only the 12 most significant bits are passed to the D/A. Also, a digital value may be signed if the device has been placed into a signed data format mode. If this is the case, then the voltage generated may correspond to a signed digital integer value ranging from –32,768 to +32,767 rather than 0 to 65,535.

D/A Resolution

With 12-bit D/A's, the digital value passed to the D/A can range from 0 to 4,095 where each bit in the digital value is equal to the D/A's voltage range divided by 4,095. With 16-bit D/A's, the digital values passed to the D/A can range from 0 to 65,535 where each bit in the digital value is equal to D/A's voltage range divided by 65,536.

DAC (D/A Converter) Channel

A channel that corresponds to a D/A on the device or expansion module.

Daq*

Any hardware device supported by the DaqX API. These devices include: DaqBooks, DaqBoards (ISA-type), Daq PC Cards, TempBooks, WaveBooks, DaqBoard/2000 Series boards, and cPCI DaqBoard/2000c Series boards.

Data type

The format of the parameters being passed into the API function/command.

Device

Specifies main unit devices, such as DaqBooks, DaqBoards, WaveBooks, TempBooks and Daq PC Cards.

Differential Mode (DE)

Differential mode measures a voltage between 2 signal lines for a single channel (compare with **Single-Ended Mode**).

Digital

A digital signal is one of discrete value, in contrast to a varying signal. Combinations of binary digits (0s and 1s) represent digital data.

DIO channel

A digital input or output channel.

Disarm

An action that disables the ability of the device to detect the trigger/start event specified. If the trigger/event has already occurred, the disarm action will stop the current operation.

DOUBLE

A 64-bit double precision (8 bytes) floating point number. This parameter type should be *passed by value* according to the dictates of the programming language used.

DWORD

A 32-bit unsigned integer (4 bytes) quantity that can range from 0 to 4,294,967,295. This parameter type should be *passed by value* according to the dictates of the programming language used.

External clock

The external clock is a device that uses pulses to drive synchronized input or output operations. The pulses originate in a foreign device, such as a machine, to which the external clock is connected.

Falling/Negative edge

Describes an event in which an input signal has exceeded or reached a particular level with a negative slope. This normally refers to a trigger or start event, or the manner in which a counter detects an event that should be counted.

False

The "false" value of a Boolean data type (= 0). Also See **True** and **Boolean**.

Flag

See **Bit Mask**.

FIFO (First In, First Out)

A mechanism for buffering input or output data on the device. This mechanism alleviates possible loss of data when transmitting clocked or synchronous data due to inherent system latencies in either the PC or controlling application.

FLOAT

A 32-bit single precision (4 bytes) floating point number. This parameter type should be *passed by value* according to the dictates of the programming language used.

Frequency Output (fout)

The output frequency programmed for a particular timer output channel. The frequency represents the rate at which the timer generates an output pulse.

Function

Normally, this refers to a DaqX API command. The command or function is accessible through the DaqX API support files.

Gain

The degree to which an input signal is amplified (or attenuated) to allow greater accuracy and resolution; can be expressed as × n (where n is some integer), or ±dB. In terms of programming, most device channels can have their gain value programmed. Gain codes are provided in the API for each device gain applicable.

Handle

An integer that represents the device when the device is being accessed, after the device session has been opened. The handle to the device may be obtained via the daqOpen command.

Hardware Trigger

A trigger event that is detected on the device. Usually, these triggers take the form of an analog level or TTL level signal. These types of triggers normally result in lower trigger detection latencies, but are not as flexible as software trigger events.

Hold Register

Represents the hold register of the 9513.

Input Sample

The data for a *single* input channel that is part of a scanned channel acquisition.

Internal clock

The internal clock resides in the acquisition device and can be set (programmed) through software. The pulse from the internal clock is used to drive synchronized input or output operations.

Linearization

Some transducers produce a voltage in linear proportion to the condition measured. Other transducers (e.g., thermocouples) have a nonlinear response. Converting nonlinear signals into accurate readings requires software to calibrate several points in the range used, and then interpolate values between these points.

Load Register

Represents the load register of the 9513.

LONG

A 32-bit signed integer (4 bytes) quantity that can range from -2,147,483,648 to +2,147,483,647. This parameter type should be *passed by value* according to the dictates of the programming language used.

LPSTR

A pointer to a character string. This parameter type should be *passed by reference* according to the dictates of the programming language used. This parameter is normally a pointer to a device name or other ASCII string value.

Multiplexer (MUX)

A device that collects signals from several inputs and outputs them on a single channel.

Parameter

An element of the function, or of the command prototype, that is passed into the function.

PBOOL

A pointer to a 32-bit Boolean (4 bytes) quantity or an array of 32-bit Boolean quantities that take on false ($= 0$) or true($\neq 0$) values. This parameter type should be *passed by reference* according to the dictates of the programming language used.

PDOUBLE

A pointer to a 64-bit double precision (8 bytes) floating point number or an array of 64-bit double precision floating point numbers. This parameter type should be *passed by reference* according to the dictates of the programming language used.

PDWORD

A pointer to a 32-bit unsigned integer (4 bytes) or an array of 32-bit unsigned integer quantities that can range from 0 to 4,294,967,295. This parameter type should be *passed by reference* according to the dictates of the programming language used.

PFLOAT

A pointer to a 32-bit single precision (4 bytes) floating point number or an array of 32-bit single-precision floating point numbers. This parameter type should be *passed by reference* according to the dictates of the programming language used.

PLONG

A pointer to a 32-bit signed integer (4 bytes) or an array of 32-bit signed integer quantities that range from: -2,147,483,648 to 2,147,483,647. This parameter type should be *passed by reference* according to the dictates of the programming language used.

Pointer

The address of the value [or variable quantity] in memory, rather than the actual value or variable itself.

Post-trigger

The data [or state of an acquisition] after the occurrence of the trigger event. If referencing data, the post-trigger data was the data collected after or during the occurrence of the trigger event. If referencing the acquisition state, then the device has been triggered.

Pre-trigger

The data or state of an acquisition before the occurrence of the trigger event. If referencing data, the pre-trigger data was the data collected before the trigger event occurred. If referencing the acquisition state, then the device has currently not been triggered.

PSHORT

A pointer to a 16-bit signed integer (2 bytes) or array of 16-bit signed integer quantities, ranging from –32,768 to +32,768. This parameter type should be *passed by reference* according to the dictates of the programming language used.

PWORD

A pointer to a 16-bit unsigned integer (2 bytes) or array of 16-bit unsigned integer quantities, ranging from 0 to 65,535. This parameter type should be *passed by reference* according to the dictates of the programming language used.

Rising/Positive Edge

An event in which an input signal has reached [or exceeded] a particular level with a positive slope. This normally refers to a trigger or start event, or the manner in which a counter detects an event that should be counted.

RTD (Resistance Temperature Detector)

An RTD is a 3 or 4 wire transducer that uses resistance to produce 3 voltage inputs that can then be converted to a temperature, using known transfer functions in software. Both 3 and 4 wire configurations may be used.   For programming, this term generally refers to a channel (using a DBK9) that has an RTD connected to it.

RTD (Resistance Temperature Detector) channel

Describes an RTD channel on a DBK9 temperature expansion module.

Sample

The value of a signal on an input or output channel at an instant in time.

Scan

Either the channels that are configured for acquisition (see **Channel Scan Group**), or the data retrieved during the acquisition for a single channel scan group.

Sequencer

A programmable device that manages input channels and channel-specific settings for devices that multiplex their input channels (MUX).

SHORT

A 16-bit signed integer (2 bytes) quantity that can range from –32,768 to +32,768. This parameter type should be *passed by value* according to the dictates of the programming language used.

Simultaneous Sample and Hold (SSH)

An operation that gathers samples from multiple channels at the same instant and holds these values until all are sequentially converted to digital values.

Single-Ended Mode (SE)

The single-ended mode measures a voltage between a signal line and a common reference that may be shared with other channels (compare with **Differential Mode**).

Software Trigger

A trigger event that is detected in the DaqX driver-software. These types of triggers normally result in higher trigger detection latencies but are much more flexible than hardware trigger events.

Stop Event

An event that terminates the acquisition of post-trigger data. This event, when satisfied, will cause the device to stop collecting post-trigger data.

T/C (Thermocouple)

A thermocouple is a transducer that produces a voltage relative to temperature at the junction of two dissimilar metals. Various types of thermocouples are available, each having characteristics particular to certain temperature ranges. For programming, this term generally refers to a channel that has a thermocouple connected to it.

T/C (Thermocouple) Channel

A thermocouple channel on a temperature measurement module or temperature expansion card.

Transistor-Transistor Logic (TTL)

Transistor-Transistor Logic (TTL) is a circuit in which a multiple-emitter transistor has replaced the multiple diode cluster (of the diode-transistor logic circuit); typically used to communicate logic signals at 5 V.

Trigger/Start Event

An event that initiates the acquisition of post-trigger data. This event, when satisfied, will cause the device to begin to collect post-trigger data.

Unipolar

A range of analog signals that is always zero or positive (e.g., 0 to 10 V). Evaluating a signal in the right range (unipolar or bipolar) allows greater resolution by using the full-range of the corresponding digital value. Also see **bipolar**.

Update Block

The data for *all* output channels that are part of a waveform/pattern output operation.

Update Clock

A pulse from an internal or external source that causes all waveform/pattern output channels to be updated synchronously.

Update Rate/Frequency

The rate at which update blocks should be presented to the outputs when using an internal clock source for the waveform/pattern output operation.

Update Sample

The data for just *one* output channel that is part of a waveform/pattern output operation.

Waveform/pattern output

A clocked, synchronous output operation [from PC memory or disk file] to one or more valid DAC or Digital Output channels. D/A and digital output data are output synchronously from the device's built-in output FIFO, which is fed from PC memory or disk file. Output updates are presented synchronously to each of the output ports based upon an external or internal clock pulse.

WORD

A 16-bit unsigned integer (2 bytes) quantity that can range from 0 to 65,535. This parameter type should be *passed by value* according to the dictates of the programming language used.

Notes