## Contents

Section I. Introduction .............................................................................................................. 1

1. Introduction ................................................................................................................ ... 3
   About This Document ................................................................................................. 4
   iDSC Fundamentals ............................................................................................... 5
   Filter Characteristics .............................................................................................. 7

2. DSC Graphical Design ................................................................................................ 11
   Individual Interface ................................................................................................... 12
   Input Screen ........................................................................................................ 12
   Filter Design Screen .............................................................................................. 14
   External Board Screen ........................................................................................... 16
   Group Interface ......................................................................................................... 17
   DSCs ................................................................................................................... 17
   Address ............................................................................................................... 17
   Mode ................................................................................................................... 18
   External Board Calibrate ..................................................................................... 18
   External Board Enable ........................................................................................ 18
   Raw Data ............................................................................................................. 18
   Remote Master ..................................................................................................... 18
   Server Disk Log ................................................................................................... 18
   Copy .................................................................................................................... 19
   Paste .................................................................................................................... 19

Input Parameters ........................................................................................................ 20
   Sample Rate ........................................................................................................ 20
   Group Delay ....................................................................................................... 20
   Input Range ......................................................................................................... 20
   Filter Name ......................................................................................................... 21
   Enabled ............................................................................................................... 21

Filter Design Parameters ........................................................................................... 22
   Filter Name ......................................................................................................... 22
   Filter Type ........................................................................................................... 22
   Sharpness ............................................................................................................ 22
   Low Cutoff Frequency ....................................................................................... 22
   Low Cutoff Slope ............................................................................................... 23
   High Cutoff Frequency ...................................................................................... 23
   High Cutoff Slope ............................................................................................... 23
   Attenuation ......................................................................................................... 23

Filter Response .......................................................................................................... 24
   Linear Display ..................................................................................................... 26
   Linear Zoom Display .......................................................................................... 27
   Log Display ......................................................................................................... 28
   Log Zoom Display .............................................................................................. 29
   Unit Step Display .................................................................................................. 30

External Board Parameters .......................................................................................... 31
Contents

Input Type ........................................................................................................... 31
Input Range ......................................................................................................... 31
Input Offset ........................................................................................................ 31
Input Offset Range ............................................................................................. 31
Output Excitation ............................................................................................... 32

Section II. Application Software ........................................................................... 33

3. DSCview ............................................................................................................. 35
   Save and Load Workspaces .............................................................................. 36
   Start & Stop ...................................................................................................... 36
   System Options ............................................................................................... 37
   Configuration Window ..................................................................................... 39
   Graph Window ............................................................................................... 42
   Table Window .................................................................................................. 46
   Disk Log Window ............................................................................................ 48
   Server Disk Log Window .................................................................................. 50

4. Using the iDSC Board with DASYLab ............................................................. 51
   Installing iDSC Board Support for DASYLab .................................................. 51
   Using an iDSC Board Module with DASYLab ................................................ 52
   Running the DASYLab iDSC Board Examples .............................................. 52
   Using More Than One iDSC Board with DASYLab ....................................... 52
   Synchronizing Several iDSC Boards .............................................................. 53
   Special Addressing ......................................................................................... 53

5. Using the iDSC Board with LabVIEW ............................................................. 55
   Installation ....................................................................................................... 55
   Creating an iDSC Board Application in LabVIEW ......................................... 56
   Running the LabVIEW iDSC Board Examples .............................................. 56
   App01 - BASIC ............................................................................................. 57
   App02 - GRAPH ........................................................................................... 57
   App03 - LOG ................................................................................................. 57
   App04 - LOGVW .......................................................................................... 57
   App05 - DaplFFT .......................................................................................... 57
   App06 - DaplCC ......................................................................................... 57
   App07 - Disk Logging (1 iDSC) ................................................................. 58
   App08 - Disk Logging (2 iDSC with synchronization) .................................... 58
   App09 – A Group of iDSC ............................................................................ 59

DLL Reference ................................................................................................... 60
   DSCIO DLL Function Reference .................................................................... 60
   MSLAPP DLL Function Reference .................................................................. 62
   Data Format ..................................................................................................... 62
   Ms1DscServerDiskLogConfigSet ...................................................................... 63

SubVIs Reference ............................................................................................... 65
   iDSC Init ...................................................................................................... 65
   iDSC Read ..................................................................................................... 66
   iDSC Close .................................................................................................... 68
   iDSC .............................................................................................................. 68
   iDSC MsSI ................................................................................................. 69
   DiskLog ......................................................................................................... 70
Section III. Programming Interfaces ................................................................................... 101

9. DSCIO DLL Programmer’s Interface ........................................................................ 103
   DSCIO Interface Examples ..................................................................................... 104
       Visual Basic Examples .................................................................................. 104
       Dvm.vbp .................................................................................................... 104
       BinLog.vbp ............................................................................................... 104
       LoadSave.vbp ............................................................................................ 104
       C/C++ Console Examples ........................................................................ 104
       BinLog.cpp .............................................................................................. 104
       TxtLog.cpp .............................................................................................. 105
       LoadSave.cpp ........................................................................................... 105
   Creating a DSCIO Interface Application ................................................................. 106
   Universal Naming Convention ........................................................................ 107
   Master/Slave Configuration ............................................................................. 108
   DAPL Support .................................................................................................. 110
       Writing DAPL ........................................................................................... 110
       Using the DAPL Interface ........................................................................ 110
   Structure Summary .......................................................................................... 112
       TBufferGetEx ............................................................................................ 113
       TDscioInt64 .............................................................................................. 115
       TFilterParam ............................................................................................ 117
       TProcSystemErrorStdcall ......................................................................... 120
       TServerDiskLogConfig ............................................................................. 121
       TXbPinConfig ............................................................................................ 125
   Function Summary ........................................................................................... 127
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DscAddressGet</td>
<td>130</td>
</tr>
<tr>
<td>DscAddressSet</td>
<td>131</td>
</tr>
<tr>
<td>DscBufferAvail</td>
<td>132</td>
</tr>
<tr>
<td>DscBufferGet</td>
<td>133</td>
</tr>
<tr>
<td>DscBufferGetEnabledGet</td>
<td>135</td>
</tr>
<tr>
<td>DscBufferGetEnabledSet</td>
<td>136</td>
</tr>
<tr>
<td>DscBufferGetEx</td>
<td>137</td>
</tr>
<tr>
<td>DscCalibrate</td>
<td>139</td>
</tr>
<tr>
<td>DscCommandsLoad</td>
<td>140</td>
</tr>
<tr>
<td>DscConfigDialogOptionsGet</td>
<td>141</td>
</tr>
<tr>
<td>DscConfigDialogOptionsSet</td>
<td>142</td>
</tr>
<tr>
<td>DscConfigDialogShow</td>
<td>144</td>
</tr>
<tr>
<td>DscConfigRead</td>
<td>147</td>
</tr>
<tr>
<td>DscConfigWrite</td>
<td>149</td>
</tr>
<tr>
<td>DscConfigWriteSize</td>
<td>151</td>
</tr>
<tr>
<td>DscDaplTextGet</td>
<td>152</td>
</tr>
<tr>
<td>DscDaplTextLengthGet</td>
<td>153</td>
</tr>
<tr>
<td>DscDaplTextSet</td>
<td>154</td>
</tr>
<tr>
<td>DscFilterIndex</td>
<td>155</td>
</tr>
<tr>
<td>DscFilterParametersGet</td>
<td>156</td>
</tr>
<tr>
<td>DscFilterParametersSet</td>
<td>157</td>
</tr>
<tr>
<td>DscGroupDelay</td>
<td>159</td>
</tr>
<tr>
<td>DscHandleClose</td>
<td>160</td>
</tr>
<tr>
<td>DscHandleOpen</td>
<td>161</td>
</tr>
<tr>
<td>DscHardwareStop</td>
<td>162</td>
</tr>
<tr>
<td>DscIdGet</td>
<td>163</td>
</tr>
<tr>
<td>DscIdSet</td>
<td>164</td>
</tr>
<tr>
<td>DscInputRangeGet</td>
<td>165</td>
</tr>
<tr>
<td>DscInputRangeSet</td>
<td>166</td>
</tr>
<tr>
<td>DscLastErrorTextGet</td>
<td>167</td>
</tr>
<tr>
<td>DscLastErrorTextSet</td>
<td>168</td>
</tr>
<tr>
<td>DscMasterGet</td>
<td>169</td>
</tr>
<tr>
<td>DscMasterSet</td>
<td>170</td>
</tr>
<tr>
<td>DscMemoryUsed</td>
<td>171</td>
</tr>
<tr>
<td>DscOnSystemErrorSet</td>
<td>172</td>
</tr>
<tr>
<td>DscOperateModeGet</td>
<td>173</td>
</tr>
<tr>
<td>DscOperateModeSet</td>
<td>174</td>
</tr>
<tr>
<td>DscPinEnabledCount</td>
<td>175</td>
</tr>
<tr>
<td>DscPinEnabledGet</td>
<td>176</td>
</tr>
<tr>
<td>DscPinEnabledSet</td>
<td>177</td>
</tr>
<tr>
<td>DscPinToFilterMapGet</td>
<td>179</td>
</tr>
<tr>
<td>DscPinToFilterMapSet</td>
<td>180</td>
</tr>
<tr>
<td>DscRemoteMasterGet</td>
<td>181</td>
</tr>
<tr>
<td>DscRemoteMasterSet</td>
<td>182</td>
</tr>
<tr>
<td>DscRunning</td>
<td>183</td>
</tr>
<tr>
<td>DscSampleRateGet</td>
<td>184</td>
</tr>
<tr>
<td>DscSampleRateSet</td>
<td>185</td>
</tr>
<tr>
<td>Method/Property</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>StopAcquiring method</td>
<td>364</td>
</tr>
<tr>
<td>StructPrepare method</td>
<td>365</td>
</tr>
<tr>
<td>SystemErrorProcess method</td>
<td>366</td>
</tr>
<tr>
<td>TcEnabled property</td>
<td>367</td>
</tr>
<tr>
<td>TcEnabledCount property</td>
<td>369</td>
</tr>
<tr>
<td>TcMaximum property</td>
<td>370</td>
</tr>
<tr>
<td>TcWidth property</td>
<td>371</td>
</tr>
<tr>
<td>TransferFunctionGet method</td>
<td>372</td>
</tr>
<tr>
<td>UnitStepGet method</td>
<td>373</td>
</tr>
<tr>
<td>UnitStepLengthGet method</td>
<td>374</td>
</tr>
<tr>
<td>XbCalibrate method</td>
<td>375</td>
</tr>
<tr>
<td>XbEnabled property</td>
<td>376</td>
</tr>
<tr>
<td>XbPinConfigGet method</td>
<td>377</td>
</tr>
<tr>
<td>XbPinConfigSet method</td>
<td>378</td>
</tr>
<tr>
<td>Obsolete Interface</td>
<td>380</td>
</tr>
</tbody>
</table>

Section IV. Installation and Setup ........................................................................................ 387

11. iDSC Board Hardware Architecture ............................................................................... 389
   Hardware Overview                                                                    389
   iDSC Board Block Diagram                                                             391
   Warm-up and Self-Calibration                                                        392
   Isolation                                                                           392
   Simultaneous Sampling and Synchronization                                            393

12. Installation .................................................................................................................. 395
   iDSC Board Handling Precautions                                                     395
   Installing the iDSC 1816 ......... System Hardware Requirements  396
   Installation Steps                                                                  396
   Installing the iDSC 816 ......... System Hardware Requirements  397
   Standard Configurations                                                             397
   Installation Steps                                                                  397
   Several iDSC Boards 398
   Installing the iDSC Board Software ................................................................ 398
   Testing the Installation ......... Troubleshooting the iDSC 1816 398

13. Advanced Installation Options ..................................................................................... 399
   Nonstandard Configurations                                                         399
   Multiple Board Installation                                                        401

14. Physical Interface ....................................................................................................... 403
   Input/Output Connector                                                             404
   Analog Inputs .............................................................................................................. 406
Section I. Introduction
1. Introduction

The iDSC board from Microstar Laboratories is a complete data acquisition system that occupies one expansion slot in a PC. The board is suitable for a wide range of applications in laboratory and industrial data acquisition and control, especially those involving spectral analysis.

Control of the iDSC board is handled through a turnkey program called DSCview; through standard applications including DASYLab, LabVIEW, LabWindows/CVI, HP VEE, and MATLAB; through a programmer’s graphical user interface; or through a programmer’s function interface.
About This Document

This document is divided into the following four sections:

1. Introduction
2. Application Software
3. Programming Interfaces
4. Installation and Setup

Each section is then subdivided into chapters that relate to specific topics within the section. The Introduction contains introductory material about spectral analysis and the iDSC board, including filter design. The Application Software section describes DSCview, and the DASYLab, LabVIEW, LabWindows/CVI, HP VEE, and MATLAB support for the iDSC board. The Programming Interfaces section describes the graphical user interface and the function interface. Finally, Installation and Setup contains hardware and software installation instructions and detailed information about the hardware architecture of the iDSC 1816 and the iDSC 816.
iDSC Fundamentals

A typical data acquisition system samples input voltages at a fixed rate, the sampling frequency, to produce a stream of numbers. The Nyquist frequency is defined as one-half of the sampling frequency and a signal whose frequency components fall only below the Nyquist can be accurately reconstructed from its components. A signal with frequency components falling both above and below the Nyquist cannot be accurately reconstructed because it is impossible to distinguish between the various frequency components.

Spectral analysis for data acquisition studies observations in the frequency domain rather than in the time domain. This type of analysis places special requirements on a data acquisition system. The system must be accurate enough to distinguish small components at some frequencies from much larger components at other frequencies. The system must also provide alias-free data. If alias-free data cannot be guaranteed, any analysis applied to the data will yield questionable results.

The two figures below illustrate aliasing. The first figure displays one cycle of a 600 Hz sine wave sampled at a rate of 3600 samples/second. Six evenly spaced sampling points are plotted.
The second figure displays the original sine wave at 600 Hz and a second sine wave at 4200 Hz that passes through the same sampling points. Aliasing makes it impossible to distinguish the 600 Hz signal from the 4200 Hz signal.

Aliasing is a sampling artifact, and digital signal processing (DSP) cannot remove aliases after a signal has been sampled. The solution to removing aliases from a data acquisition system is to apply analog filters to the system before sampling. Combining analog filters with digital oversampling, filtering, and decimation produces alias-free, high quality data suitable for spectral analysis.

**Filter Characteristics**

The frequency response of a filter is characterized by its graphs of amplitude response and phase response in terms of frequency. A filter’s amplitude response can be plotted either on a logarithmic scale or a linear scale. The frequency axis of this graph is usually plotted on a linear scale, but can also be plotted on a logarithmic scale. A filter’s phase response is plotted either in degrees or in radians.
The figure below displays an ideal ‘brick wall’ filter. The ideal filter has no attenuation up to its cutoff frequency, and then has infinite attenuation. The ideal filter is not realizable but may be approximated by realizable filters.

The figure below displays typical characteristics of analog filters. Typical rolloff rates range from 24 dB/octave to 72 dB/octave. Even at 72 dB/octave, it takes more than 2 octaves - a factor of four in frequency - to roll off by 96 dB.
In addition to its attenuation, a filter is characterized by its phase response. The figure below displays a typical phase response curve for an analog filter.

![Typical Phase Response of Analog Filters](image)

The ideal phase response increases linearly with frequency. The delay through the filter is independent of frequency and therefore signals pass through the filter without distortion.

Analog filters are complex circuits, requiring precision resistors and capacitors. It is difficult and costly to make an analog filter with a variable cutoff frequency, especially if the cutoff frequency must cover a wide range. Also, analog filters cannot be perfectly linear, and for analog filters there is a trade-off between approximating the ideal amplitude response and approximating the ideal phase response.
The DSC Graphical Design simplifies the iDSC board input configuration and filter design process through graphical user interfaces. The Individual Interface allows configuration of a single iDSC board, and the Group Interface allows configuration of multiple iDSC boards. The Group Interface encapsulates the Individual Interface and therefore can also be used to configure a single iDSC board.

The Individual Interface and Group Interface consists of two main screens, the Input Screen and the Filter Design Screen. There is also an External Board Screen that is only accessible if \texttt{XbEnabled} is set to true. There is a tab for the input screen, filter design screen, and external board screen if \texttt{XbEnabled} is true. Within the filter design screen, there are eight tabs for the filter designs.

The Input Parameters, Filter Design Parameters, and External Board Parameters topics describe all the parameters in detail.
Individual Interface

Input Screen

The Input Screen is activated by selecting the Input tab.

The following figure displays the Input Screen.

The following parameters are configured from the Input Screen.
- Sample Rate
- Input Range
- Filter Name
- Enabled

The following read only properties are displayed in the Input Screen.
- Channel Count
• Group Delay
• Input Pin
• Cutoff Frequency
**Filter Design Screen**

The **Filter Design Screen** is activated by selecting the Filter Design tab. Eight distinct filter designs, activated by a second set of tabs, exist.

The following figure displays the Filter Design Screen.

The following parameters are configured for each filter design from the Filter Design Screen.

- **Filter Name**
- **Filter Type**
- **Sharpness**
- **Low Cutoff Frequency**
- **Low Cutoff Slope**
- **High Cutoff Frequency**
- **High Cutoff Slope**
- **Attenuation**
The Filter Design Screen displays a filter’s amplitude response in one of the following forms:

- **Linear Display**
- **Linear Zoom Display**
- **Log Display**
- **Log Zoom Display**
- **Unit Step Display**

The default amplitude response is **Log Display**. To change the amplitude response, click the right mouse button over the Filter Design Screen and select the **Y Display** menu item.
**External Board Screen**

The External Board Screen is activated by enabling \texttt{XbEnabled} and then selecting the External Board tab.

The following figure displays the External Board Screen.

The following parameters are configured from the External Board Screen.

- **Input Type**
- **Input Range**
- **Input Offset**
- **Output Excitation**

The following read only properties are displayed in the External Board Screen.

- **Input Pin**
- **Input Offset Range**
**Group Interface**

The Group Interface encapsulates the Individual Interface and therefore it supports the Input Screen, Filter Design Screen and External Board Screen.

The following parameters are configured from the Group Interface.

**DSCs**

The DSCs parameter configures the number of iDSC boards in the system. The up arrow adds iDSCs and the down arrow deletes iDSCs.

**Address**

The Address parameter is enabled when the iDSC address is selected from the tree of iDSC boards. The address can be changed in the tree or through the edit box.
Mode
The **Mode** parameter is enabled when the mode is selected from the tree of iDSC boards. The mode can be changed in the tree or through the drop down list. If the mode is changed in the tree, it must be either **Independent** or **Slave of Xxx**, where **Xxx** is the name of the iDSC board. It cannot be set to **Master** since that is automatically set when the slaves are selected.

These additional parameters are configured by clicking the right mouse button.

External Board Calibrate
The **External Board Calibrate** menu item calibrates the external board if the external board is enabled. The external board calibration is not automatic, and has to be invoked explicitly. It is a slow process and should be performed only if necessary.

External Board Enable
The **External Board Enable** menu item enables the external board. Before any communication with the external board is performed, for example external board calibration, the external board must first be enabled.

Raw Data
The **Raw Data** menu item selects raw data rather than filtered data from the iDSC. When **Raw Data** is enabled, all filters configured from the **Filter Design Screen** are ignored.

Remote Master
The **Remote Master** menu item configures a Master iDSC board as a remote master.

Server Disk Log
The **Server Disk Log** menu item configures the server disk log parameters which includes the filename, flags, file share mode, open flags, and file flags attributes. The filename must be valid and enabled must be checked for server disk logging to begin.
The following figure displays the Server Disk Log dialog.

**Copy**
The *Copy* menu item will copy the entire iDSC configuration from the iDSC to the clipboard.

**Paste**
The *Paste* menu item will paste the copied iDSC configuration from the clipboard to another iDSC.
Input Parameters

The following Input Parameters are accessible from the Input Screen.

Sample Rate
The Sample Rate parameter sets the sample rate on each channel, in units of samples per second (s/s). The table below displays the valid sample rates arranged in octaves.

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>153600</th>
</tr>
</thead>
<tbody>
<tr>
<td>102400</td>
<td>76800</td>
</tr>
<tr>
<td>51200</td>
<td>38400</td>
</tr>
<tr>
<td>25600</td>
<td>19200</td>
</tr>
<tr>
<td>12800</td>
<td>10240</td>
</tr>
<tr>
<td>6400</td>
<td>5120</td>
</tr>
<tr>
<td>3200</td>
<td>3072</td>
</tr>
<tr>
<td>1600</td>
<td>1536</td>
</tr>
<tr>
<td>800</td>
<td>768</td>
</tr>
<tr>
<td>400</td>
<td>384</td>
</tr>
<tr>
<td>200</td>
<td>192</td>
</tr>
<tr>
<td>100</td>
<td>96</td>
</tr>
<tr>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
</tr>
</tbody>
</table>

Group Delay
The Group Delay parameter is the amount of time it takes for a reading to pass through the digital filters. All signal frequency components that are present in the passband of the filter are delayed by group delay. The group delay property is read-only and is expressed in units of seconds.

The group delay is calculated using the selected Sample Rate and filter designs. The group delay does not include the filter designs of disabled input pins.

Input Range
The Input Range parameter configures the input range to either +/- 5 V or +/- 10 V.
Filter Name
The Filter Name parameter maps an input pin to a named filter design. The same filter design may be applied to several input pins. An input pin is mapped to a filter design regardless of whether the input pin is disabled.

The filter name can be changed in the Filter Design Screen. When the filter name is changed, the filter name drop down list is updated to reflect the new filter name.

The default filter names are FD0, FD1, FD2, FD3, FD4, FD5, FD6, and FD7.

Enabled
The Enabled parameter check box enables or disables each input pin. At run-time, the iDSC board returns data to the PC only from enabled input pins.
Filter Design Parameters

The following Filter Design Parameters are accessible from the Filter Design Screen.

Each Filter Design Parameter is configured by either entering a number or moving a slider. If you choose to enter a number, you must press ‘Enter,’ or exit the edit box when finished configuring the parameter.

Filter Name
The Filter Name parameter assigns a unique filter name to each filter design. If an already existing filter name is assigned, the existing filter name will not change. A filter name is limited to 63 characters.

The filter name parameter is displayed on the tab at the top of the Filter Design Screen. Changing the filter name parameter automatically updates this tab, and also updates the filter name in the Input Screen.

Filter Type
The Filter Type parameter assigns a filter type to each filter design. The filter type is either lowpass or bandpass.

Sharpness
The Sharpness parameter of a filter determines the sharpness of the corner frequency response. The sharpness of an ideal filter is the number of taps in the final filter stage.

Valid sharpness values are odd numbers in the range 37 to 255, depending on the sample rate.

Low Cutoff Frequency
The Low Cutoff Frequency parameter determines the ideal cutoff frequency that a filter is designed to approximate. The low cutoff frequency of an ideal filter is the filter’s first transition band. The low cutoff frequency for a filter is specified in Hertz (Hz).

Valid low cutoff frequency values are in the range 2% to 80% of the Nyquist frequency. The Nyquist frequency is half the Sample Rate.
For a lowpass filter the first transition band is the transition from passband to stopband. For a bandpass filter the first transition band is the transition from stopband to passband.

**Low Cutoff Slope**

The **Low Cutoff Slope** parameter determines the ideal cutoff response that a filter is designed to approximate. The low cutoff slope of an ideal filter is the width of the filter’s first transition band. The low cutoff slope of a filter is defined as a percentage of the Nyquist frequency.

Valid low cutoff slope values are in the range 0% to 80% of the Nyquist frequency. The Nyquist frequency is half the *Sample Rate*.

**High Cutoff Frequency**

The **High Cutoff Frequency** parameter determines the ideal cutoff frequency that a filter is designed to approximate. The high cutoff frequency of an ideal filter is the filter’s second transition band. The high cutoff frequency for a filter is specified in Hertz (Hz).

Valid high cutoff frequency values are in the range 2% to 80% of the Nyquist frequency. The Nyquist frequency is half the *Sample Rate*.

For a bandpass filter the second transition band is the transition from passband to stopband.

**High Cutoff Slope**

The **High Cutoff Slope** parameter determines the ideal cutoff response that a filter is designed to approximate. The high cutoff slope of an ideal filter is the width of the filter’s second transition band. The high cutoff slope of a filter is defined as a percentage of the Nyquist frequency.

Valid high cutoff slope values are in the range 0% to 80% of the Nyquist frequency. The Nyquist frequency is half the *Sample Rate*.

**Attenuation**

The **Attenuation** parameter determines the response in the stopband of a filter.

Valid attenuation values are in the range 6.0 to 12.0.
Filter Response

The Filter Design Screen displays a filter’s amplitude response based on selected filter parameters.

The Filter Response may be displayed graphically by one of the following methods.

- Linear Display
- Linear Zoom Display
- Log Display
- Log Zoom Display
- Unit Step Display

The default amplitude response is a Log Display. To change the amplitude response, click the right mouse button over the Filter Design Screen and select the Y Display menu item. The Y Display menu item allows the user to change the amplitude response display units to either linear, linear zoom, log, log zoom or unit step.

The linear, linear zoom, log, and log zoom graphs display the amplitude response at frequencies from zero to the Nyquist frequency. At all frequencies above the Nyquist frequency, the iDSC board has attenuation of more than 96 dB. A unit step display shows the amplitude response of the filter in relation to a change in the input from zero to positive full scale.

There are several more options when you click the right mouse button over the Filter Design Screen. These options include:

- Crosshair Track
- Defaults Load
- Copy
- Paste

Holding the left mouse button down and dragging the mouse over the graph will display a crosshair with the x and y coordinates of a particular point. If the Crosshair Track menu item is checked, the crosshair will track the filter response curve. This feature is useful in determining the exact cutoff frequency, cutoff slope, or attenuation at a particular point.

Selecting the Defaults Load menu item will load the default filter response graphs for a particular sample rate. The default filter response graphs always have very flat passband characteristics and stop band attenuation of more than 96 dB.
The Copy menu item will copy the filter design parameters from the filter design to the clipboard. The Paste menu item will paste the copied filter design parameters from the clipboard to another filter design.
Linear Display

A **Linear Display** shows the linear amplitude response in the frequency domain. A linear display graph plots the linear attenuation from 0 to 110%, with frequencies from zero to the Nyquist frequency. The Nyquist frequency is half the sample rate.

The filters used in the iDSC board have linear phase, so the phase response is specified by the **Group Delay**.

The following figure shows a linear display of the filter’s amplitude response.
Linear Zoom Display

A **Linear Zoom Display** shows the linear amplitude response in the frequency domain, with focus on the passband performance of the filter. A linear zoom display graph plots the linear attenuation, in 16-bit counts, from -16 counts to +16 counts, with frequencies from zero to the Nyquist frequency. The Nyquist frequency is half the **Sample Rate**.

The following figure shows a linear zoom display of the filter’s amplitude response.
**Log Display**

A **Log Display** shows the logarithmic amplitude response in the frequency domain. A logarithmic display graph plots the logarithmic attenuation, in decibels, from -120 dB to 10 dB, with frequencies from zero to the Nyquist frequency. The Nyquist frequency is half the **Sample Rate**.

The following figure shows a logarithmic display of the filter’s amplitude response.
Log Zoom Display

A Log Zoom Display shows the logarithmic amplitude response in the frequency domain, with focus on the passband performance of the filter. A logarithmic zoom display graph plots the logarithmic attenuation, in decibels, from -4 dB to +1 dB, with frequencies from zero to the Nyquist frequency. The Nyquist frequency is half the Sample Rate.

The following figure shows a logarithmic zoom display of the filter’s amplitude response.
**Unit Step Display**

A **Unit Step Display** shows the amplitude response of the filter in relation to a change in the input from zero to positive full scale. Note that the response to a unit step change in the input is spread out over many samples. This is a necessary consequence of the very sharp cutoff in the frequency domain. It is possible to sharpen the unit step input response by increasing the cutoff slope parameter and decreasing the sharpness parameter.

The following figure shows a unit step display of the filter’s amplitude response.
External Board Parameters

The following External Board Parameters are accessible from the External Board Screen.

Input Type
The Input Type parameter configures the type of input signal. Valid input types include DC coupling, AC coupling and Excitation.

Input Range
The Input Range parameter configures the input range. Please note that this input range is different from the iDSC Input Range, and only works if the iDSC input range is set to +/- 5V.

Valid input ranges are:
- +/- 10 mV, +/- 20 mV, +/- 50 mV,
- +/- 100 mV, +/- 200 mV, +/- 500 mV,
- +/- 1 V, +/- 2 V, +/- 5 V, +/- 10 V

Input Offset
The Input Offset parameter configures the input offset. The specified input offset must be in the Input Offset.

Input Offset Range
The Input Offset Range parameters displays the valid input offset range. The input offset range is determined by the selected Input Range.

For example, if the input range is +/- 500 mV then the input offset range is +/- 2.5 V, if the input range is +/- 2 V then the input offset range is +/- 1V. This is a read only parameter that is dependent on the input range and cannot be specified.

Valid input offset ranges are:
- +/- 0.5 V when the input range is +/- 10 mV
- +/- 1.0 V when the input range is +/- 20 mV
- +/- 2.5 V when the input range is +/- 50 mV
- +/- 0.5 V when the input range is +/- 100 mV
- +/- 1.0 V when the input range is +/- 200 mV
- +/- 2.5 V when the input range is +/- 500 mV
- +/- 1.0 V when the input range is +/- 1 V

DSC Graphical Design
Output Excitation

The **Output Excitation** parameter configures the output excitation. Valid output excitation voltages include 0 V, 1 V, 2 V, 5 V, 10 V.

\[
\begin{array}{ll}
  \pm 1.0 \text{ V} & \text{when the input range is } \pm 2 \text{ V} \\
  \pm 5.0 \text{ V} & \text{when the input range is } \pm 5 \text{ V} \\
  \pm 5.0 \text{ V} & \text{when the input range is } \pm 10 \text{ V}
\end{array}
\]
3. DSCview

DSCview provides immediate and easy access to one or more iDSC boards. From DSCview, communication with an iDSC board is automatically initiated and requires no programming.

DSCview provides a graphical interface to:
- Save and load workspaces
- Select system options
- Design and configure filters
- Perform signal conditioning
- Configure multiple iDSC boards
- Output data in a graph and table
- Disk log data to a text or binary file
- Server disk log data to a binary file by the server

DSCview is an invaluable diagnostic tool for confirming the proper operation of an iDSC board and for testing filter characteristics.

The following five windows are available in DSCview, and are discussed in detail in this chapter.
- Configuration Window
- Graph Window
- Table Window
- Disk Log Window
- Server Disk Log Window
Save and Load Workspaces

In DSCview you can save current workspaces and load new or saved workspaces. The extension for DSCview workspaces is .CFG.

The File menu item on DSCview’s main menu contains options to save and load workspaces.

The following File menu options are discussed in detail below.

- **File|New**
  - Opens a new workspace. If changes have been made to the current workspace, DSCview prompts you to save the changes.

- **File|Open**
  - Opens an existing workspace. If changes have been made to the current workspace, DSCview prompts you to save the changes.

- **File|Close**
  - Closes an opened workspace. If changes have been made to the current workspace, DSCview prompts you to save the changes.

- **File|Save**
  - Saves the current workspace. If a filename has not yet been assigned, DSCview prompts you for a filename.

- **File|Save As**
  - Saves the current workspace with a specified filename.

- **File|Exit**
  - Exits DSCview.

In addition to loading workspaces from the file menu, a configuration may also be automatically loaded from the command line as shown below. In this example, we are assuming that DSCview.exe and Graph.cfg are located in C:.

```
C:\DSCview C:\Graph.cfg
```

Start & Stop

The Start! and Stop! menu items are available from the main DSCview window.

Start!

The Start! menu item appears when DSCview is not acquiring data. When Start! is selected, data acquisition begins, and Start! changes to Stop!. 
While DSCview is acquiring data the following items become gray and cannot be changed.

- File and System Options menu items of the main system.
- Selectable options of the Configuration window.
- Graph Options|History and Graph Options|Display menu items of the Graph Window.
- Table Options|History and Table Options|Display menu items of the Table Window.
- Channels and Disk Log Options menu items of the Disk Log Window.

Stop!
The Stop! menu item appears when DSCview is acquiring data. When Stop! is selected, data acquisition terminates, and Stop! changes to Start!.

**System Options**

The System Options are available from the main DSCview menu, and are accessible from all active windows.

The following System Options are discussed in detail below.

- **System|Address**
- **System|Board Setup Display**
- **System|Calibrate**
- **System|Commands Load**
- **System|Configuration Lock**
- **System|Memory Display**

**System|Address**
The System|Address menu item selects the iDSC board to communicate with. When selecting the iDSC board, System|Address uses the Universal Naming Convention. The default iDSC board name is \\Dap0 for DSC0, \\Dap1 for DSC1, \\Dap2 for DSC2, etc. For detailed information about the Universal Naming Convention, please refer to the Programming Interfaces section of this document.

**System|Board Setup Display**
The System|Board Setup Display menu item displays the setup configuration of a group of iDSC boards in the Configuration Window. Applications that have more than one iDSC board must enable this menu item to gain access to multiple iDSCs boards.
**System|Calibrate**  
The **System|Calibrate** menu item calibrates the iDSC board for DC gain and offset. The first time **Start!** is selected, the iDSC board is automatically calibrated. The progress of iDSC calibration is displayed in the bottom border of DSCview.

**System|Commands Load**  
The **System|Commands Load** menu item configures the iDSC board with the appropriate programs and coefficients. **System|Commands Load** should be selected only when the iDSC board configuration or filter designs change.

If the iDSC board configuration or filter designs change and **Start!** is selected, **Start!** automatically invokes **System|Commands Load** and downloads new commands to the iDSC board. Data will show up at the PC two times Group Delay seconds later.

If the iDSC board configuration or filter designs change and **System|Commands Load** is selected before **Start!**, data show up immediately at the PC. Data that show up immediately are actually data that were sampled group delay seconds ago.

**System|Configuration Lock**  
The **System|Configuration Lock** menu items locks the configuration of the iDSC. This means that the number of iDSC boards, address and mode, and the Input Screen and Filter Design Screen parameters cannot be changed until the configuration is unlocked.

**System|Memory Display**  
The **System|Memory Display** menu item displays the used memory on the iDSC board. **System|Memory Display** is useful in determining whether or not the iDSC board will be able to sustain a particular sample rate without overflowing. The memory used is displayed in the bottom border of DSCview.
**Configuration Window**

The **Configuration Window** is designed to simplify the iDSC board input configuration and filter design process. The Configuration Window is a derivative of **DSC Graphical Design** interface. For detailed information about the Configuration Window, please refer to the **DSC Graphical Design** chapter of this document.

The following figure displays the **Input Screen** of the Configuration Window.
The following figure displays the Filter Design Screen of the Configuration Window.
The following figure displays the Filter Design Screen of the Configuration Window with **System|Board Setup Display** checked.
Graph Window

The Graph Window displays graphical representations of acquired and filtered data from the iDSC board. Only data from the selected channels are displayed. Any number of graph windows are allowed. To select a new graph window, select Window|New Graph.

When the Graph Window is active, the Channels and Graph Options menu items are available on the main menu. The Channels menu item displays a button grid of the enabled input pins, and the Graph Options menu item contains several options for configuring the graphical display. The Graph Options menu items are described later.

The Graph Window has zooming features. It also has channel data tracking features. These features are described later.

The following figure displays the Graph Window.
**Channels**

The Channels menu item displays a button grid of enabled iDSC board input pins. A channel is selected when the button is down and deselected when the button is up. There can be up to sixteen channels selected.

**Graph Options|Enabled**

The Graph Options|Enabled menu item enables or disables the Graph Window. A check mark beside Enabled signifies that the Graph Window is enabled. If the Graph Window is disabled and DSCview is started, no data are plotted.

**Graph Options|History...**

The Graph Options|History menu item determines the number of data points to store in the Graph Window. Any number of data points from 100 up to 10000 may be stored.

When DSCview is started, the Graph Options|History menu item is grayed and cannot be changed.

**Graph Options|Display...**

The Graph Options|Display menu item modifies the Graph Window display. Display items which can be modified include the titles, left axis, bottom axis, and series colors.

When DSCview is started, the Graph Options|Display menu item is grayed and cannot be changed.
The following figure displays the Graph Options|Display dialog.

The **Titles** box is used to select titles for the main graphical display, left axis and bottom axis. The default titles are Filtered Data, Volts and Sample.

The **Left Axis** box is used to configure the left axis for displaying data in Voltage, Digital, or Custom units. Full Scale is used internally to calculate an appropriate scale factor for the left axis display. The scale factor is \( \frac{\text{Full Scale}}{32768} \).

Using Voltage units, Full Scale is either 5 or 10 because the input voltage range is either -5 to +5 volts or -10 to +10 volts. The scale factor is \( \frac{5}{32768} \) or \( \frac{10}{32768} \).
Using Digital units, Full Scale is 32768 because the integer range is -32768 to 32767. The scale factor is \( \frac{32768}{32768} \) or no scaling. Using Custom units, Full Scale is a number, \( x \) that corresponds to a 5 or 10 volt reading. The scale factor is \( \frac{x}{32768} \).

\( \text{Min} \) and \( \text{Max} \) are used to define a range for graphically displaying the data. The display range must fall between \(-\text{Full Scale}\) and \(+\text{Full Scale}\) or an error box will appear. \( \text{Inc} \) displays the spacing between the left axis ticks in the \( \text{Min} \) to \( \text{Max} \) range. If \( \text{Inc} \) is too small to fit all of the ticks on the display, \( \text{Inc} \) will be automatically adjusted.

The Bottom Axis box is used to configure the bottom axis for displaying data in Sample, Seconds, or Milliseconds units. The number of points that the user wishes to view per graph display are entered into the View Points field. Any number of points within the range 100 to 1024 are valid.

The Series Colors box is used to select from sixteen different colors to represent the selected channels. The legend on the right side of the Graph Window displays the selected channels and their corresponding colors. Any sequence of up to sixteen colors may be displayed.

**Zooming**

The Graph Window has zooming in features. To zoom in, place the cursor over the points of interest and use the left mouse button to create a rectangular box. When the left mouse button is released, the area inside the rectangular box is enlarged. You can zoom in indefinitely but the Graph Window only keeps a history of the five most recent zoomed in levels. There is also a minimum zoom in of 20 samples for the x axis and 20 counts for the y axis.

To view the saved zooms, use the right mouse button. Zoom Level shows which saved zoomed in level is currently displayed, with a maximum of five zoomed in levels. It is read only and non selectable. The Zoom Level of 0 is the original unzoomed display. Zoom Next displays the next saved zoom and Zoom Back displays the previous saved zoom. Zoom Clear clears all saved zooms.

**Data Tracking**

The Graph Window has channel data tracking features. To track the data of a channel, click and hold down the left mouse button over the channel of interest from the legend located on the right side of the Graph Window. Then drag the cursor over the plotting screen, and the corresponding data values will be located at the bottom left corner of the Graph Window.
**Table Window**

The Table Window displays text representations of acquired and filtered data from the iDSC board. Only data from the selected channels are displayed. Any number of table windows are allowed. To select a new table window, select **Window|New Table**.

When the Table Window is active, the **Channels** and **Table Options** menu items are available on the main menu. The **Channels** menu item displays a button grid of the enabled input pins, and the **Table Options** menu item contains several options for configuring the table display. The **Table Options** menu items are described later.

The following figure displays the Table Window.
Channels
The Channels menu item displays a button grid of enabled iDSC board input pins. A channel is selected when the button is down and deselected when the button is up. There is no limit on the number of channels selected.

Table Options|Enabled
The Table Options|Enabled menu item enables or disables the Table Window. A check mark beside Enabled signifies that the Table Window is enabled. If the Table Window is disabled and DSCview is started, no data are displayed.

Table Options|History...
The Table Options|History menu item determines the number of data points to store in the Table Window. Any number of data points from 100 to 10000 may be stored.
When DSCview is started, the Table Options|History menu item is grayed and cannot be changed.

Table Options|Display
The Table Options|Display menu item modifies the Table Window display. Display items which can be modified include the column widths and row heights.
In addition to modifying the column widths and row heights as a group, each column in a display may have a different width. To change the width of an individual column, place the mouse over the column separators and click the left mouse button and then drag the mouse left or right until you have obtained the desired column width.
When DSCview is started, the Table Options|Display menu item is grayed and cannot be changed.
**Disk Log Window**

The Disk Log Window logs acquired and filtered data from the iDSC board to a disk file in binary or text format. Only data from the selected channels are logged. Any number of disk log windows are allowed. To select a new disk log window, select Window|New Disk Log. Each disk log window only allows logging data from within the same iDSC.

When the Disk Log Window is active, the **Channels** and **Disk Log Options** menu items are available on the main menu. The **Channels** menu item displays a button grid of the enabled input pins, and the **Disk Log Options** menu item contains several options for logging data to disk. The **Disk Log Options** menu items are described later.

The following figure displays the Disk Log Window.
**Channels**

The Channels menu item displays a button grid of enabled iDSC board input pins. A channel is selected when the button is down and deselected when the button is up. There is no limit on the number of channels selected.

When the channels are selected or deselected, the Disk Log Window updates the selected channels under the Input Sources label. Each disk log window only allows channel selection within the same iDSC.

When DSCview is started, the Channels menu item is grayed and cannot be changed.

**Disk Log Options|Open File**

The Disk Log Options|Open File menu item opens a new file for logging data to disk. When a file is opened, the filename is displayed under the Output File label.

Selecting either the .BIN or .TXT extension from List files of type automatically marks the Disk Log Options|Format as Binary or Text. Note that all file extensions must be explicitly typed.

When DSCview is started, the Disk Log Options menu item is grayed and cannot be changed.

**Disk Log Options|Close File**

The Disk Log Options|Close File menu item closes the opened log file.

To view a log file in an editor or environment other than DSCview, it is important to first close the log file in DSCview before switching to the other environment.

**Disk Log Options|Format**

The Disk Log Options|Format menu item selects the format for logging data to disk. Data may be formatted as binary or text, and a check mark beside a formatting option indicates which format is selected. Data logged as text are delimited by tabs.

When a new file is opened with the Disk Log Options|Open File menu option, the file extension may be selected using List files of type. Selecting either the .BIN or .TXT extension from List files of type automatically marks the Disk Log Options|Format as Binary or Text.
Server Disk Log Window

The Server Disk Log Window is available only when server disk logging is enabled by clicking the right mouse button in the Group Interface and selecting Server Disk Log. This window displays the iDSC board name, the number of samples logged, and the output file name.

The following figure displays the Server Disk Log Window.
4. Using the iDSC Board with DASYLab

The iDSC board includes a driver to seamlessly integrate with DASYLab 4.x. Combining DASYLab with the iDSC board provides many convenient features:
• Access to standard iDSC board filter parameters through an iDSC board module
• iDSC board channels wired directly to other DASYLab modules
• Additional iDSC board modules to access more than one iDSC board
• All iDSC board parameters saved with the DASYLab worksheet
• Master/slave mode used to synchronize several iDSC boards

Installing iDSC Board Support for DASYLab

When the iDSC board software is installed, a directory named \IDSC\APPSW\DLAB is created that contains DASYLab support files. Copy this directory and its contents to the DASYLab installation directory. The following example displays how to copy the files in a DOS shell:

```bash
XCOPY    *.*   “C:\PROGRAM FILES\DASYLAB”
```

Note: Replace the directory names above with the correct paths for your PC.

To enable the iDSC board menu, edit the file DASYLAB.INI located in the Windows directory. Find the section named [EXTEND] and modify the DLL1 line to appear as follows:

```ini
DLL1=DLAB_DSC.DLL
```

Start DASYLab to verify that installation was successful. A new menu option will appear in the main menu titled “Microstar iDSC.” Use File/Open to open the example DSC1.DSB in the DASYLab directory. The example will display an iDSC board module and will show one trace on a graph when the worksheet is started.
Using an iDSC Board Module with DASYLab

To place an iDSC board module on the worksheet, select New iDSC from the “Microstar iDSC” menu. An iDSC board module will appear with one active channel. The outputs of the iDSC board module may be wired to other DASYLab modules.

To configure the iDSC board, double-click on the iDSC board module to display the standard iDSC board configuration screen. You may activate additional channels and configure the filter parameters for each channel. The iDSC board configuration dialog box used in DASYLab is the same dialog box used by DSCview and other iDSC board applications.

Running the DASYLab iDSC Board Examples

Several examples show how DASYLab interfaces with the iDSC board. The examples are located in the IDSC directory under the DASYLab install directory. To run the examples, verify that the iDSC board and iDSC board software are installed and that DSCview works properly. Exit DSCview before running the DASYLab iDSC board examples. The DASYLab examples are titled “DSCx.DSB” where x is the example number.

Using More Than One iDSC Board with DASYLab

Up to fourteen iDSC boards may be used with DASYLab. Select New iDSC from the “Microstar iDSC” menu to create a new module for each board. An iDSC board must be installed for each new iDSC board module placed on the DASYLab worksheet or an error message will be displayed. Configure each new module in the same manner as the first iDSC board module.
Synchronizing Several iDSC Boards

The DASYLab iDSC board module supports master/slave mode which allows synchronization between several iDSC boards. Synchronization provides true simultaneous sampling across several iDSC boards. To enable master/slave mode, select *Use Master/Slave Mode* from the “Microstar iDSC” menu.

In DASYLab, the first iDSC board, *iDSC0*, is always the master board. All additional iDSC boards are slave boards. Slave iDSC boards are automatically configured to use the same sample rate as the master iDSC board. Note that this automatic configuration will cause iDSC board parameters to change if the sample rate is not already set to the same value as that of the master iDSC board. For this reason, we recommend saving the worksheet before selecting the *Use Master/Slave Mode* option.

---

**Note:** Refer to the *Multiple Board Installation* section of this document for hardware configuration details of master/slave boards.

Special Addressing

There may be times when DASYLab needs to access an iDSC board that is not at the default UNC address. This may occur when an iDSC board is used in a PC with DAP boards, or when DAPcell is used to access an iDSC board in another PC. To change the default address that DASYLab uses edit the file *DASYLAB.INI*. Add the following section and key words:

```
[iDSC]
Addr0 = \\.dap0
```

Modify the address to match the desired address. Additional lines may be added for more than one iDSC board.
5. Using the iDSC Board with LabVIEW

LabVIEW provides a graphical programming environment for which the iDSC board provides an ideal signal conditioning and data acquisition front end. A LabVIEW application can call iDSC board functions in the DSCIO DLL to acquire the full programmability and power of the iDSC board.

LabVIEW accesses iDSC board functions in DSCIO DLL through the Call Library Function node, which is a VI library function in LabVIEW. It is easy to use with a dialog box to configure all of the parameters required for each function.

Microstar Laboratories simplifies the task further by providing this package - iDSC Board support for LabVIEW. This package contains this reference manual and DSC.LLB, which contains the following components.

1. A collection of Call Library Function nodes for each functions in DSCIO DLL
2. Several subVIs to aid reducing the development time for an application
3. Several examples to show the use of many of the Call Library Function nodes and subVIs.

The iDSC support for LabVIEW supports LabVIEW version 5.1 or later.

Installation

The following steps show the procedure on installing iDSC Board support of LabVIEW.

1. LabVIEW must be installed correctly. Please refer to the LabVIEW installation instructions.

2. Install iDSC Development from the DAPtools CD. The default installed directory is C:\Program Files\Microstar Laboratories\iDscDev. The iDSC Board Support for LabVIEW can be found in the APPSW\BIN subdirectory.

3. To verify the iDSC board and its software are installed and running properly, please run DSCview in the APPSW\BIN subdirectory. Exit DSCview before running any iDSC board examples or applications in LabVIEW.
Creating an iDSC Board Application in LabVIEW

A new application in LabVIEW may be created by modifying one of the examples described below. The following section provides an outline about how LabVIEW interfaces to the iDSC board.

An iDSC board application should perform the following steps:
1. Opens a handle to an iDSC board with the function `DscHandleOpen`.
2. Defines the configuration of the board using a dialog box, a saved configuration file, or iDSC board configuration functions.
3. Starts data acquisition with the function `DscStartAcquiring`.
4. Reads and processes the acquired data.
5. Stops data acquisition with the function `DscStopAcquiring`.
6. Closes the handle to the iDSC board with the function `DscHandleClose`.

The iDSC Init subVI provides all of the initialization routines, step 1 to 3, in one subVI.

After initialization, an application can read data from the iDSC board by using either `DscBufferGet` or `DscBufferGetEx`. The iDSC Read subVI performs the read data routines, step 4, in one subVI.

The iDSC Close subVI provides all of the termination routines, step 5 and 6, to stop acquisition and terminate communication with the iDSC.

It is important to terminate the communication with the iDSC board using the provided functions in DSCIO DLL. The STOP button on the button bar does not stop the iDSC board, meaning THE iDSC WILL CONTINUE TO RUN EVEN THOUGH THE LABVIEW STOP BUTTON WAS PRESSED. It is best to put a button on the user panel that controls the termination of a run. The STOP button should enable the final sequence to run the iDSC Close subVI which closes the ACCEL32 Handle assigned to the application. Please see examples on how to accomplish this.

Running the LabVIEW iDSC Board Examples

Several examples applications are included and demonstrate how LabVIEW interfaces with an iDSC board. The examples can be found in DSC.LLB in the APPSW\LABVIEW subdirectory under the installed directory. To run the examples, an iDSC board and its software have to be installed and running properly.
App01 - BASIC

This example provides a very easy to use interface to an iDSC board. It uses one object, iDSC subVI, to provide access to iDSC configuration options and resulting data. This example reconfigures the iDSC board for each new block of data. For continuous operation and even more flexible options see the next example, App02.

App02 - GRAPH

This example is similar to App01 but adds transferring data continuously and configurability in accessing an iDSC board. This example uses an iDSC Init subVI to initialize iDSC board communication. The iDSC Init subVI will return the number of active channels. Based on this information, an iDSC Read subVI is used to read blocks of data and send the data to a graph, which automatically configures itself for the proper number of channels.

App03 - LOG

This example is similar to App02 but adds logging data to a file. Note that LabVIEW may use a special data format for data storage that may need conversion if read by applications other than LabVIEW.

App04 - LOGVW

This example shows how to read the file created by App03. This example is configured to read data for eight channels. If the data file is acquired for a different number of channels, the Number of Channels field needs to be modified accordingly.

App05 - DaplFFT

This example shows how to use DAPL interface. It configures an iDSC board with DAPL commands, reads data, and graphs the data with one trace for each channel. In this example, the DAPL commands configure the iDSC board to calculate forward fast Fourier transforms (FFT) of blocks of real-values data and send an amplitude spectrum to this example in LabVIEW. The amplitude spectrum received from iDSC Read will be sent to a graph for display.

App06 - DaplCC

This example is similar to App05 but applies custom commands BZTRUNC and RAVE to input channels instead of FFT. The BZTRUNC command truncates any number
below 0 in an input channel, and RAVE computes the running average of the specified number of data points, 100 in this example, in an input channel. In this example, both raw and analyzed data for the input channels are sent to this example. The data received from iDSC Read will be sent to a graph for display.

**App07 - Disk Logging (1 iDSC)**

This example shows how to stream data directly to a disk file by using DAPcell Server disk logging service. This is implemented by the DiskLog subVI, which loads configurations to a server by a wrapper function. While the server is logging data to a disk file, the number of data being logged will be queried by using a Num Data subVI.

The server continues to log data to the disk file until the specified amount of data in DiskLog subVI, which is 100000 values per channel in this case, have been recorded, or the STOP button is pressed.

Before running this example, please make sure the following configurations are correct.

- The disk logging option is enabled and a valid default path is entered in Windows Control Panel | Data Acquisition Processor | Disk I/O.

**App08 - Disk Logging (2 iDSC with synchronization)**

This example is similar to App07 but adds one more iDSC board. The two iDSC boards are synchronized as master and slave by using iDSC MaSl subVI. Similar to App07, this example logs data to a disk file by a server and queries for the number of byte being logged to each file for each board.

The server continues to log data to the disk file until the specified amount of data in DiskLog subVI, which is 100000 values per channel in this case, have been recorded for each boards, or the STOP button is pressed.

Before running this example, please make sure the following configurations are correct.

- Two iDSC 1816 boards are connected by a synchronization cable, with part number MSCBL 078.
- The disk logging option is enabled and a valid default path is entered in Windows Control Panel | Data Acquisition Processor | Disk I/O.
This example shows how to use group services provided by DSCIO DLL. In this example a DSCIO DLL function DscGroupConfigDialogShow is called to display a modal dialog screens for graphical configuration of multiple iDSC boards. It loads a configuration file GROUP.DSC for a group of two iDSC boards, which are configured as independent. The two iDSC boards can be synchronized by setting the appropriate mode. For more information, please see DscGroupConfigDialogShow.

The iDSC Read subVI are used to read blocks of data from the iDSC boards. The data will be sent data to graphs, which automatically configures themselves for the proper number of channels based on the dimension of the data array.
## DLL Reference

This package contains a list of Call Library Function nodes that are configured to call the functions in DSCIO DLL and MSLAPP DLL. The list can be found in DSC.LLB, which is located in the APPSW\LABVIEW subdirectory under the installed directory.

### DSCIO DLL Function Reference

This DSCIO DLL provides a complete set of functions for communicating with iDSC boards. The table below shows a complete list of Call Library Function nodes and its corresponding functions in DSCIO DLL. The name of each Call Library Function node may slightly different than the corresponding function in DSCIO DLL, but they share the same parameters list. For more information on the parameter lists, please see DSCIO Reference manual.

<table>
<thead>
<tr>
<th>Call Library Function node in DSC.LLB</th>
<th>DSCIO DLL Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DscHandleOpenA</td>
<td>DscHandleOpen</td>
</tr>
<tr>
<td>DscHandleClose</td>
<td>DscHandleClose</td>
</tr>
<tr>
<td>DscCalibrate</td>
<td>DscCalibrate</td>
</tr>
<tr>
<td>DscCommandsLoad</td>
<td>DscCommandsLoad</td>
</tr>
<tr>
<td>DscStartAcquiring</td>
<td>DscStartAcquiring</td>
</tr>
<tr>
<td>DscStopAcquiring</td>
<td>DscStopAcquiring</td>
</tr>
<tr>
<td>DscBufferAvail</td>
<td>DscBufferAvail</td>
</tr>
<tr>
<td>DscBufferGet</td>
<td>DscBufferGet</td>
</tr>
<tr>
<td>DscBufferGetEx</td>
<td>DscBufferGetEx</td>
</tr>
<tr>
<td>DscConfigRead</td>
<td>DscConfigRead</td>
</tr>
<tr>
<td>DscConfigWrite</td>
<td>DscConfigWrite</td>
</tr>
<tr>
<td>DscConfigWriteSize</td>
<td>DscConfigWriteSize</td>
</tr>
<tr>
<td>DscConfigDialogShow</td>
<td>DscConfigDialogShow</td>
</tr>
<tr>
<td>DscFilterNameGet</td>
<td>Obsolete</td>
</tr>
<tr>
<td>DscFilterNameSet</td>
<td>Obsolete</td>
</tr>
<tr>
<td>DscFilterParametersGet</td>
<td>DscFilterParametersGet</td>
</tr>
<tr>
<td>DscFilterParametersSet</td>
<td>DscFilterParametersSet</td>
</tr>
<tr>
<td>DscPinToFilterMapGet</td>
<td>DscPinToFilterMapGet</td>
</tr>
<tr>
<td>DscPinToFilterMapSet</td>
<td>DscPinToFilterMapSet</td>
</tr>
<tr>
<td>DscFilterIndexA</td>
<td>DscFilterIndex</td>
</tr>
<tr>
<td>DscGroupDelay</td>
<td>DscGroupDelay</td>
</tr>
<tr>
<td>DscAddressGetA</td>
<td>DscAddressGet</td>
</tr>
</tbody>
</table>
DscGroupConfigWrite (new)    DscGroupConfigWrite
DscGroupConfigWriteSize (new)  DscGroupWriteSizes

* For more information, please see obsolete interface in the DSCIO Function Summary.

**MSLAPP DLL Function Reference**
Due to the data types in LabVIEW, some DSCIO functions, which initialized by structures, cannot be accessed directly in LabVIEW. A wrapper function is implemented to interface between DSCIO DLL and LabVIEW. The wrapper function builds the structure, and passes it to the DSCIO function it interfaces with. All wrapper functions have the prefix Msl with the function for which they interface.

**DSCIO DLL Routines**

DscServerDiskLogConfigSet    MslDscServerDiskLogConfigSet

**Data Format**
The DSCIO DLL function reference uses C data types when specifying the type of each parameter. Here is a brief cross references to help determine corresponding LabVIEW data types:

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>LabVIEW Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char*</td>
<td>C String Pointer</td>
</tr>
<tr>
<td>Int</td>
<td>Signed 32-bit Integer</td>
</tr>
<tr>
<td>Short *</td>
<td>Signed 16-bit Integer</td>
</tr>
<tr>
<td>Long</td>
<td>Unsigned 32-bit Integer</td>
</tr>
<tr>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>HDSC</td>
<td>Signed 32-bit Integer</td>
</tr>
<tr>
<td>TbufferGetEx</td>
<td>Array Data Pointer of Signed 32-bit Integer. See iDSC Read subVI for an example of using this structure in LabVIEW. See DSCIO Reference Manual for details.</td>
</tr>
<tr>
<td>TDscIoInt64</td>
<td>Array Data Pointer of Signed 32-bit Integer. For example usage, please see Num Data subVI. For more information, please see DSCIO Reference Manual.</td>
</tr>
</tbody>
</table>
TfilterParam Array Data Pointer of Signed 32-bit Integer. For more information, please see DSCIO Reference Manual.

TxbPinConfig Array Data Pointer of Signed 32-bit Integer. For more information, please see DSCIO Reference Manual.

**MsIDscServerDiskLogConfigSet**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Corresponding parameters in TServerDiskLogConfig and brief descriptions</th>
<th>Output</th>
<th>Pass-through</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDsc (Integer)</td>
<td>---</td>
<td>dwDwFlags</td>
<td>1:</td>
<td>True</td>
</tr>
<tr>
<td>Flag (Long)</td>
<td>Specifies various logging options.</td>
<td>dwFlags</td>
<td>2:</td>
<td>True</td>
</tr>
<tr>
<td>LogFile (String)</td>
<td>Points to a null-terminated string that specifies the name of the disk logfile.</td>
<td>pszFileName</td>
<td>3:</td>
<td>True</td>
</tr>
<tr>
<td>FileShareMode (Long)</td>
<td>Specifies the file share properties of the disk logfile.</td>
<td>dwFileShareMode</td>
<td>4:</td>
<td>True</td>
</tr>
<tr>
<td>OpenFlags (Long)</td>
<td>Specifies how file opening is to be handled.</td>
<td>dwOpenFlags</td>
<td>5:</td>
<td>True</td>
</tr>
<tr>
<td>FileFlagsAttributes (Long)</td>
<td>Specifies additional file attributes.</td>
<td>dwFileFlagsAttributes</td>
<td>6:</td>
<td>True</td>
</tr>
<tr>
<td>BlockSize (Long)</td>
<td>Specifies the minimum amount of data, in bytes, to write to the</td>
<td>dwBlockSize</td>
<td>7:</td>
<td>True</td>
</tr>
</tbody>
</table>
logfile at one time.

8: \texttt{Lowi64}\texttt{(Long)} \texttt{I64MaxCount}
Specifies the low 32-bit maximum number of bytes to log.

9: \texttt{Highi64}\texttt{(Long)} \texttt{I64MaxCount}
Specifies the high 32-bit maximum number of bytes to log.

This \texttt{Call Library Function} node builds the structure \texttt{TServerDiskLogConfig} and passes it to \texttt{DscServerDiskLogConfigSet}, which initiates a disk logging session between an iDSC board specified by \texttt{hDsc} and a disk file specified by \texttt{LogFile}. The parameter \texttt{BlockSize} is provided for disk transfer optimization. The default value is 8192.

If a full path is not given for the parameter \texttt{LogFile}, the log file resides in the default directory specified in the Control Panel | Data Acquisition Processor | Disk I/O on the server PC.

The disk logging session starts when \texttt{DscStartAcquiring} is invoked. Once it starts, it continues until the number of bytes specified in \texttt{Lowi64} and \texttt{Highi64} has been logged or until \texttt{DscStopAcquiring} or \texttt{DscHandleClose} is invoked on \texttt{hDsc}.

If the function succeeds, the return value is 1. If the function fails, the return value is 0.
SubVIs Reference

This package provides several subVIs to make it easy to configure and cleanup iDSC board communication. In LabVIEW if you select Show | Help from the Help menu, and place the mouse cursor over the subVI, diagrams as shown below will appear. To place a subVI onto a LabVIEW diagram:

1. Right click on the diagram.
2. Choose “Select a VI”.
3. Open DSC.LLB and select the desire subVI.

In the following subVIs, most input parameters are optional. If the input parameters are missing, default values will be used.

iDSC Init

This subVI provides an easy way to configure an iDSC board. It allows the user to access different boards, download DAPL and custom commands, and select a previously saved iDSC board filter configuration.

iDSC Init has five inputs:
1. Custom Command list is a list containing names and size of custom commands.
   String.
   Default is (None).
2. iDSC configuration file contains the filename of an iDSC board configuration.
   String.
   Default is IDSC1.DSC.
3. Configuration Dialog Box Show determines whether the iDSC board configuration dialog box is displayed.
   Boolean.
   Default is TRUE
4. iDSC Path is an UNC path specifies the target iDSC board to be opened.
   String.
   Default is \\.

Using the iDSC Board with LabVIEW
5. DAPL Commands contains DAPL commands for configuring an iDSC board
   String.
   Default is (None).

iDSC Init has three outputs:
1. iDSC handle Pass Through is the handle of the iDSC board.
   Integer.
2. Sample rate reports how fast the iDSC is acquiring data.
   Integer.
3. Number of channels reports the number of active channel.
   Integer.

iDSC Init performs the following operations:
1. Opens a handle to the iDSC board specified by iDSC Path with the function
   DscHandleOpen.
2. Defines custom commands with DscDaplCCListSetA and enables downloading
   with DscDaplCCDownloadSet if Custom Command list is not equal to
   (None).
3. Defines DAPL commands with DscDaplTextSetA if DAPL Commands is not
   equal to (None).
4. Attempts to load the specified iDSC board configuration file with
   DscConfigRead, and defaults to the standard configuration.
5. If Show dialog box? input option is true, an iDSC board configuration dialog
   box will be loaded with DscConfigDialogShow.
6. Saves the configuration to a file with DscConfigWrite after a prompt to the user
   to replace an old file. If the file does not exist, it will be created. If no file name is
   specified, the default file will be used.
7. Begins data acquisition with the function DscStartAcquiring.
8. Gets the sample rate with the function DscSampleRateGet.
9. Gets the active channels with the function DscPinEnabledGet, counts and
   returns the number of active channels.

The inputs Custom Command list and DAPL commands normally are not used.
They can be used if data processing is desired.
If you are using more than one iDSC board, it generally is the best to make a copy of
iDSC Init, and provides an address to a target iDSC board and a configuration file if
there is any.

iDSC Read

This subVI gets one block of data from an iDSC board.
iDSC Read has five inputs:
1. **TimeOut** is the maximum amount of time in milliseconds that the get operation should complete. If it fails to complete in this amount of time, the service aborts the operation.
   - Integer.
   - Default is 10000.
2. **iDSC Handle** specifies the handle to the target iDSC board. It has to be passed by **iDSC Init**, or previously opened **DscHandleOpen**.
   - Integer.
   - Default is 0.
3. **ValuesToRead** specifies the number of data per channel should get for each operation.
   - Integer.
   - Default is 500.
4. **Number of Channels** is the number of active channel.
   - Integer.
   - Default is 1.
5. **TimeWait** is the maximum amount of time in milliseconds that the get operation can be blocked waiting for data. If no data show up in that amount of time, the service aborts the operation.
   - Integer.
   - Default is 10000.

iDSC Read has three outputs:
1. **iDSC Handle Pass Through** is the handle of the iDSC board being passed through.
   - Integer.
2. **Data** is a pointer to an array, which contains data from the iDSC board.
   - Array pointer.
3. **Return Code** reports the result of the get operation. If the get operation is succeeds, it contains the number of data bytes read. If the get operation fails, it contains -1.
   - Integer.

iDSC Read performs the following operations:
1. Builds and initializes an array for the required structure **TBufferGet**.
2. Builds and initializes a one-dimensional array for storing the returned data.
3. By passing the arrays to the function DscBufferGetEx, iDSC Read gets data from the iDSC board specified by iDSC Handle.
4. Re-dimension the returned data array as a two-dimensional array with size M by N, where M is ValuesToRead and N is Number of Channels.

**iDSC Close**

This subVI terminates the communication with the iDSC board.

![iDSC Close SubVI](image)

**iDSC Close** has one input:
1. iDSC Handle specifies the handle to the target iDSC board. It has to be passed by iDSC Init, or previously opened DscHandleOpen.
   Integer.
   Default is 0.

**iDSC Close** has no output:

**iDSC Close** performs the following operations:
1. Stops data acquisition on the iDSC board specified by iDSC Handle with the function DscStopAcquiring.
2. Terminates the communication with the iDSC boards specified by iDSC Handle with the function DscHandleClose.

**iDSC**

This subVI initiates a communication with an iDSC, reads one block of data, and terminates the communication.

![iDSC SubVI](image)

**iDSC** has one input:
1. ValuesToRead specifies how many data per channel should be get for each operation.
   Integer.
   Default is 500

**iDSC** has four outputs:
1. **Return Code** reports the result of the get data operation. If the get data operation is succeeds, this variable contains the number of data bytes read. If the get data operation fails, this variable contains -1.
   Integer.
2. **Data** is a pointer to an array, which contains data from the iDSC board.
   Array pointer.
3. **Sample rate** reports how fast the iDSC is acquiring data.
   Integer.
4. **Number of channels** reports the number of active channel.
   Integer.

iDSC performs the following operations:
1. Initiates a communication with iDSC Init subVI.
2. Reads a block of **Number of Values To Read** data with iDSC Read subVI.
3. Terminates the communication with iDSC Close subVI.

For example usage, please see App01.

**iDSC MaSI**

This sub-VI initiates a communication with two iDSC boards and synchronizes them by setting master/slave properties.

```
IDSC0 Path ----- DSC MaSI ------ IDSC0 Handle Pass Through
IDSC1 Path ----- DSC MaSI ------ IDSC1 Handle Pass Through
```

iDSC MaSI has two inputs:
1. **iDSC0 Path** is an UNC path specifies the target iDSC board to be opened, and it will be configured as a master board.
   String.
   Default is \\.\Dap0.
2. **iDSC1 Path** is an UNC path specifies the target iDSC board to be opened, and it will be configured as a slave board for iDSC0 Path.
   String.
   Default is \\.\Dap1.

iDSC MaSI has two outputs:
1. **iDSC0 Handle Pass Through** is the handle of the master iDSC board.
   Integer.
2. **iDSC1 Handle Pass Through** is the handle of the slave iDSC board.
   Integer.
iDSC MaSI performs the following operations:
1. Open handles to iDSC boards specified by iDSC0 Path and iDSC1 Path with the function DscHandleOpen.
2. Connects the slave board at iDSC1 Path and the master board at iDSC0 Path with DscMasterSet.

This subVI configures the boards in software only. The configuration for master and slave must be done in hardware by using a special cable. For more information, please see Master/Slave Configuration in the DSCIO Reference Manual.

For example usage, please see App08.

**DiskLog**

DiskLog has two inputs:
1. **ValuesToLog** specifies the amount of data will be logged from each active channel.
   - Double.
   - Default is 100000 data per channel.
2. **FileFlagAttributes** specifies additional file attributes.
   - Integer.
   - Default is 1, which means normal attributes.
3. **OpenFlags** specifies how file opening is to be handled.
   - Integer.
   - Default is 2, which means always open an existing file. If the file does not exist, it will be created.
4. **iDSC Handle** specifies the handle to the target iDSC board. It has to be passed by iDSC Init, or previously opened DscHandleOpen.
   - Integer.
   - Default is 0.
5. **DiskLogFlag** specifies various logging options.
   - Integer.
   - Default is 1, which means log on the same side of the network connection as the iDSC.
6. **LogFile** specifies the name of the log file.
   - String.
   - Default is *data.bin*.

7. **FileShareMode** specifies the file share properties of the *LogFile*.
   - Integer.
   - Default is 1, which means the file can be read by another process.

8. **BlockSize** specifies the minimum amount of data, in bytes, to write to the *LogFile* at one time.
   - Integer.
   - Default is 8192.

**DiskLog** has four outputs:
1. **Return code** is 1 if the server succeeds on setting the configuration for disk logging. It is 0 if the server fails.
   - Integer.
2. **iDSC handle Pass Through** is the handle of the iDSC board being passed through.
   - Integer.
3. **Error Buffer** contains the error message if it fails to configure for server disk logging.
   - String.
4. **Number of Values To Log** indicates the amount of data will be logged to *LogFile*. It is the product of number of active channels and **ValuesToLog**.
   - Double.

**DiskLog** performs the following operations:
1. Converts **ValuesToLog** to total number of bytes to read by multiplying **ValuesToLog**, number of active channels from **DscPinEnabledCount**, and two.
2. Calls the wrapper function `MslDscServerDiskLogConfigSet` to initiate server disk logging.
3. Sets the state of the server disk logging option with the function `DscServerDiskLogEnabledSet`.
4. Returns the number of values will be logged for all channels in **Number of Values To Log**.

The iDSC 1816 board has to be configured for server disk logging before start acquiring data. So, a **Disk Log** subVI has to be called before **DscStartAcquiring**.

If a full path is not given for **LogFile**, the log file resides in the default directory specified in the Control Panel | Data Acquisition Processor | Disk I/O on the server PC.
For example usage, please see App07 and App08.

**Num Data**

This subVI queries for the number of bytes being logged to a disk file by a server.

![Diagram](72x711 to 262x649)

**Num Data** has one input:
1. `iDSC Handle` specifies the handle to the target iDSC board. It has to be passed by
   - `iDSC Init`, or previously opened `DscHandleOpen`.
   - Integer.
   - Default is 0.

**Num Data** has one output:
1. `NumDataLogged` specifies how many data have been logged by the server.
   - Double.

**Num Data** performs the following operations:
1. Initializes a two-element data array.
2. Queries for the number of bytes being logged by a server with the function
   - `DscServerDiskLogBytes`.
3. Recovers the result and stores it in a `DOUBLE`.

This subVI queries for the number of bytes being logged to a disk file by a server. Since the function `DscServerDiskLogBytes` returns a 64-bit integer, which is not supported by LabVIEW, this subVI is created to interface between DSCIO and LabVIEW. The results are stored in two 32-bit integers. The first and second elements of `i64Count` represent the low and high 32-bit of the result respectively. The 64-bit result can be recovered by performing the following calculation:

\[
64\text{-bit result} = low\text{32-bit} + high\text{32-bit} \times 2^{32}
\]

The result of the calculation is stored in a `DOUBLE` in LabVIEW.

**Get Error**

This subVI gets the last error message that occurred in the DSCIO DLL.

![Diagram](72x221 to 165x246)

Using the iDSC Board with LabVIEW
Get Error has no input:

Get Error has one outputs:
   1. Error Buffer contains the error message. String.

Get Error performs the following operations:
   1. Creates and initializes a buffer to store the error message.
   2. Get the last error message with the function DscLastErrorTextGet.

This subVI gets the last error message occurred in the DSCIO DLL. An error occurs in the DSCIO DLL when a function call fails. It should be called immediately after the Call Library Function node of interest.
6. Using the iDSC Board with LabWindows/CVI

LabWindows/CVI provides graphical user-interface development with C code programming for which the iDSC board supplies an ideal signal conditioning and data acquisition front end. LabWindows/CVI uses DLL function calls to access the full programmability and power of the iDSC board.

Running the LabWindows/CVI iDSC Board Example

A standard example shows how LabWindows/CVI interfaces with the iDSC board. The example is located in the directory where IDSC software was installed under the subdirectory LWCVI. A typical location would be under “c:\program files\Microstar Laboratories\idsc\apps\lwcvi.”

To run the examples, verify that the iDSC board and iDSC board software are installed and that DSCview works properly. Exit DSCview before running the LabWindows/CVI iDSC board example. Set the working directory of LabWindows to the directory with the example. Load the file GRAPH.PRJ and select the “Run Project” menu option.

The example displays iDSC data in a strip chart. Select the buttons to configure, start, and stop the iDSC board.
Creating an iDSC Board Application in LabWindows/CVI

A new LabWindows/CVI application may be created by modifying the example described above. The following section provides additional details about how LabWindows/CVI interfaces to the iDSC board.

An iDSC board application performs the following steps:
1. Opens iDSC board handles with the function `DscHandleOpen`. This operation is generally performed in the main() function before `RunUserInterface()` is called.
2. Defines the configuration of the board using a dialog box, a saved configuration file, or iDSC board configuration functions. The example above uses a dialog box when a button is pressed. Please see the section on Programming Interfaces and the C examples for more details on using a saved configuration file or configuration functions.
3. Starts data acquisition. The example starts acquisition when a button is pressed.
4. Reads data. An idle event may be used to run a function that periodically check for data.
5. Stops acquisition and terminates communication. The example uses a button to control when acquisition is stopped.

After initialization, an application reads data from the iDSC board using the function `DscBufferGetEx`.

LabWindows/CVI calls iDSC board functions through DLL function calls. The section in this manual on Programming Interfaces describes the DLL functions. Use the example and the function descriptions as a reference for building your application. The C examples provided with the iDSC software also provide a useful resource for building applications in LabWindows/CVI.
7. Using the iDSC Board with HP VEE

HP VEE is a graphical programming environment for which the iDSC board provides an ideal signal conditioning and data acquisition front end. HP VEE uses objects and DLL function calls to acquire the full programmability and power of the iDSC board.

Installing iDSC Board Support for HP VEE

When the iDSC board software is installed, a directory named IDSC\APPSEM\HPVEE32 is created that contains HP VEE support files. Copy this directory and its contents to a directory named IDSC under the HP VEE installation directory. The following example displays how to copy the files in a DOS shell:

```
XCOPY   *.*   “C:\PROGRAM FILES\HP VEE 4.0” /S
```

Note: Replace the directory names with the correct paths for your PC.

Restart HP VEE to enable a new iDSC menu that provides access to iDSC board functions.
Running the HP VEE iDSC Board Examples

Several examples show how HP VEE interfaces with the iDSC board. The examples are located in the IDSC directory under the HP VEE install directory. To run the examples, verify that the iDSC board and iDSC board software are installed and that DSCview works properly. Exit DSCview before running the HP VEE iDSC board examples.

APP01.VEE

APP01.VEE shows how to configure the iDSC board to sample two channels and display the results in a graph. To run the example press the Start! button. To stop the example press the Stop! button on the worksheet rather than HP VEE’s normal Stop! button. Pressing the Stop! button on the worksheet allows termination code to execute before HP VEE stops.

APP02.VEE

APP02.VEE is similar to APP01.VEE but adds disk logging to save the data to a disk file.

APP03.VEE

APP03.VEE shows how to read the data that were saved in APP01.VEE.

APP04.VEE

APP04.VEE is similar to APP01.VEE but it samples eight channels and displays the results in a graph.
Creating an iDSC Board Application in HP VEE

A new HP VEE application can be created by modifying one of the examples described above. The following section provides additional details about how HP VEE interfaces to the iDSC board.

An iDSC board application performs the following steps:
1. Opens iDSC board handles with the function `DscHandleOpen`.
2. Defines the configuration of the board using a dialog box, a saved configuration file, or iDSC board configuration functions.
3. Starts data acquisition.

The `iDSC Init` object provides all of these initialization routines in one object. `iDSC Init` stores the iDSC board handle in a global variable. An HP VEE application must be designed so that `iDSC Init` executes before any other functions that access the iDSC board.

After initialization, an application reads data from the iDSC board using `iDSC Data` object.

HP VEE calls iDSC board functions through the `Call Function` object which calls functions stored in the DSCIO.DLL dynamic link library. HP VEE automatically configures a `Call Function` for any of the DSCIO.DLL functions. To configure a `Call Function`, load and run one of the examples. HP VEE loads a `DSCIO.VH` file with definitions for each function. From the Device menu select `Math & Functions`. Next, select `Compile functions and DSCIO` from the dialog box. A complete list of DSCIO functions appears to the right. Choosing any function in this list creates a `Call function` object with the proper inputs and outputs.

Refer to the `DSCIO Function Summary` in this document for complete descriptions of the iDSC board functions.
Object Reference

iDSC Init

The iDSC Init object makes it easy to configure and initialize the iDSC board. Three buttons on the iDSC Init object provide configuration options. A New button displays the iDSC board configuration dialog box and saves the configuration. A Select button selects an existing configuration file. An Edit button displays the iDSC board configuration dialog box used to edit an existing file. At run-time iDSC Init opens a handle to the iDSC board, downloads the configuration, and starts acquisition.

iDSC Data

The iDSC Data object reads data from the iDSC board. Data values are provided on eight output pins corresponding to eight channels. If fewer channels are actively being sampled, the data for those channels appear consecutively on the iDSC Data outputs. For example if iDSC board inputs 0 and 7 are sampled, the data appear on the first and second pins of iDSC Data.

iDSC Data reads 200 values from each active channel each time it is called. The number of values on each read may be changed by editing the object and modifying the constant named Values.

iDSC Close

The iDSC Close object terminates data acquisition and closes the iDSC board handle.
8. Using the iDSC Board with MATLAB

The iDSC board support for MATLAB allows a user to communicate with the iDSC board using the MATLAB programming environment. It integrates the processing power of an iDSC with the powerful technical computing environment of MATLAB.

MATLAB performs high-level numeric computation and visualization of data sets, iDSC board support for MATLAB allows acquired and processed data from an iDSC to be further manipulated under MATLAB to suit the specific requirements of an application.

The iDSC board support for MATLAB supports MATLAB version 6.0.

Installing iDSC Board Support for MATLAB

When the iDSC board support software is installed, a directory is created that contains iDSC board support for MATLAB. The default name for this directory path is usually C:\Program Files\Microstar Laboratories\iDSCdev\APPSW\Matlab. This discussion assumes that the default location was used.

This directory must be added to the MATLAB search path. The MATLAB search path is established in the toolbox\local\MATLABRC.M file, located in the directory where MATLAB is installed. One way to add the directory is to select it as the current directory at the top of the MATLAB main window. The best way is to add an addpath command to the STARTUP.M file. If STARTUP.M does not exist, you can create one with a text editor. MATLABRC.M will run STARTUP.M when a MATLAB application starts.

Below is a description of adding a search path; for iDSC board software for MATLAB; to the MATLAB search path.

1) Create STARTUP.M, if it does not already exist, under toolbox\local in the MATLAB installation directory.

2) Add the following lines to STARTUP.M.

    disp('Adding iDSC board support to MATLAB search path');
    addpath(["C:\Program Files\Microstar Laboratories\", ...
             'iDSCdev\AppSw\Matlab'], path);
**Using an iDSC Board with MATLAB**

MATLAB cannot directly call DLL functions in DSCIO, but it can call and execute DLL MEX functions. The DLL MEX functions provide an interface to the DSCIO DLL, which is the interface to Accel32 and; iDSC hardware and drivers. The name of each DLL MEX function is slightly different than the corresponding function in DSCIO DLL. Also, the usage of some of the functions may vary slightly.

The table below shows the comparison between DLL MEX functions and DSCIO DLL routines. Please refer to the DLL MEX function reference provided in this manual for a description of each function.

<table>
<thead>
<tr>
<th>DLL MEX Functions for MATLAB</th>
<th>Corresponding DSCIO.DLL Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle = dscopen ('unc_path')</td>
<td>DscHandleOpen</td>
</tr>
<tr>
<td>&lt;code&gt; = dscclose (Handle)</td>
<td>DscHandleClose</td>
</tr>
<tr>
<td>Numbytes = dscavail (Handle)</td>
<td>DscBufferAvail</td>
</tr>
<tr>
<td>[data,&lt;retval&gt;] = dscbufg (Handle, Length, &lt;LengthMax, TimeWait, TimeOut, BytesMultiple&gt;)</td>
<td>DscBufferGetEx</td>
</tr>
<tr>
<td>&lt;boolean&gt; = dsccal (Handle)</td>
<td>DscCalibrate</td>
</tr>
<tr>
<td>&lt;boolean&gt; = dsccmdld (Handle)</td>
<td>DscCommandsLoad</td>
</tr>
<tr>
<td>[&lt;retVal&gt;] = dsccfgrd (Handle,buffer)</td>
<td>DscConfigurationRead</td>
</tr>
<tr>
<td>[data,&lt;retval&gt;] = dsccfgwt (Handle)</td>
<td>DscConfigurationWrite</td>
</tr>
<tr>
<td>&lt;boolean&gt; = dscdialog (Handle)</td>
<td>DscFilterDialogShow</td>
</tr>
<tr>
<td>[FilterType, Sharpness, CoFL, CoSL, CoFH, CoSH, Attenuation, &lt; retval&gt;] = dscfpget (Handle,FiltIndex)</td>
<td>DscFilterParametersGet</td>
</tr>
</tbody>
</table>
Using the iDSC Board with MATLAB

```
[<retVal>] = dscfpset (Handle, FiltIndex, FiltType, Sharpness, CutoffFreqLow, CutoffSlopeLow, CutoffFreqHigh, CutoffSlopeHigh, Attenuation)

[<retVal>,<retval>] = dscgdela (Handle)

[string] = dscerrg

[<retVal>] = dscmaset (SlaveHandle, MasterHandle)

<mempercent> = dscmemus (Handle)

<count> = dsccpescnt (Handle)

<pins> = dsccpescnt (Handle)

<boolean> = dsccpeset (Handle)

<rate> = dscsrget (Handle)

[<rate,<retVal>] = dscsrset (Handle, SampleRate)

<boolean> = dscstart (Handle)

<boolean> = dsccstop (Handle)

<boolean> = dscteset (Handle,par1,par2,par3)

ret = dsctccnt(Handle)

<max> = dsctcmax (Handle)

<width> = dsctcwdt (Handle)

[buf,<retVal>] = dscstfget (Handle, FiltIndex, Length)

[buf,<retVal>] = dscustget (Handle, FiltIndex, Length)
```
Using the iDSC Board with MATLAB

```matlab
[r, v] = dscdcls(Handle, 'CC list', stack_size)  
DscDaplCCListSetA

[r, v] = dscdtset(Handle, 'dapl text', stack)  
DscDaplTextSetA

[unc_path, r] = dscaddr(dscHandle)  
DscAddressGet

<bool> = dscaddr(dscHandle, 'unc_path')  
DscAddressSet

[scansDiscarded, bool] = dscdscan(handle)  
DscScanDiscarded

masterHandle = dscmaget(slaveHandle)  
DscMasterGet

[numBytes, bool] = dsclgsiz(handle)  
DscServerDiskLogBytes

[flags, filename, fileShareMode, openFlags, fileFlagAttributes, blockSize, maxCount, bool] = dsclgcg(handle)  
DscServerDiskLogConfigGet

<bool> = dsclgcs(handle, 'flags', 'filename', 
[['fileShareMode', 'openFlags', 'fileFlagsAttributes', 'blockSize', 'maxCount']])  
DscServerDiskLogConfigSet

r = dsclgqry(handle)  
DscServerDiskLogEnabledGet

<bool> = dsclgset(handle, setState)  
DscServerDiskLogEnabledSet

r = dsctccnt(dscHandle)  
DscTcEnabledCount

r = dscgdig(grpHandle)  
DscGroupConfigDialogShow

grpHandle = dscgopen  
DscGroupHandleOpen

<bool> = dscgclse(grpHandle)  
DscGroupHandleClose
```
index = dscgadd1(grpHandle <, unc_path>)

retVal = dscgcnt(grpHandle)  

<boolean> = dscgdell(grpHandle)

Handle = dscgdsc(grpHandle,
index)

<retVal> = dscgcfgr(grpHandle,
buffer)

[data,<retval>] =
dscgcfgw(grpHandle)

<boolean> = dscxbcal(handle <, sampleRate>)

ret = dscxbeg(handle)

<boolean> = dscxbes(handle,
isEnabled)

[iType, iRange, iOffset,
iOffsetRange, oExcitation,
<,boolean>] =
dscxbpcg(dscHandle, pinIndex)

<boolean> = dscxbpcs(dscHandle,
pinIndex <, iType, iRange,
iOffset, oExcitation>)

For more information, please refer to the corresponding function in the DSCIO Reference Manual.
Running the MATLAB iDSC Board Examples

Several example applications are included and demonstrate the use of an iDSC board with MATLAB. The examples can be run by typing app1, app2, app3 etc. at the MATLAB command prompt.

APP1.M
Application 1 reads a block of 1000 data values from the iDSC board. Input channels 0,1,2 etc can be selected using the iDSC dialog window. The values are merged into a MATLAB matrix. This column matrix has the size [1000,1]. This application shows how to initialize communication, get data, and terminate communication with the iDSC board.

APP2.M
Application 2 shows how to read an iDSC configuration block from a file to configure the parameters of an iDSC board. It also writes modified parameters back to the original file.

APP3.M
Application 3 shows the usage for many of the functions that modify the parameters of the iDSC board filters.

APP4.M
Application 4 shows the usage of the calibration function.

APP5.M
Before running this example, the module DAPLIIR.DLM should be installed. Use the DAP Service under the Control Panel; or use the module-related DLL MEX functions, such as DapModuleInstall, DapModuleUninstall, DapModuleLoad and DapModuleUnload, in DAPtools for MATLAB. (Please look at the DAPtools for MATLAB help for more information on module-related functions.) Application 5 gives an example of how to install a 32-bit module and configure DAPL commands for onboard processing.

APP6.M
Application 6 shows how to configure input timing channels.
**APP7.M**
Application 7 provides an example of getting matrix data with the transfer function and unit step function of an iDSC filter configuration.

**APP8.M**
Application 8 shows a simple example of using two iDSC boards in the master/slave configuration.

All the above examples open the handle to an iDSC board, get data from the iDSC board, and close the handle to the iDSC board. If you want to force a running example to stop, use **CTRL-C** to exit the example. When you do this, not all of the M file is executed and it is possible that the opened iDSC handle will not be closed. The opened handle should be closed explicitly by **dscclose**. If that does not work, restart MATLAB or the DAP service through the Control Panel.
**DLL MEX Reference**

Most of the returned arguments are message or error codes. These report whether or not the functions have been executed successfully.

Some functions have optional `TimeWait` and `TimeOut` parameters. The parameter `TimeWait` is the longest time in milliseconds that the operation can be blocked waiting for completion. The parameter `TimeOut` is the longest time in milliseconds that the operation can take for completion. If it is not completed in that amount of time, the operation is aborted. When this number is specified, it takes precedence over `TimeWait`. If values for these parameters are not given, the default is `TimeWait` of 100 ms and `TimeOut` of 20 seconds. These parameters are discussed in more detail in the DSCIO reference manual.

```matlab
hDsc = dscopen('uncPath')
```

This function opens a handle to the communication pipe defined by the UNC (Universal Naming Convention) path. This function should be the first iDSC function called in an application. Examples include:

```matlab
hDsc = dscopen('unc_path');
hDsc = dscopen('\\.\Dap0');
hDsc = dscopen('\\TestPC\Dap1');
```

The return value in `hDsc` is a handle to the iDSC board. The value of `hDsc` is 0 if an error occurred.

```matlab
<code> = dscclose(hDsc)
```

This function closes a handle specified by `hDsc` and terminates communication with an iDSC board. The return value `code` is 1 if the handle was closed successfully or 0 if an error occurred.

```matlab
code = dscavail(hDsc)
```

This function returns the number of bytes of data available for reading from an iDSC board specified by `hDsc`. For efficient data transfer, it is best to use `dscbufg` with a time wait and avoid using this function. The function `dscbufg` returns the actual number of bytes read which lets the application know what data have been transferred. The return value `code` is -1 if an error occurred.
[data, <code>] = dscbufg(hDsc, Len, <MaxLen, TimeWait, TimeOut, LenMult>)

This function reads a block of data from an iDSC board specified by hDsc, with two parameters, TimeWait and TimeOut, to control the transfer behavior. Parameters Len and MaxLen are the specified minimum and maximum number of bytes returned to data respectively. Both Len and MaxLen are always a multiple of LenMult. In most cases, data read from the iDSC is multiplexed: channel0, channel1, etc. It is possible to use standard MATLAB matrix operations to separate the channels. The return value code is the number of bytes read if the function succeeds, 0 if there is no data, or -1 if the function fails.

<code> = dsccal(hDsc)

This function performs calibration on an iDSC board specified by hDsc. It calibrates an iDSC board for DC gain and offset, and saves the calibration values. It is automatically invoked when dscstart is called for the first time. The saved calibration values are used until this function is invoked again. If the calibration values are not found, this function is automatically re-invoked to start calibration on the iDSC. The return value code is 1 if the function succeeds, or 0 if the function fails.

<code> = dsccmdld(hDsc)

This function downloads configuration commands from the host computer to an iDSC board specified by the handle hDsc and performs the necessary configuration for filtering. This function configures the iDSC board with the appropriate programs and coefficients.

The filter data are used by the iDSC internally to calculate and download the appropriate commands. If the iDSC board configuration or filter designs change and dscstart is executed, dscstart automatically invokes dsccmdld and downloads new commands to the iDSC board. Data will arrive at the PC two times dscgdela(hDsc) seconds later.

If the iDSC board configuration or filter designs have changed and this function is executed before dscstart, data will appear immediately on the host computers. These data are samples that were in the filtering buffers but are no longer needed there after the filter reconfiguration. The return value code is 1 if the function succeeds, or 0 if the function fails.
[<code>] = dsccfgrd(hDsc, buffer)
This function transfers iDSC board configuration information from the host computer to an iDSC specified by the handle hDsc. The iDSC board information, which is specified by buffer, includes sample rate, enabled input pins, input pin to filter mappings, and details of the filter designs. The return value code is the number of bytes being read from the host computer if the function succeeds, or 0 if the function fails.

[buffer, <code>] = dsccfgwt(hDsc)
This function returns information about sampling, filtering, and hardware channel assignments in the buffer array, for the board specified by the handle hDsc. The return value code is the number of bytes in the buffer if the function succeeds, or 0 if the function fails.

<code> = dscdialg(hDsc)
This function displays modal dialog screens for graphical filter configuration and design associated with an iDSC board specified by the handle hDsc. A filter configuration screen provides a quick method for selecting sample rates, mapping input pins to selected filter designs, and enabling or disabling input pins. If the function succeeds, and the OK button is selected, the return value code is 1. If the OK button is selected without changes, the return value code is 5. If the CANCEL button is selected, the return value code is 2. If the function fails, the return value is 0.

[filtIndex, filtType, shrp, coFL, coSL, coFH, coSH, atten, <code>] = dscfpget(hDsc, filtIndex)
This function gets filter parameters associated with a filter at index filtIndex of an iDSC board specified by the handle hDsc. Parameters filtType, shrp, coFL, coSL, coFH, coSH, and atten specify a filter type, sharpness, cutoff frequency low and high, and cutoff slope low and high parameters at the beginning of the transition band. If the filtIndex is not in the range of 0 through 7, the function fails. The return value code is 1 if the function succeeds, or 0 if the function fails.

<code> = dscfpset(hDsc, filtIndex, filtType, shrp, coFL, coSL, coFH, coSH, atten)
This function sets filter parameters associated with a filter at index filtIndex of an iDSC board specified by hDsc. Filter parameters can be set graphically in dscdialg. Parameters filtType, shrp, coFL, coSL, coFH, coSH, and atten
specify the filter type, sharpness, cutoff frequency low and high, and cutoff slope 
low and high parameters at the beginning of a transition band. Each iDSC board can 
store up to 8 filter designs, which can be accessed by the filtIndex parameter. If 
filtIndex is not in the range of 0 through 7, the function fails. The return value 
code is 1 if the function succeeds, or 0 if the function fails.

\[ \text{[gdelay, <code>] = dscgdela(hDsc)} \]
This function returns the group delay (in seconds) through all filter designs. The 
group delay is the amount of time to wait before data arrives at the PC.

The return value code is 1 if the function succeeds, or 0 if the function fails.

code = dscerrg
This function retrieves the text of the last error message that occurred in DSCIO 
DLL. This is useful in determining if there was an error related to any of the DSCIO 
DLL function calls. This function should be called immediately after a function of 
interest. If a function call fails, the last error message is updated. The last error 
message will stay the same until another function call fails. The return value code is 
the text of the last error message from the iDSC board if the function succeeds, or 0 
if this function either fails or returns no error message.

\[ \text{<code> = dscmaset(slavehDsc, masterhDsc)} \]
This function connects a Slave iDSC board specified by slavehDsc to a Master 
iDSC board specified by masterhDsc. It is only useful in a Master/Slave 
Configuration. The only way to specify an iDSC board as a master or slave is by 
invoking this function using two different iDSC board handles. If the same iDSC 
board handles are used, the Master and Slave iDSC boards will not be connected. 
When this function is completed, the iDSC board specified by slavehDsc becomes 
a Slave iDSC board and the iDSC board specified by masterhDsc becomes a 
Master iDSC board. The return value code is 1 if the function succeeds, or 0 if the 
function fails.

\[ \text{<code> = dscmemus(hDsc)} \]
This function determines the used memory on an iDSC board specified by the 
handle hDsc. It is useful in determining whether or not the iDSC board will be able 
to sustain a particular sample rate without overflowing. The return value code is the 
amount of memory used, expressed as a percent, if the function succeeds, or 0 if the 
function fails.
<code> = dscpecnt(hDsc)</code>
This function returns the number of enabled input pins on an iDSC board specified by the handle hDsc. Valid values for the enabled input pin count are in a range of one to eight since at least one input pin must be enabled. The return value code is the number of enabled input pins if the function succeeds, or 0 if the function fails.

<code> = dscpeget(hDsc)</code>
This function gets the enabled input pins on an iDSC board specified by the handle hDsc as a one-by-eight array of integer. When an input pin is enabled, the corresponding element in the array is set to one. When an input pin is disabled, the corresponding element in the array is set to zero. To find out the number of enabled input pins, use the dscpecnt function. The return value buffer is a one-by eight-integer array if the function succeeds, or 0 if the function fails since at least one input pin must be enabled.

<code> = dscpeset(hDsc, pin0, pin1, pin2, pin3, pin4, pin5, pin6, pin7)</code>
This function sets enabled input pins on an iDSC board specified by the handle hDsc. Set a pin variable to 1 to enable the corresponding pin, or to 0 to disable the pin. To find out the number of enabled input pins, use dscpecnt function. The return value code is 1 if the function succeeds, or 0 if the function fails.

The following example enables input pins S1, S3 and S6:
DscPinEnabledSet(hDsc, 0,1,0,1,0,0,1,0);

code = dscsrget(hDsc)
This function gets an effective sampling rate, in units of samples per second, for each channel on an iDSC board specified by the handle hDsc. The return value code is the sampling rate if the function succeeds, or 0 if the function fails.

<code> = dscsrset(hDsc, samRate)</code>
This function sets an effective sampling rate, in units of samples per second, for each channel to samRate on an iDSC board specified by the handle hDsc. Table 1 below displays valid sample rates.

<table>
<thead>
<tr>
<th>samRate</th>
<th>valid sample rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>153600</td>
<td>153600</td>
</tr>
<tr>
<td>102400</td>
<td>76800</td>
</tr>
<tr>
<td>51200</td>
<td>38400</td>
</tr>
<tr>
<td>25600</td>
<td>19200</td>
</tr>
</tbody>
</table>

Using the iDSC Board with MATLAB
If an invalid sample rate is selected, the function will automatically select the closest larger sample rate. The return value `code` is 1 if the function succeeds, or 0 if the function fails.

```markdown
<code> = dscstart(hDsc)
```

This function starts a data acquisition process on an iDSC board specified by the handle `hDsc`, causing data to start appearing on the host computer. This function internally forces a `dsccal` if it cannot find saved calibration values and a `dsccmdld` if the iDSC board configuration or filter designs have changed. If the iDSC board configuration or filter designs have changed, data will appear `dscg dela(hDsc)` seconds later.

Once this function is invoked, `dscbufg` can be called to read blocks of data into buffers. The return value `code` is 1 if the function succeeds, or 0 if the function fails.

```markdown
<code> = dscstop(hDsc)
```

This function stops a data acquisition process on an iDSC specified by the handle `hDsc`. It is called after `dscstart` to stop data from appearing on the host computer. The return value `code` is 1 if the function succeeds, or 0 if the function fails.

```markdown
<code> = dscteset(hDsc, Tc0, Tc1, TcWidth)
```

This function enables timing channels on an iDSC board specified by the handle `hDsc`. Parameters `Tc0` and `Tc1` are for Timing Channel 0 and 1 respectively. A value of one for one of a pin parameter enables a timing channel, or a value of zero disables it. Parameter `TcWidth` is for the width of the timing channel, which can be either 2 or 4 bytes depending on sampling rate. Setting `TcWidth` to one forces timing channel width to be 4 bytes (32 bits) regardless of the sample rate. The return value `code` is 1 if the function succeeds, or 0 if the function fails.
The following example is used to enable Timing Channel 1 with 4 bytes for the
timing channel width:

```matlab
DscTcEnabledSet(hDsc, 0, 1, 1);
```

The following example is used to enable Timing Channel 0 and Timing Channel 1
with 4 bytes for the timing channel width:

```matlab
DscTcEnabledSet(hDsc, 1, 1, 1);
```

<code> = dsctcmax(hDsc)
This function returns the maximum value a timing channel can contain based on the
sampling rate associated with an iDSC board specified by the handle hDsc. It ranges
from 128 to 2457600. The return value code is the maximum timing channel value
if the function succeeds, or 0 if the function fails.

[buffer, <code>] = dsctfget(hDsc, filtIndex, len)
This function gets the transfer function data points of a filter design used by an
iDSC board specified by the handle hDsc. Parameter filtIndex ranges from 0 to 7
since there are up to eight filter designs. The parameter len specifies number of
data points in the returned transfer function array buffer. The return value code
is 1 if the function succeeds, or 0 if the function fails.

<code> = dsctcwdt(hDsc)
This function returns the timing channel width in bytes associated with an iDSC
board specified by the handle hDsc, either 2 bytes for 16-bit values or 4 bytes for
32-bit values. Width is dependent on the sample rate. The timing channel width for
sample rates 8 s/s to 600 s/s is always 4 bytes. The timing channel width for sample
rates 640 s/s to 153600 s/s is 2 bytes by default. The timing channel width for these
higher sample rates can be forced to 4 bytes by setting Tcwidth in the dscteset
function. The return value code is the timing channel width if the function succeeds,
or 0 if the function fails.

[buffer, <code>] = dscusget(hDsc, filtIndex, <len>)
This function gets the unit step response data points of the filter design used by an
iDSC board specified by the handle hDsc. Parameter filtIndex ranges from 0 to 7
since there are up to eight filter designs. Optional parameter len is the number of
data points that should be returned in buffer. The return value code is 1 if the
function succeeds, or 0 if the function fails.
<code> = dscdtset(hDsc, 'daplTxt')</code>
Defines DAPL commands in text format for an iDSC board specified by the handle 
hDsc. The DAPL commands are defined by parameter daplTxt, which will be sent 
to the iDSC when dsccmddld or dscstart is invoked. Each line of the DAPL 
commands must be delimited by a carriage-return, line-feed, or both. The return 
value code is 1 if the function succeeds, or 0 if the function fails.

<code> = dscdccls(hDsc, 'custCmdTxt', stacksize)
Defines a list of DAPL custom commands and sends the commands to an iDSC 
board specified by the handle hDsc when dsccmddld or dscstart is invoked. A 
custom command list is defined by parameter custCmdTxt. Each custom command 
in the list must be delimited by a carriage-return by putting integer 13. Parameter 
stacksize is a matrix that defines the stack size for each command in the list. The 
return value code is 1 if the function succeeds, or 0 if the function fails.

This function is used with obsolete 16-bit custom commands only. For downloading 
a 32-bit custom command module, please use Control Panel->Data Acquisition 
Processor, or dapmdin in DAPtools for MATLAB.

[unc_path, <code>] = dscaddrh(hDsc)
This function gets a UNC path associated with an iDSC board specified by the 
handle hDsc. The return argument unc_path is a UNC string consisting of a 
machine name and an iDSC board name. The return value code is the length of 
unc_path if the function succeeds, or -1 if the function fails.

<code> = dscaddrs(hDsc, 'uncPath')
This function changes the address, specified by uncPath, to an iDSC board 
specified by hDsc. The return value code is 1 if the function succeeds, or 0 if the 
function fails.

[numData, <code>] = dscdscan(hDsc)
This function returns the number of scans thrown away since dsccmddld. Each scan 
consists of one sample for each of the active input pins. For example, if there are 
two enabled input pins, a scan consists of two samples. The scan value is used to 
synchronizing multiple iDSC boards. Since dscstart is a software command, the 
iDSC boards receive it at different times, which means the number of scans per 
iDSC board is different when the START command is received. Since iDSC boards 
must be synchronized, the same number of scans per iDSC board between 
dscmddld and dscstart have to be ignored. The returned scan will be stored in
Using the iDSC Board with MATLAB

numData. The return value code is 1 if the function succeeds, or 0 if the function fails.

code = dscmaget(slavehDsc)
This function gets the handle to a master iDSC board when invoked on a slave iDSC board specified by slavehDsc. It is only useful in a Master/Slave Configuration. This function fails when invoked on a master or normal iDSC handle. The return value code is a handle to a master iDSC board if the function succeeds, or 0 if the function fails.

[numBytes <, code>] = dsclgsiz(hDsc)
This function gets the number of bytes logged to a disk server from an iDSC board specified by the handle hDsc. It should be called after the function dscstart is invoked, and when dsclgqry returns ‘1’. The return value code is 1 if the function succeeds, or 0 if the function fails.

[flags, filename, fileShareMode, openFlags, fileFlagAttributes, blockSize, maxCount <, code>] = dsclgcg(hDsc)
This function gets a server disk log configuration associated with the iDSC board specified by the handle hDsc. Below is a brief description of the returned parameters:

  flags – logging behavior
  filename – name of disk log file
  fileShareMode – file share mode of the disk log file
  openFlags – file open options of the disk log file
  fileFlagAttributes – file attributes of the disk log file
  blockSize – minimum amount of data, in bytes, to write to the disk log file at one time
  maxCount – maximum number of bytes to log. The default is 0, which causes logging to continue indefinitely until dscstop is invoked

The return value code is 1 if the function succeeds, or 0 if the function fails. For more information, please look at TServerDiskLogConfig in the DSCIO reference manual.
Using the iDSC Board with MATLAB

<code> = dsclgcs(hDsc, 'flags', 'filename', [fileShareMode', 'openFlags', 'fileFlagsAttributes', blockSize, maxCount])

This function sets a server disk log configuration associated with the iDSC board specified by hDsc. Below is a brief descriptions of return parameters:

- **flags** – logging behavior
- **filename** – name of disk log file
- **fileShareMode** – file share mode of the disk log file
- **openFlags** – file open options of the disk log file
- **fileFlagsAttributes** – file attributes of the disk log file
- **blocksize** – minimum amount of data, in bytes, to write to the disk log file at one time
- **maxCount** – maximum number of bytes to log. The default is 0, which causes logging to continue indefinitely until **dscstop** is invoked.

The return value **code** is 1 if the function succeeds, or 0 if the function fails. For more information, please look at TServerDiskLogConfig in the DSCIO reference manual.

**code = dsclgqry(hDsc)**

This function queries the specified server, associated with an iDSC board specified by the handle hDsc, to determine whether disk logging is enabled. The return value **code** is 1 if the function succeeds, or 0 if the function fails.

<code> = dsclgset(hDsc, setState)

This function changes the state of the specified server, associated with an iDSC board specified by the handle hDsc. Setting the parameter **setState** to one or zero enables and disables server disk logging, respectively. The return value **code** is 1 if the function succeeds, or 0 if the function fails.

**code = dsctccnt(hDsc)**

This function returns the number of enabled timing channels associated with an iDSC board specified by hDsc. The return value **code** is the number of enabled timing channels if the function succeeds, or -1 if the function fails.

**code = dscgdlg(grpHDsc)**

This function displays modal dialog screens for the graphical configuration of multiple iDSC boards. The modal dialog screen from **dscdialog** is for one iDSC only. This function should be used to configure more than one iDSC board. It simplifies the configuration of multiple iDSC boards with a graphical interface. The
return value code is 1 if the OK button has been clicked, 2 if the cancel button has been clicked, and 0 if the function fails.

**code = dscgopen**
This function opens a handle to an iDSC group. Similar to dscopen, this function is the first function called by an application ready to begin communication with the iDSC group. Once the iDSC group communication has begun, a specific iDSC board can be accessed through dscgdsc. The return value code is a handle to a group of iDSC boards if the function succeeds, or 0 if the function fails.

<boolean> = dscgcclse(grpHDsc)
This function terminates communication and releases an iDSC group handle specified by grpHDsc previously opened with dscopen. It should be the last function called by an application to end communication with the iDSC group. Once the communication has ended, an iDSC board cannot be accessed through dscgdsc. The return value code is 1 if the function succeeds, or 0 if the function fails.

**code = dscgadd1(grpHDsc <, 'unc_path'>)**
This function adds one iDSC to the tail of the iDSC group specified by grpHDsc. It provides an option to set the address of an iDSC. The return value code is an index of the newly added iDSC board if the function succeeds, or -1 if the function fails. The maximum capacity of an iDSC group handle is 64, which means the maximum index is 63.

**code = dscgcnt(grpHDsc)**
This function returns the number of iDSC boards in an iDSC group specified by grpHDsc. The count is increased by one when dscgadd1 is invoked, and decreased by one when dscgdel1 is invoked. The return value code is the number of iDSC boards in the group if the function succeeds, or -1 if the function fails.

<code> = dscgdel1(grpHDsc)**
This function deletes one iDSC board from the tail of the iDSC group specified by grpHDsc. The iDSC board with highest index always is deleted first. The return value code is 1 if the function succeeds, or 0 if the function fails.
code = dscgdsc(grpHDsc, iDscIndex)
This function accesses an iDSC board at index iDscIndex of an iDSC group
specified by a group handle grpHDsc. Valid iDSC board indices are 0 through
dscgcnt – 1. The individual iDSC board handle should not be stored because it is
destroyed and recreated in functions like dscgdlg, dscgdel1, dscgadd1, etc. It
should be called whenever accessing a specific iDSC board. The return value code
is a handle to the target iDSC board if the function succeeds, or 0 if the function
fails.

<code> = dscgcfgr(grpHDsc, buffer)
This function transfers configuration information in binary format from a host
computer to an iDSC group specified by the group handle grpHDsc. iDSC group
information includes the number of iDSC boards, address, mode, sample rate, input
range, input pin to filter mappings, enabled input pin information, and details of
filter designs. The return value code is the number of bytes read if the function
succeeds, or 0 if the function fails.

[buffer <, code>] = dscgcfw(grpHDsc)
This function transfers configuration information from an iDSC group specified by
the group handle grpHDsc to the host computer. iDSC group information written
includes the number of iDSC boards, address, mode, sample rate, input range, input
pin to filter mappings, enabled input pin information, and details of the filter
designs. The return value code is the number of bytes written if the function
succeeds, or 0 if the function fails.

<code> = dscxbcal(hDsc <, sampleRate>)
This function performs calibration on an expansion board, part number MSXB042,
associated with an iDSC board specified by the handle hDsc. This function works
only if the board’s enable flag is on, which can be queried by calling dscxbeg. If
dscxbeg returns false, a failure has occurred. The sample rate should not exceed
1024 samples per second or the calibration values may be unstable. A higher sample
rate allows this function to execute faster but the calibration values are less accurate.
A lower sample rate executes more slowly but the calibration values are more
accurate. The default and recommended value of 100 samples per second works
well. The return value code is 1 if the function succeeds, or 0 if the function fails.
<code> = dscxbeg(hDsc)</code>
This function gets the state of an MSXB042 expansion board associated with an iDSC board specified by the handle hDsc. The return value code is 1 for enabled and 0 for disabled if function succeeds, or -1 if the function fails.

<code> = dscxbes(hDsc, isEnabled)</code>
This function sets the state of an MSXB042 expansion board associated with an iDSC board specified by the handle hDsc. Setting isEnabled to 1 enables the board and makes it visible in dscdialg, whereas setting isEnabled to 0 disables the board and makes it invisible in dscdialg. The return value code is 1 if function succeeds, or 0 if the function fails.

<iType, iRange, iOffset, iOffsetRange, oExcitation <,code> =
 dscxbpcg(hDsc, pIndex)</i>
This function gets a pin configuration at index pIndex of an MSXB042 expansion board associated with an iDSC board specified by the handle hDsc. Parameters iType, iRange, iOffset, iOffsetRange, and oExcitation specify an input type, input range, input offset, input offset range, and an output excitation. The return value code is 1 if the function succeeds, or 0 if the function fails. For more information, please see DscXbPinConfigGet and TXbPinConfig in the DSCIO Reference Manual.

<code> = dscxbpcs(hDsc, pIndex <, iType, iRange, iOffset, oExcitation>)</code>
This function sets a pin configuration at index pIndex of an MSXB042 expansion board associated with an iDSC board specified by the handle hDsc. Parameters iType, iRange, iOffset, iOffsetRange, and oExcitation specify an input type, input range, input offset, input offset range, and an output excitation. The return value code is 1 if the function succeeds, or 0 if the function fails. For more information, please see DscXbPinConfigSet and TXbPinConfig in the DSCIO Reference Manual.
Section III. Programming Interfaces
9. DSCIO DLL Programmer’s Interface

The Digital Signal Conditioning Input Output (DSCIO) Dynamic Link Library Programmer’s Interface provides the link between an application and the iDSC board.


An application opens a handle to the iDSC board and then uses the handle to configure the system and receive data. An opened handle is reserved for access by the application exclusively until the application closes the handle. A handle is a 32-bit value that references the iDSC board.

To open a handle to the iDSC board, use the function `DscHandleOpen`. `DscHandleOpen` requires a single parameter, `pszAddress`, which is the name of the iDSC board expressed using the Universal Naming Convention (The Universal Naming Convention is discussed later in this chapter).

To close a handle to the iDSC board, use the function `DscHandleClose`. `DscHandleClose` requires a single parameter, `hDsc`, the handle previously opened with `DscHandleOpen`.

The DSCIO Function Summary at the end of this chapter provides a summary of all of the supported functions.
DSCIO Interface Examples

There are several examples located in the `<InstallDir>\Examples\VB` directory for Visual Basic and the `<InstallDir>\Examples\C` directory for C/C++ which demonstrate the use of the DSCIO interface. Before running the examples, verify that the iDSC board and iDSC board software are properly installed by running DSCview. Exit DSCview before running the DSCIO interface examples.

Visual Basic Examples

Dvm.vbp

Dvm.vbp shows how to display voltage measurements. It transfers blocks of data from the iDSC board into a data buffer. It displays the 1st index from the data buffer, assuming a 5V input signal.

BinLog.vbp

BinLog.vbp shows how to log binary data to disk. It allows the user to design filters and configure the iDSC board using the `DscConfigDialogShow` function. When the iDSC board is started, the selected disk log file is opened and data logging starts. When the iDSC board is stopped, the selected disk log file is closed and data logging stops.

LoadSave.vbp

LoadSave.vbp shows how to load and save the iDSC board configuration from and to a file. It displays a previously saved configuration using the `DscConfigDialogShow` function.

C/C++ Console Examples

BinLog.cpp

BinLog.cpp shows how to log binary data to disk. It allows the user to design filters and configure the iDSC board using the `DscConfigDialogShow` function.
TxtLog.cpp

TxtLog.cpp shows how to log text data to disk. It allows the user to design filters and configure the iDSC board using the DscConfigDialogShow function.

LoadSave.cpp

LoadSave.cpp shows how to load and save the iDSC board configuration from and to a file. It displays a previously saved configuration using the DscConfigDialogShow function.
Creating a DSCIO Interface Application

Before you create a DSCIO Interface application, make sure that you have installed the Accel32/DAPcell server. The DAPIO32.DLL that is shipped with the Accel32/DAPcell server is needed for any application that uses the DSCIO Interface.

The following steps describe the functions required for creating a simple application with the DSCIO interface.

1) Open a handle to the iDSC board using `DscHandleOpen`.

2) Configure the iDSC board by invoking the `DscConfigDialogShow` function. This allows the user to set the sample rate, enable/disable channels, design filters, etc.

3) Start data acquisition by invoking the `DscStartAcquiring` function.

4) Get binary data from the iDSC board by invoking the `DscBufferGetEx` function.

5) Stop data acquisition by invoking the `DscStopAcquiring` function.

6) Close the handle to the iDSC board using `DscHandleClose`.

When programming in C/C++, include the header file `Dscio.h` that is located in `<InstallDir>\Dll\Import\C`. If you are using a Microsoft C/C++ compiler, you must link with the `Dscio.lib` located in `<InstallDir>\Dll\Lib\MC`. If you are using a Borland C/C++ compiler, you must link with the `Dscio.lib` located in `<InstallDir>\Dll\Lib\BC`.

When programming in Visual Basic, include the header file `Dscio.bas` that is located in `<InstallDir>\Dll\Import\VB`.

When programming in Pascal, include the header file `Dscio.pas` that is located in `<InstallDir>\Dll\Import\Pascal`.

More complicated applications that use DAPL text and DAPL custom commands are also available with the DSCIO interface.
Universal Naming Convention

DSCIO addresses the iDSC board using the Universal Naming Convention (UNC). A UNC name consists of two parts, the machine name and the iDSC board name. A UNC name begins with two backslashes, and the parts of the name are separated by a single backslash as shown below.

```
\<Machine name>\<iDSC board name>
```

A local machine is denoted by a period. A remote machine is represented by its unique network machine name. Only the DAPcell implementation of the iDSC board supports remote machine names. All other implementations support only local machine names.

iDSC board names are predefined as Dap0, Dap1, ..., Dap(N-1) where N is the number of iDSC boards installed on the system. The maximum value for N is 14.

The UNC name is used with the DSCIO interface functions, DscAddressGet and DscAddressSet. For \.\Dap0, \. denotes the name of the local machine and \Dap0 denotes the name of the iDSC board. If connected through DAPcell to a remote machine named PC45 that contains Dap0, then pszAddress is \PC45\Dap0.
**Master/Slave Configuration**

To synchronize multiple iDSC boards on a system, a user must designate one iDSC board as the master unit and the other iDSC boards as slave units. The configuration for master and slave must be done in hardware by using a special cable, and in software by calling the `DscMasterSet` function.

Synchronous iDSC boards must share the same sampling clock; therefore, a special cable is necessary to distribute the sampling clock. The Synchronization Connector used in Master/Slave Configurations is described in the Hardware Architecture chapters of this document.

Apart from connecting the cable, one iDSC board must be configured as the Master iDSC board by using the `DscMasterSet` function. The `DscMasterSet` function associates a slave iDSC board to a master iDSC board.

The following functions support Master/Slave Configurations.

- `DscMasterGet`
- `DscMasterSet`
- `DscOperateModeGet`
- `DscOperateModeSet`
- `DscRemoteMasterGet`
- `DscRemoteMasterSet`
- `DscSlaveCount`
- `DscSlaveHandle`

The master unit and slave units must use the same effective sample rate. Therefore, the sample rate should only be changed on the Master iDSC board using `DscSampleRateSet`. Changing the sample rate on the Slave iDSC board will have no effect. The Slave iDSC board will continue using the sample rate of its master.

If two independent iDSC boards are running at different sample rates, and the user decides to set up the iDSC boards in a Master/Slave configuration, the Slave iDSC board will use the sample rate of the Master iDSC board.

In a Master/Slave Configuration, the system services listed below should only be invoked on the Master iDSC board. If these functions are invoked on the Slave iDSC board, an error will occur and the functions will return zero.

- `DscCalibrate`
- `DscCommandsLoad`
- `DscStartAcquiring`
The memory streaming functions `DscConfigWrite`, `DscConfigWriteSize`, and `DscConfigRead` manage iDSC board information to and from memory. The iDSC board information includes the sample rate, enabled input pins, input pin to filter mappings, and the details of all of the filter designs. However, the iDSC board information does not include Master/Slave Configuration information related to `DscMasterSet`. Because the functions do not store Master/Slave Configuration information, a user must store this information explicitly.

`DscGroupConfigDialogShow` allows easy graphical configuration of masters and slaves.
DAPL Support

Writing DAPL

The DSC system writes the filtered data to pipe pDscData for use in DAPL. Pipe pDscData has interleaved data only from the enabled channels. Before the user can perform further processing on the data, the SEPARATE command is recommended for separating the data from pipe pDscData into the correct number of pipes. If the user only has five channels enabled out of the maximum of eight, the user should separate the interleaved data into five pipes. Once the user has completed processing the data, the results must be merged to $BINOUT.

The restricted DAPL interface requires that input and output procedures, and START and STOP commands, not be included in the DAPL listing. The iDSC can only be started and stopped using the DscStartAcquiring and DscStopAcquiring functions.

The example below shows the form of the first few lines in DAPL. Pipe pDscData has interleaved data from five enabled channels.

```
    PPIPES P0, P1, P2, P3, P4
    PDEF A
    SEPARATE(pDscData, P0, P1, P2, P3, P4)
    ...
    ...(further processing)
    ...
    ...(must write output to $BINOUT)
    END
```

P0, P1, P2, P3, P4 are the number of enabled channels, which is five in this example.

Using the DAPL Interface

To write custom DAPL text, the user should use DscDaDaplTextSet. Once the DAPL text has been defined for the iDSC using DscDaDaplTextSet, the DAPL text will be sent to the iDSC when the user invokes DscCommandsLoad or DscStopAcquiring.
If the user wants to change the already defined DAPL text, `DscDaplTextSet` must be reinvoked with the new DAPL text followed by either a `DscCommandsLoad` or `DscStartAcquiring` to activate the new DAPL text.

If the user wants to retrieve the defined DAPL text, `DscDaplTextLengthGet` along with `DscDaplTextGet` should be used.

Below is a C/C++ example:

```c
HDSC hDsc;
hDsc = DscHandleOpen("\\\.\Dap0"); // Open iDSC handle.
DscDaplTextSet(hDsc, "PIPES P0, P1, P2, P3, P4\r\n" "PIPES R0, R1, R2, R3, R4\r\n" "PDEF A\r\n" "SEPARATE(pDscData, P0, P1, P2, P3, P4)\r\n" "FFT(5, 9, 4, P0, R0)\r\n" "FFT(5, 9, 4, P1, R1)\r\n" "FFT(5, 9, 4, P2, R2)\r\n" "FFT(5, 9, 4, P3, R3)\r\n" "FFT(5, 9, 4, P4, R4)\r\n" "MERGE(R0, R1, R2, R3, R4, $BINOUT)\r\n" "END\r\n");
DscStartAcquiring(hDsc); // Send DAPL text.
//... (other function calls) ...
```
Structure Summary

Structures are used to pass information to and from many DSCIO functions. Applications must initialize all fields before passing structures to DSCIO functions.

To use the supported structures, the memories of the structures must first be fully initialized to zero. This memory initialization is necessary because different versions of the DSCIO interface may have different size structures. Therefore it is important to use `DscStructPrepare` for proper initialization.

The following structures are discussed in detail on the following pages.

- `TBufferGetEx`
- `TDscIoInt64`
- `TFilterParam`
- `TProcSystemErrorStdcall`
- `TServerDiskLogConfig`
- `TXbPinConfig`
**TBufferGetEx**

The `TBufferGetEx` structure defines the behavior of `DscBufferGetEx`.

```c
typedef struct tag_TBufferGetEx {
    int iInfoSize; // Size of this structure.
    int iMinBytes; // Minimum number of bytes to get.
    int iMaxBytes; // Maximum number of bytes to get.
    int iT imeWait; // Longest time to wait for data.
    int iTimeOut; // Longest total time for operation.
    int iBytesMultiple; // Bytes to get is a multiple of this.
} TBufferGetEx;
```

**Members**
- `iInfoSize`
  - Size of this information structure.
- `iMinBytes`
  - Minimum number of bytes to get. It can be zero or any positive integer.
- `iMaxBytes`
  - Maximum number of bytes to get. `iMaxBytes` must be greater than or equal to `iMinBytes`.
- `iT imeWait`
  - Longest time in milliseconds that the get operation can be blocked waiting for data. If no data shows up in that amount of time, the operation should be aborted.
- `iT imeOut`
  - Longest time in milliseconds that the get operation should complete. If it fails to complete in that amount of time, the operation is aborted. When this member is specified, it takes precedence over `iT imeWait`. This member is ignored if its value is zero.
- `iBytesMultiple`
  - The number of bytes to get is always a multiple of `iBytesMultiple`.

**Remarks**
- `TBufferGetEx` is used in `DscBufferGetEx`. `TBufferGetEx` should be initialized using `DscStructPrepare` or `DscBufferGetEx` will fail.
Each member of TBufferGetEx must be initialized to the appropriate value before being passed to DscBufferGetEx. The member iMinBytes must be greater than or equal to zero, and the member iMaxBytes must be greater than or equal to iMinBytes. If iMinBytes is zero DscBufferGetEx will never block even when there are no data available.

A zero value of iBytesMultiple is treated the same as one. The value of iBytesMultiple cannot be larger than the maximum pipe buffer size on the PC side (converted to bytes) minus 1024, or minus the iDSC side blocking size (converted to bytes), whichever is larger; otherwise, the first condition causes an error. The second condition is not checked and may cause a deadlock. It is the application's responsibility to guarantee that it never happens.

Both iMinBytes and iMaxBytes must be an integral multiple of the iBytesMultiple value; otherwise, an error occurs.

See Also
DscBufferGetEx, DscStructPrepare
The `TDscIoInt64` structure defines the behavior of the `DscServerDiskLogBytes` function and the `i64MaxCount` parameter of `TServerDiskLogConfig`. `TDscIoInt64` represents a 64-bit integer type.

```c
#ifdef M_DscIoNoInt64
typedef struct tag_TDscIoInt64 {
    unsigned long dwLowPart; // Low 32-bits of 64-bit integer type.
    unsigned long dwHighPart; // High 32-bits of 64-bit integer type.
} TDscIoInt64;
#else
typedef __int64 TDscIoInt64;
#endif
```

**Example**

The following examples show how to initialize the `i64MaxCount` parameter of `TServerDiskLogConfig` to a value of 4294967296. The first example is for environments that support the `__int64` data type while the second example is for environments that do not support the `__int64` data type.

```c
// Example 1: __int64 type is supported
#include <dscio.h>
...
TServerDiskLogConfig sdlc;
DscStructPrepare(&sdlc, sizeof(sdlc));
sdlc.i64MaxCount = 4294967296;

// Example 2: __int64 type is not supported
#define M_DscIoNoInt64 1
#include <dscio.h>
...
TServerDiskLogConfig sdlc;
DscStructPrepare(&sdlc, sizeof(sdlc));
```
sdlc.i64MaxCount.dwLowPart = 0;
sdlc.i64MaxCount.dwHighPart = 1;

See Also

TServerDiskLogConfig.DscServerDiskLogBytes
The **TFilterParam** structure defines the behavior of **DscFilterParametersGet** and **DscFilterParametersSet**. The defaults for the members depend on the sample rate.

```c
typedef struct tag_TFilterParam {
    int iInfoSize; // Size of this structure.
    char achName[64]; // Filter name.
    int iFilterType; // Filter type.
    int iSharpness; // Filter sharpness.
    float fCutoffFreqLow; // Filter low cutoff frequency.
    float fCutoffSlopeLow; // Filter low cutoff slope.
    float fCutoffFreqHigh; // Filter high cutoff frequency.
    float fCutoffSlopeHigh; // Filter high cutoff slope.
    float fAttenuation; // Filter attenuation.
} TFilterParam;
```

**Members**

- **iInfoSize**
  Size of this information structure.

- **achName**
  Name of the filter. The name is restricted to 63 characters plus one null-terminated character. The default is FD0, FD1, FD2, FD3, FD4, FD5, FD6, and FD7 for each of the eight filters.

- **iFilterType**
  Type of the filter, either lowpass or bandpass. These constants simplify selecting the filter type: **Dsc_LowPass**, **Dsc_BandPass**. The default is lowpass.

- **iSharpness**
  Sharpness of the filter specified as an odd number. Valid numbers are in the range 37 to 255, depending on the sample rate.

  The default is 37 for sample rate 153600 s/s, 119 for sample rate 102400 s/s, 95 for sample rate 76800 s/s, 195 for sample rates of 51200 s/s, 1024 s/s, 2048 s/s, and 1024 s/s and 137 for all other sample rates.

- **fCutoffFreqLow**
  Low cutoff frequency of the filter specified in Hertz. Valid numbers are in the range 2% to 80% of the Nyquist frequency (0.02 * Nyquist frequency to 0.8 * Nyquist frequency) for each filter.
Nyquist frequency). The Nyquist frequency is half the sample rate. The default is half the Nyquist frequency.

\textit{fCutoffSlopeLow}

Low cutoff slope of the filter specified as a fraction of the Nyquist frequency, with the Nyquist frequency normalized to 1.0. Valid numbers are in the range 0.0 to 0.8. 0.0 corresponds to 0% of the Nyquist frequency and 0.8 corresponds to 80% of the Nyquist frequency. The Nyquist frequency is half the sample rate.

The default is 0.0 for sample rates of 153600 s/s, 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, and 1024 s/s and 0.04 for all other sample rates.

\textit{fCutoffFreqHigh}

High cutoff frequency of the filter specified in Hertz. Valid numbers are in the range 2% to 80% of the Nyquist frequency (0.02 * Nyquist frequency to 0.8 * Nyquist frequency). The Nyquist frequency is half the sample rate. The default is three-quarters the Nyquist frequency.

\textit{fCutoffSlopeHigh}

High cutoff slope of the filter specified as a fraction of the Nyquist frequency, with the Nyquist frequency normalized to 1.0. Valid numbers are in the range 0.0 to 0.8. 0.0 corresponds to 0% of the Nyquist frequency and 0.8 corresponds to 80% of the Nyquist frequency. The Nyquist frequency is half the sample rate.

The default is 0.0 for sample rates of 153600 s/s, 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, and 1024 s/s and 0.04 for all other sample rates.

\textit{fAttenuation}

Attenuation of the filter in the stopband region. Valid numbers are in the range 6.0 to 12.0. The larger the number, the more attenuation in the stopband, but this will further attenuate the frequencies near cutoff.

The default is 10.2 for sample rate 153600 s/s, 9.4 for sample rate 76800 s/s, 9.8 for sample rates of 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, and 1024 s/s and 9.0 for all other sample rates.

Remarks

\textit{TFilterParam} is used in \textit{DscFilterParametersGet} and \textit{DscFilterParametersSet}. \textit{TFilterParam} should be initialized using \textit{DscStructPrepare} or both functions will fail.

\textit{DscFilterParametersGet} will return the filter parameters in the \textit{achName}, \textit{iFilterType}, \textit{iSharpness}, \textit{fCutoffFreqLow}, \textit{fCutoffSlopeLow}, \textit{fCutoffFreqHigh}, \textit{fCutoffSlopeHigh}, and \textit{fAttenuation} members.
Before invoking `DscFilterParametersSet`, the `achName`, `ifFilterType`, `iSharpness`, `fCutoffFreqLow`, `fCutoffSlopeLow`, `fCutoffFreqHigh`, `fCutoffSlopeHigh`, and `fAttenuation` members must be set to the new desired values.

**See Also**

`DscFilterParametersGet`, `DscFilterParametersSet`, `DscStructPrepare`
**TProcSystemErrorStdcall**

The `TProcSystemErrorStdcall` structure defines the behavior of the `DscOnSystemErrorSet` function.

```c
typedef void_stdcall TProcSystemErrorStdcall(
    void *pInfo, // Pointer to additional information.
    HDSC hDsc, // iDSC board handle.
    const char *sError // Error from the iDSC board.
);
```

**Members**

- `pInfo`  
  A pointer to additional information to satisfy type calling conventions.

- `hDsc`  
  Handle of the iDSC board.

- `sError`  
  The error returned from the iDSC board.

**Remarks**

`TProcSystemErrorStdcall` supports `DscOnSystemErrorSet`. The parameter, `sError`, displays an error message when a system error occurs. The user can use the `sError` parameter to display the error message in an error handling routine.

The `DscOnSystemErrorSet` function runs when the `DscSystemErrorProcess` function is invoked. `DscSystemErrorProcess` is automatically invoked during `DscBufferGetEx` if there are errors from the iDSC. The user can also call `DscSystemErrorProcess` and invoke `DscOnSystemErrorSet` if errors from the iDSC are suspected.

**See Also**

`DscOnSystemErrorSet`, `DscSystemErrorProcess`
**TServerDiskLogConfig**

The `TServerDiskLogConfig` structure defines the behavior of `DscServerDiskLogConfigGet` and `DscServerDiskLogConfigSet`.

```c
typedef struct tag_TServerDiskLogConfig {
    int iInfoSize;                // Size of this structure.
    unsigned long dwFlags;        // Logging behavior flags.
    char *pszFileName;            // File name.
    unsigned long dwFileNameSize; // Size of file name.
    unsigned long dwFileShareMode; // File share properties.
    unsigned long dwOpenFlags;    // File open options.
    unsigned long dwFileFlagsAttributes; // File attributes.
    unsigned long dwBlockSize;    // Size of block to write.
    TDscIoInt64 i64MaxCount;      // File maximum count.
} TServerDiskLogConfig;
```

**Members**

- **iInfoSize**
  Size of this information structure.

- **dwFlags**
  Flags to control disk logging behavior. The constants below simplify selecting `dwFlags`. The default is `DscDlfServerSide` and `DscDlfFlushBefore`.

  - **DscDlfServerSide**
    Log on the same side of the network connection as the iDSC. If not specified, logging will take place on the application (client) side of the network connection.

  - **DscDlfFlushBefore**
    Flush the input data pipe before beginning the logging session. Default action is to not flush the pipes before logging.

  - **DscDlfFlushAfter**
    Flush the input pipe after the logging session has terminated. Default action is to not flush the pipes after logging.

  - **DscDlfMirrorLog**
    Enable mirror logging. Mirror logging creates a copy of the logged data in another file.
DscDlfAppendData
Allow new data to be appended to an existing file. Only the DscOfOpenAlways and DscOfOpenExisting flags of the dwOpenFlags member can be used for appending.

DscDlfBlockTransfer
Open the file with no intermediate buffering or caching and access the file in a special way that is highly dependent on the target disk attributes to improve performance. This transfer mode adds overhead to slow rate transfer with small buffers. It should only be used when necessary with a very large dwBlockSize value (such as 1048576 and above). If this option is selected, dwBlockSize is automatically set to 1048576.

pszFileName
Name of the primary disk log file and name of a possible mirror disk log file, if mirror logging is enabled. The default is an empty string. The size of the user allocated buffer must be specified using dwFileNameSize when used with DscServerDiskLogConfigGet.

Mirror logging is enabled by selecting the DscDlfMirrorLog flag of the dwFlags member. Multiple file names are separated by semi-colons. Currently, only one mirror file is allowed. Both files must be on the same side of the DAPcell Local/DAPcell service (the PC application side or the iDSC side).

dwFileNameSize
Size of the user allocated buffer to store the file name specified by pszFileName. The size must include an extra space for the null terminator. This field is important in DscServerDiskLogConfigGet. It is not used in DscServerDiskLogConfigSet.

If dwFileNameSize is 0, the file name is not returned in pszFileName. If dwFileNameSize is 1, only the null terminator is returned in pszFileName.

dwFileShareMode
File share mode of the disk log file. The constants below simplify selecting dwFileShareMode. The default is DscFsmRead.

DscFsmNone The file cannot be used by another process.
DscFsmRead The file can be read by another process.
DscFsmWrite The file can be written to by another process.
DscFsmReadWrite The file can be read and written to by another process.

dwOpenFlags
File open options of the disk log file. The constants below simplify selecting dwOpenFlags. The default is DscOfCreateAlways.
DscOfCreateNew
Create a new file. Creation fails if the file already exists.

DscOfCreateAlways
Create a new file. If the file already exists, it is overwritten.

DscOfOpenAlways
Open an existing file. If the file does not exist, it will be created.

DscOfOpenExisting
Open an existing file without resetting permissions. Opening fails if the file does not exist.

dwFileFlagsAttributes
File attributes of the disk log file. The constants below simplify selecting dwFileFlagsAttributes. The default is DscFfaAttributeNormal.

DscFfaAttributeNormal
No special attributes.

DscFfaAttributeEncrypted
The data in the file is encrypted.

DscFfaFlagWriteThrough
Write through any intermediate caching and go directly to disk.

DscFfaFlagSequentialScan
Can be used to optimize the transfer of large blocks of data. Most applications will not need this flag.

dwBlockSize
Minimum amount of data, in bytes, to write to the disk log file at one time. This field is provided for disk transfer optimization. The default is 8192.

i64MaxCount
Maximum number of bytes to log. The default is 0, which causes logging to continue indefinitely until DscStopAcquiring is invoked. It is a TDscIoInt64 structure.

Remarks
TServerDiskLogConfig is used in DscServerDiskLogConfigGet and DscServerDiskLogConfigSet. TServerDiskLogConfig should be initialized using DscStructPrepare or both functions will fail.

DscServerDiskLogConfigGet will return the server disk log configuration in the dwFlags, pszFileName, dwFileShareMode, dwOpenFlags, dwFileFlagsAttributes, dwBlockSize, and i64MaxCount members of TServerDiskLogConfig.

To use DscServerDiskLogConfigSet it is recommended that the user invokes DscServerDiskLogConfigGet to get the server disk log configuration defaults, updates the pertinent fields, and then invokes DscServerDiskLogConfigSet.
See Also

DscStructPrepare, DscServerDiskLogConfigGet, DscServerDiskLogConfigSet
TXbPinConfig

The TXbPinConfig structure defines the behavior of DscXbPinConfigGet and DscXbPinConfigSet.

typedef struct tag_TXbPinConfig {
    int iInfoSize; // Size of this structure.
    int iInputType; // Input type.
    float fInputRange; // Input range voltage.
    float fInputOffset; // Input offset voltage.
    float fInputOffsetRange; // Input offset range voltage.
    float fOutputExcitation; // Output excitation voltage.
} TXbPinConfig;

Members
iInfoSize
Size of this information structure.

iInputType
Type of input signal, DC coupling, AC coupling or excitation. These constants simplify selecting the input type: DscXb_DCCoupling, DscXb_ACCoupling, DscXb_Excitation. The default is DscXb_DCCoupling.

fInputRange
Input range of the signal specified in Volts. Please note that this input range is different from DscInputRangeGet, and only works if DscInputRangeSet is set to +/- 5V.

If you select 0.5 the input range is +/- 500 mV, if you select 2.0 the input range +/- 2V. If you specify an invalid input range, the input range will not change from its previous setting. The default is 10.0.

Valid input ranges are:
0.01, 0.02, 0.05,
0.1, 0.2, 0.5,
1.0, 2.0, 5.0,
10.0

fInputOffset
Input offset of the signal specified in Volts. The input offset must be within the range of the input offset range. If you specify an invalid input offset, the input offset will not change from its previous setting. The default is 0.0.
\texttt{fInputOffsetRange}

Input offset range of the signal returned in Volts. The input offset range is determined by the input range. For example, if the input range is 0.5 then the input offset range is 2.5 for +/-2.5 V, if the input range is 2.0 then the input offset range is 1.0 for +/- 1V. This is a read only property that is dependent on the input range and you cannot specify it.

Valid input offset ranges are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Input Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.01</td>
</tr>
<tr>
<td>1.0</td>
<td>0.02</td>
</tr>
<tr>
<td>2.5</td>
<td>0.05</td>
</tr>
<tr>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
</tr>
<tr>
<td>2.5</td>
<td>0.5</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>5.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

\texttt{fOutputExcitation}

Output excitation signal specified in Volts. If you specify an invalid output excitation, the output excitation will not change from its previous setting. The default is 0.0.

Valid output excitation ranges are:

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0, 1.0, 2.0, 5.0, 10.0</td>
</tr>
</tbody>
</table>

**Remarks**

\texttt{TXbPinConfig} is used in \texttt{DscXbPinConfigGet} and \texttt{DscXbPinConfigSet}. \texttt{TXbPinConfig} should be initialized using \texttt{DscStructPrepare} or both functions will fail.

\texttt{DscXbPinConfigGet} will return the pin configuration in the \texttt{iInputType}, \texttt{fInputRange}, \texttt{fInputOffset}, \texttt{fInputOffsetRange}, and \texttt{fOutputExcitation} members of \texttt{TXbPinConfig}.

Before invoking \texttt{DscXbPinConfigSet}, the \texttt{iInputType}, \texttt{fInputRange}, \texttt{fInputOffset}, and \texttt{fOutputExcitation} members of \texttt{TXbPinConfig} must be set to the new desired values. \texttt{fInputOffsetRange} cannot be set since it is a read only property.

**See Also**

\texttt{DscStructPrepare, DscXbPinConfigGet, DscXbPinConfigSet}
**Function Summary**

The DSCIO interface provides a complete set of functions for communicating with the iDSC board. Each function falls into one of several categories.

<table>
<thead>
<tr>
<th>Category</th>
<th>Dsc Services</th>
</tr>
</thead>
</table>
| Graphical services     | DscConfigDialogOptionsGet  
DscConfigDialogOptionsSet  
DscConfigDialogShow      |
| Filter design services | DscFilterIndex  
DscFilterParametersGet  
DscFilterParametersSet  
DscPinToFilterMapGet  
DscPinToFilterMapSet  
DscTransferFunctionGet  
DscUnitStepGet  
DscUnitStepLengthGet   |
| Handle services        | DscHandleClose  
DscHandleOpen                                                                |
| Communication services | DscCalibrate  
DscCommandsLoad  
DscStartAcquiring  
DscStopAcquiring                                               |
| I/O services           | DscBufferAvail  
DscBufferGet  
DscBufferGetEnabledGet  
DscBufferGetEnabledSet  
DscBufferGetEx                                             |
| System services        | DscAddressGet  
DscAddressSet  
DscGroupDelay  
DscHardwareStop  
DscIdGet  
DscIdSet  
DscInputRangeGet  
DscInputRangeSet |
DscMemoryUsed
DscPinEnabledCount
DscPinEnabledGet
DscPinEnabledSet
DscRunning
DscSampleRateGet
DscSampleRateSet
DscScansDiscarded
DscStructPrepare

DAPL services
DscDaplTextGet
DscDaplTextLengthGet
DscDaplTextSet

Master/slave services
DscMasterGet
DscMasterSet
DscOperateModeGet
DscOperateModeSet
DscRemoteMasterGet
DscRemoteMasterSet
DscSlaveCount
DscSlaveHandle

External board services
DscXbCalibrate
DscXbEnabledGet
DscXbEnabledSet
DscXbPinConfigGet
DscXbPinConfigSet

Server disk log services
DscServerDiskLogBytes
DscServerDiskLogConfigGet
DscServerDiskLogConfigSet
DscServerDiskLogEnabledGet
DscServerDiskLogEnabledSet
DscServerDiskLogFileNameGet
DscServerDiskLogFileNameSet

Timing channel services
DscTcEnabledCount
DscTcEnabledGet
DscTcEnabledSet
DscTcMaximum
DscTcWidth

Error handling services
DscLastErrorTextGet
DscLastErrorTextSet

DSCIO DLL Programmer's Interface
DscSystemErrorProcess

Events
DscOnSystemErrorSet

Memory services
DscConfigRead
DscConfigWrite
DscConfigWriteSize

Category
DscGroup Services

Graphical services
DscGroupConfigDialogShow

Handle services
DscGroupHandleClose
DscGroupHandleOpen

Configuration services
DscGroupAddOne
DscGroupCount
DscGroupDeleteOne
DscGroupDsc

Memory services
DscGroupConfigRead
DscGroupConfigWrite
DscGroupConfigWriteSize
**DscAddressGet**

The `DscAddressGet` function gets the machine name and iDSC board name. The name returned is in UNC format.

```c
int __stdcall DscAddressGet(
    HDSC hDsc,           // iDSC board handle.
    int iSize,           // Size of pszAddress buffer.
    const char *pszAddress // Pointer to buffer of characters.
);
```

### Parameters

**hDsc**
- Handle of the iDSC board.

**iSize**
- Size of the `pszAddress` buffer that stores the DscAddress. The buffer is allocated by the application.

**pszAddress**
- Pointer to a buffer that stores the DscAddress. The buffer is allocated by the application.

### Return Values

If the function succeeds, the return value is the length of the string. If the function fails, the return value is -1.

### Remarks

`DscAddressGet` returns the machine name and the iDSC board name. An example is `\\.\Dap0`, where `\.` denotes the local machine and `\Dap0` is the name of the iDSC board.

### See Also

`DscAddressSet`, `UNC`
DscAddressSet

The DscAddressSet function sets the machine name and iDSC board name. The name returned is in UNC format.

```
int __stdcall DscAddressSet(
    HDSC hDsc,           // iDSC board handle.
    const char *pszAddress  // UNC name.
);
```

Parameters

- **hDsc**
  - Handle of the iDSC board.

- **pszAddress**
  - Pointer to a UNC name that specifies which iDSC board to open.

Return Values

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remarks

DscAddressSet takes the parameter pszAddress, that consists of two portions, the machine name and the iDSC board name. The naming method is based on the UNC.

Using `\.,\Dap0` as an example, \. denotes the local machine and \Dap0 is the name of the iDSC board. If you are connected through DAPcell to a remote machine named PC45, then `pszAddress` is `\PC45\Dap0`.

See Also

DscAddressGet, UNC
**DscBufferAvail**

The *DscBufferAvail* function gets the number of bytes available for reading from the iDSC board.

```c
int __stdcall DscBufferAvail(
    HDSC hDsc
); // iDSC board handle.
```

**Parameters**

*hDsc*

Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the number of bytes available for reading. If the function fails, the return value is -1. If there are no data available, the return value is 0.

**Remarks**

*DscBufferAvail* returns the number of bytes already buffered. An application is safe to read that number from the iDSC board without being blocked.

For efficient data transfer, it is best to use *DscBufferGetEx* with a time wait, and avoid using *DscBufferAvail*. *DscBufferGetEx* returns the actual number of bytes read which lets the application know what data have been transferred.

**See Also**

*DscBufferGet, DscBufferGetEx*
**DscBufferGet**

The **DscBufferGet** function reads a block of data from the iDSC board.

```c
int __stdcall DscBufferGet(
    HDSC hDsc, // iDSC board handle.
    int iBytes, // Number of bytes to read.
    int iTimeWait, // Longest time to wait for data.
    void *pvBuffer // Address of buffer to receive data.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iBytes`  
  Number of bytes to read.

- `iTimeWait`  
  Longest time in milliseconds that the get operation can be blocked waiting for data. If no data shows up in that amount of time, the operation should be aborted.

- `pvBuffer`  
  Pointer to a buffer for storing the data from the iDSC board.

**Return Values**

If the function succeeds, the return value is the number of bytes read. If the function fails, the return value is -1. If there are no data available, the return value is 0.

**Remarks**

**DscBufferGet** attempts to read all the requested `iBytes` from the iDSC board. If, all the requested `iBytes` are not available for `iTimeWait` milliseconds, it returns with the number of bytes read so far.

An application that cannot guarantee data availability should use **DscBufferGetEx** to avoid waiting for data.

This function is useful for displaying the acquired and filtered data in graphs, tables, etc.
See Also
DscBufferGetEx
DscBufferGetEnabledGet

The DscBufferGetEnabledGet function gets the state of the DscBufferGet function and DscBufferGetEx function.

```c
int __stdcall DscBufferGetEnabledGet(
    HDSC hDsc  // iDSC board handle.
);
```

**Parameters**

- **hDsc**
  
  Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is 0 for disabled and 1 for enabled. If the function fails, the return value is -1.

**Remarks**

DscBufferGetEnabledGet returns the state of the DscBufferGet function and DscBufferGetEx function. If DscBufferGetEnabledGet is 0, DscBufferGet and DscBufferGetEx are disabled, and invoking those functions will return -1. If DscBufferGetEnabledGet is 1, DscBufferGet and DscBufferGetEx are enabled.

**See Also**

DscBufferGet, DscBufferGetEnabledSet, DscBufferGetEx
**DscBufferGetEnabledSet**

The `DscBufferGetEnabledSet` function sets the state of the `DscBufferGet` function and `DscBufferGetEx` function.

```c
int __stdcall DscBufferGetEnabledSet(
    HDSC hDsc, // iDSC board handle.
    int iBufferGetEnabled // DscBufferGet/GetEx state.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **iBufferGetEnabled**
  - DscBuffetGet and DscBufferGetEx state.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscBufferGetEnabledSet` sets the state of the `DscBufferGet` function and `DscBufferGetEx` function. If `iBufferGetEnabled` is 0, `DscBufferGet` and `DscBufferGetEx` are disabled, and invoking those functions will return -1. If `iBufferGetEnabled` is 1, `DscBufferGet` and `DscBufferGetEx` are enabled.

`DscBufferGetEnabledSet` is useful if another PC wants to access the iDSC data from the PC with the iDSC through networking. The networking capability is only available with the DAPcell Local and DAPcell servers. It is not available with the Accel32 server.

**See Also**

`DscBufferGet`, `DscBufferGetEnabledGet`, `DscBufferGetEx`
The `DscBufferGetEx` function reads a block of data from the iDSC board, using the `TBufferGetEx` structure to control its behavior.

```
int __stdcall DscBufferGetEx(
    HDSC hDsc,                  // iDSC board handle.
    const TBufferGetEx *pBufferGetEx, // Address of structure.
    void *pvBuffer             // Address of buffer to receive data.
);
```

**Parameters**

- **hDsc**
  Handle of the iDSC board.

- **pBufferGetEx**
  Pointer to a `TBufferGetEx` structure that passes the appropriate parameters. `TBufferGetEx` must be initialized using `DscStructPrepare`.

- **pvBuffer**
  Pointer to a buffer for storing the data from the iDSC board.

**Return Values**

If the function succeeds, the return value is the number of bytes read. If the function fails, the return value is -1. If there are no data available, the return value is 0.

**Remarks**

`DscBufferGetEx` is an extended version of `DscBufferGet`. It allows a minimum request count, `iMinBytes`, and a maximum request count, `iMaxBytes`, specification. Both `iMinBytes` and `iMaxBytes` are passed into this function through the `TBufferGetEx` structure. Both `iMinBytes` and `iMaxBytes` are in bytes and must be an integral multiple of `iBytesMultiple`. The function reads at least `iMinBytes` bytes of data. Once it reads enough data to cover `iMinBytes`, the function reads all available data up to `iMaxBytes` without waiting. The actual bytes returned is always an integral multiple of `iBytesMultiple`.

Before `iMinBytes` is covered, the function will be blocked waiting for data if the target pipe becomes empty. In this case, the two members of the `TBufferGetEx` structure, `iTimeWait` and `iTimeOut`, determine the behavior of the function. If
iMinBytes is not covered in iTimeout milliseconds, or if no data are available for iTimeoutWait milliseconds, the function returns immediately. The return value is then the number of bytes actually read up to the point where the operation is aborted. It can be zero or any integral multiple of iBytesMultiple less than iMinBytes. An application can check the return value to determine if a time-out has occurred.

See Also
DscBufferGet
**DscCalibrate**

The **DscCalibrate** function performs calibration on the iDSC board.

```c
int __stdcall DscCalibrate(
    HDSC hDsc // iDSC board handle.
);`
```

**Parameters**

*hDsc*  
Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is
0.

**Remarks**

**DscCalibrate** calibrates the iDSC board for DC gain and offset, and saves the calibration values.

**DscCalibrate** is automatically invoked when a user selects **DscStartAcquiring** for the first time. The next time a user uses the DLL, the saved calibration values are reused.

If the calibration values are not found, calibration is automatically re-invoked. A user can force recalibration by calling **DscCalibrate**.

**See Also**

**DscStartAcquiring**
**DscCommandsLoad**

The `DscCommandsLoad` function downloads the configuration commands to the iDSC board and performs the necessary configuration for filtering.

```c
int __stdcall DscCommandsLoad(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscCommandsLoad` configures the iDSC board with the appropriate programs and coefficients. The filter designs are used internally by `DscCommandsLoad` to calculate and download the appropriate commands.

If the iDSC board input or filter designs change and `DscStartAcquiring` is selected, `DscStartAcquiring` automatically invokes `DscCommandsLoad` and downloads new commands to the iDSC board. Data will show up at the PC two times `DscGroupDelay` seconds later.

If the iDSC board input or filter designs change and `DscCommandsLoad` is selected before `DscStartAcquiring`, data will show up immediately at the PC. Data that show up immediately are actually data that were sampled `DscGroupDelay` seconds ago.

**See Also**

`DscGroupDelay`, `DscStartAcquiring`
The `DscConfigDialogOptionsGet` function gets the display options of `DscConfigDialogShow`.

```c
int __stdcall DscConfigDialogOptionsGet(
    HDSC hDsc  // iDSC board handle.
);
```

**Parameters**

`hDsc`

Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the display options. If the function fails, the return value is -1.

**Remarks**

`DscConfigDialogOptionsGet` returns the display options as an integer value. The lowest eleven bits (bits 0 through bit 10) are used since there are eleven display options, `InputScreenHide`, `FDScreenHide` (filter design screen hide), `TcHide` (timing channels selection hide), `FD0Hide` (filter design index 0 hide), `FD1Hide`, `FD2Hide`, `FD3Hide`, `FD4Hide`, `FD5Hide`, `FD6Hide`, `FD7Hide`.

- `InputScreenHide` is bit 0, `FDScreenHide` is bit 1, `TcHide` is bit 2, `FD0Hide` is bit 3, `FD1Hide` is bit 4, `FD2Hide` is bit 5, `FD3Hide` is bit 6, `FD4Hide` is bit 7, `FD5Hide` is bit 8, `FD6Hide` is bit 9, `FD7Hide` is bit 10.

When the display options are enabled, the corresponding bits are set. When the display options are disabled, the corresponding bits are cleared.

**See Also**

`DscConfigDialogOptionsSet`, `DscConfigDialogShow`
DscConfigDialogOptionsSet

The DscConfigDialogOptionsSet function allows manipulating the display options of DscConfigDialogShow.

```c
int __stdcall DscConfigDialogOptionsSet(
    HDSC hDsc, // iDSC board handle.
    int iOptions // Display options.
);
```

**Parameters**
- **hDsc**
  Handle of the iDSC board.
- **iOptions**
  DscConfigDialogShow display options.

**Return Values**
If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
DscConfigDialogOptionsSet specifies the display options of DscConfigDialogShow. The default is DscConfigDialog_TcHide.

The constants below simplify selecting the display options.

- `DscConfigDialog_InputScreenHide` Hide the Input Screen.
- `DscConfigDialog_FDScreenHide` Hide the Filter Design Screen.
- `DscConfigDialog_TcHide` Hide the timing channels selection.
- `DscConfigDialog_FD0Hide` Hide filter design index 0.
- `DscConfigDialog_FD1Hide` Hide filter design index 1.
- `DscConfigDialog_FD2Hide` Hide filter design index 2.
- `DscConfigDialog_FD3Hide` Hide filter design index 3.
- `DscConfigDialog_FD4Hide` Hide filter design index 4.
- `DscConfigDialog_FD5Hide` Hide filter design index 5.
- `DscConfigDialog_FD6Hide` Hide filter design index 6.
- `DscConfigDialog_FD7Hide` Hide filter design index 7.
See Also
DscConfigDialogOptionsGet, DscConfigDialogShow
**DscConfigDialogShow**

The `DscConfigDialogShow` function displays modal dialog screens for graphical configuration of the inputs and filter designs.

```c
int __stdcall DscConfigDialogShow(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**
- `hDsc`: Handle of the iDSC board.

**Return Values**
- If the user selects the OK button when changes are made, the return value is `IDOK` (of value 1).
- If the user selects the OK button when no changes are made, the return value is `IDIGNORE` (of value 5).
- If the user selects the Cancel button, the return value is `IDCANCEL` (of value 2).
- If the function fails, the return value is 0.

**Remarks**
`DscConfigDialogShow` simplifies the configuration of the input and filter design process with a graphical interface. The Input Screen provides a quick method for selecting the sample rate, selecting the input range, mapping the input pins to selected filter designs, and enabling or disabling the input pins.
The figure below displays the Input Screen.

The Filter Design Screen allows easy manipulation of filter parameters like sharpness, low cutoff frequency, low cutoff slope, high cutoff frequency, high cutoff slope, and attenuation, either by entering a valid number or by adjusting a slider. When the user selects the right click button, there are several more options like Crosshair Track, Y Display (linear, linear zoom, log, log zoom, and unit step), Defaults Load, Copy, and Paste. The Copy (Ctrl-C) and Paste (Ctrl-V) feature copies the filter parameters from one filter design tab to another.
The figure below displays the Filter Design Screen.

A user can also design filters using the run-time design techniques available in the one of the pairs of functions below.

- `DscFilterParametersGet` & `DscFilterParametersSet`
- `DscPinToFilterMapGet` & `DscPinToFilterMapSet`

The Input Screen, Filter Design Screen, timing channels selection, and individual filter designs can be hidden using `DscConfigDialogOptionsSet`.

**See Also**
- `DscConfigDialogOptionsGet`, `DscConfigDialogOptionsSet`,
- `DscFilterParametersGet`, `DscFilterParametersSet`
DscConfigRead

The DscConfigRead function reads iDSC board information from memory.

```c
int __stdcall DscConfigRead(
    HDSC hDsc,       // iDSC board handle.
    int iSize,       // Size of memory allocated.
    void *pvBuffer   // Pointer to memory buffer.
);
```

Parameters

- **hDsc**
  - Handle of the iDSC board.

- **iSize**
  - Size of the memory buffer for the iDSC board information. The memory buffer is allocated by the application.

- **pvBuffer**
  - Pointer to the memory buffer that stores the iDSC board information. The memory buffer is allocated by the application.

Return Values

If the function succeeds, the return value is the number of bytes read from memory.
If the function fails, the return value is 0.

Remarks

DscConfigRead reads iDSC board information in binary format from memory. The information in memory is probably retrieved from a disk file.

The iDSC board information read includes the sample rate, input range, input pin to filter mappings, enabled input pins, iDSC address, and details of the filter designs. Note that when DscConfigRead is invoked, the iDSC address used in DscHandleOpen will be overwritten with the iDSC address that was stored in DscConfigRead.

*iSize* is the value that was previously used as the *iSize* parameter of DscConfigWrite when the iDSC board information was stored to memory. The user cannot use DscConfigWriteSize for *iSize* in DscConfigRead because it
will not return the correct size. *DscConfigWriteSize* is provided only for the *DscConfigWrite* operation.

**See Also**

*DscConfigWrite, DscConfigWriteSize*
**DscConfigWrite**

The `DscConfigWrite` function writes iDSC board information to memory.

```c
int __stdcall DscConfigWrite(
    HDSC hDsc,           // iDSC board handle.
    int iSize,           // Size of memory buffer.
    void *pvBuffer       // Pointer to memory buffer.
);
```

**Parameters**

- **hDsc**
  Handle of the iDSC board.

- **iSize**
  Size of the memory buffer for the iDSC board information. The memory buffer is allocated by the application.

- **pvBuffer**
  Pointer to the memory buffer that stores the iDSC board information. The memory buffer is allocated by the application.

**Return Values**

- If the function succeeds, the return value is the number of bytes written to memory.
- If the function fails, the return value is 0.

**Remarks**

`DscConfigWrite` writes iDSC board information in binary format to memory. The information in memory can then be written to a disk file.

The iDSC board information written includes the sample rate, input range, input pin to filter mappings, enabled input pins, iDSC address, and details of the filter designs.

`iSize` is the size of the memory buffer allocated and must be at least `DscConfigWriteSize`. `iSize` should be stored by the program to be used later as the `iSize` parameter of `DscConfigRead`. 

---

DSCIO DLL Programmer's Interface
See Also
DscConfigRead, DscConfigWriteSize
DscConfigWriteSize

The DscConfigWriteSize function returns the minimum number of bytes to allocate for DscConfigWrite.

    int __stdcall DscConfigWriteSize(
        HDSC hDsc               // iDSC board handle.
    );

Parameters

    hDsc
    Handle of the iDSC board.

Return Values

    If the function succeeds, the return value is the minimum number of bytes to allocate. If the function fails, the return value is 0.

Remarks

    DscConfigWriteSize must be used to determine the minimum number of bytes to allocate for the iSize parameter of DscConfigWrite. It cannot be used as the iSize parameter of DscConfigRead.

See Also

    DscConfigWrite, DscConfigRead
**DscDaplTextGet**

The `DscDaplTextGet` function gets the DAPL text defined for the iDSC.

```c
int __stdcall DscDaplTextGet(
    HDSC hDsc,       // iDSC board handle.
    int iSize,       // Size of `pszDaplText` buffer.
    const char *pszDaplText, // Pointer to buffer of characters.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iSize`  
  Size of the `pszDaplText` buffer for the DAPL text. The buffer is allocated by the application.

- `pszDaplText`  
  Pointer to a buffer that stores the DAPL text. The buffer is allocated by the application.

**Return Values**

If the function succeeds, the return value is the length of the string. If the function fails, the return value is -1.

**Remarks**

`DscDaplTextGet` returns the DAPL text defined for the iDSC. Each line of DAPL text is delimited by a carriage-return and line-feed.

The DAPL text is sent to the iDSC when the user invokes `DscCommandsLoad` or `DscStartAcquiring`.

**See Also**

`DscDaplTextLengthGet`, `DscDaplTextSet`
**DscDaplTextLengthGet**

The `DscDaplTextLengthGet` function gets the length of the DAPL text defined for the iDSC.

```c
int __stdcall DscDaplTextLengthGet(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the length of the buffer for the DAPL text. If the function fails, the return value is 0.

**Remarks**

`DscDaplTextLengthGet` returns the length of the buffer for the DAPL text defined for the iDSC. This includes an extra character at the end for the null-terminator.

`DscDaplTextLengthGet` allows the user to determine the size of buffer to allocate when using the `DscDaplTextGet` function.

**See Also**

`DscDaplTextGet`, `DscDaplTextSet`
**DscDaplTextSet**

The **DscDaplTextSet** function defines the DAPL text for the iDSC.

```c
int __stdcall DscDaplTextSet(
    HDSC hDsc,              // iDSC board handle.
    const char *pszDaplText  // DAPL text.
);
```

**Parameters**
- **hDsc**
  - Handle of the iDSC board.
- **pszDaplText**
  - Pointer to the DAPL text.

**Return Values**
- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
- **DscDaplTextSet** allows the user to define custom DAPL text for the iDSC. Each line of DAPL text must be delimited by a carriage-return, line-feed, or both.

  The DAPL text is sent to the iDSC when the user invokes **DscCommandsLoad** or **DscStartAcquiring**.

**See Also**
- **DscDaplTextGet, DscDaplTextLengthGet**
DscFilterIndex

The **DscFilterIndex** function returns the filter index when given a filter name.

```c
int __stdcall DscFilterIndex(
    HDSC hDsc,       // iDSC board handle.
    const char *pszFilterName  // Filter name.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **pszFilterName**
  - Pointer to a filter name.

**Return Values**

If the function succeeds, the return value is the filter index. So, if the filter name of interest is associated with filter index 0, the return value is 0. If the function fails, the return value is -1. For example, if `pszFilterName` is an invalid name, failure occurs.

**Remarks**

**DscFilterIndex** determines the filter index associated with a particular filter name. Since there are a maximum of eight filter designs, valid filter indices are 0 through 7.

The filter index is useful in functions like **DscFilterParametersGet**, **DscFilterParametersSet**, **DscPinToFilterMapGet** and **DscPinToFilterMapSet**.

**See Also**

**DscFilterParametersGet**, **DscFilterParametersSet**
DscFilterParametersGet

The DscFilterParametersGet function gets the filter parameters associated with a filter index. The filter parameters include the name, type, sharpness, cutoff frequency, cutoff slope, and attenuation.

```c
int __stdcall DscFilterParametersGet(
    HDSC hDsc, // iDSC board handle.
    int iFilterIndex, // Filter index.
    TFilterParam *pFilterParam // Address of structure.
);
```

**Parameters**

- **hDsc**: Handle of the iDSC board.
- **iFilterIndex**: Filter index of interest. Valid filter indices are 0 through 7.
- **pFilterParam**: Pointer to a TFilterParam structure that receives the filter parameters. TFilterParam must be initialized using DscStructPrepare.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

DscFilterParametersGet returns the filter parameters for an associated filter index in the TFilterParam structure. TFilterParam must be initialized using DscStructPrepare before invoking DscFilterParametersGet or the function will fail.

Valid filter indices are 0 through 7. If the filter index is invalid, the function will fail.

**See Also**

DscConfigDialogShow, DscFilterParametersSet, TFilterParam
**DscFilterParametersSet**

The **DscFilterParametersSet** function sets the filter parameters associated with a filter index. The filter parameters include the name, type, sharpness, cutoff frequency, cutoff slope, and attenuation.

```c
int __stdcall DscFilterParametersSet(
    HDSC hDsc, // iDSC board handle.
    int iFilterIndex, // Filter index.
    const TFilterParam *pFilterParam // Address of structure.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **iFilterIndex**
  - Filter index of interest. Valid filter indices are 0 through 7.

- **pFilterParam**
  - Pointer to a **TFilterParam** structure that passes the filter parameters. **TFilterParam** must be initialized using **DscStructPrepare**.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscFilterParametersSet** modifies the filter parameters for an associated filter index. All members of **TFilterParam** must be set before invoking **DscFilterParametersSet**. **TFilterParam** must be initialized using **DscStructPrepare** before invoking **DscFilterParametersSet** or the function will fail. The **achName**, **iFilterType**, **iSharpness**, **fCutoffFreqLow**, **fCutoffSlopeLow**, **fCutoffFreqHigh**, **fCutoffSlopeHigh**, and **fAttenuation** members of **TFilterParam** must be set to the new desired values. Note that the filter parameters can be graphically set in **DscConfigDialogShow**.
Valid filter indices are 0 through 7. If the filter index is invalid, the function will fail.

See Also

DscConfigDialogShow, DscFilterParametersGet, TFilterParam
**DscGroupDelay**

The **DscGroupDelay** function returns the group delay in seconds for all of the filter designs.

```c
int __stdcall DscGroupDelay(
    HDSC hDsc,   // iDSC board handle.
    float *fGroupDelay  // Address to receive group delay.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **fGroupDelay**
  - Pointer to a float that receives the group delay. The float variable is allocated by the application.

**Return Values**

- If the function succeeds, the return value is 1.
- If the function fails, the return value is 0.

**Remarks**

**DscGroupDelay** informs a user of the group delay (in seconds) through all the filter designs. The group delay is the amount of time to wait before data start showing up at the PC, and does not include the filter designs of disabled input pins.

If the iDSC board inputs or filter designs change and **DscStartAcquiring** is selected, **DscStartAcquiring** automatically invokes **DscCommandsLoad** and downloads new commands to the iDSC board. Data will show up at the PC two times **DscGroupDelay** seconds later.

If the iDSC board inputs or filter designs change and **DscCommandsLoad** is selected before **DscStartAcquiring**, data will show up immediately at the PC. Data that show up immediately are actually data that were sampled **DscGroupDelay** seconds ago.

**See Also**

- **DscCalibrate, DscCommandsLoad, DscStartAcquiring**
**DscHandleClose**

The **DscHandleClose** function releases a handle previously opened with **DscHandleOpen**.

```c
int __stdcall DscHandleClose(
    HDSC hDsc // iDSC board handle to close.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board to close. It must be a handle previously opened by **DscHandleOpen**.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

- **DscHandleClose** terminates communication with the iDSC board.
- **DscHandleClose** is the last function called by any application ready to end communication with the iDSC board.

**See Also**

- **DscHandleOpen**
DscHandleOpen

The `DscHandleOpen` function returns a handle to the target iDSC board when given a UNC name. The UNC name includes the machine name and the iDSC board name set by `DscAddressSet` and returned by `DscAddressGet`.

```
HDSC __stdcall DscHandleOpen(
    const char *pszAddress // UNC name of the iDSC board.
);
```

**Parameters**

`pszAddress`

Pointer to a UNC target name string that specifies which iDSC board to open.

**Return Values**

If the function succeeds, the return value is an open handle to the specified target. If the function fails, the return value is a NULL handle (of value 0).

**Remarks**

`DscHandleOpen` initiates communication with the iDSC board. `DscHandleOpen` is the first function called by any application ready to begin communication with the iDSC board. It returns a handle to the target iDSC board when given a UNC name.

A handle is a 32-bit value that references the iDSC board. This handle is used in all `DscXxxx` services to reference the appropriate iDSC board.

**See Also**

`DscAddressGet, DscAddressSet, DscHandleClose`
DscHardwareStop

The DscHardwareStop function stops the hardware on the iDSC board.

```c
int __stdcall DscHardwareStop(
    HDSC hDsc, // iDSC board handle.
);
```

Parameters

- **hDsc**
  - Handle of the iDSC board.

Return Values

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remarks

DscHardwareStop stops and resets the hardware on the iDSC board.

See Also

DscRunning
**DscIdGet**

The **DscIdGet** function gets the identification name of the iDSC board.

```c
int __stdcall DscIdGet(
    HDSC hDsc,      // iDSC board handle.
    int iSize,      // Size of pszId buffer.
    const char *pszId  // Pointer to buffer of characters.
);
```

**Parameters**

- **hDsc**
  Handle of the iDSC board.

- **iSize**
  Size of the pszId buffer that stores the DscId. The buffer is allocated by the application.

- **pszId**
  Pointer to a buffer that stores the DscId. The buffer is allocated by the application.

**Return Values**

If the function succeeds, the return value is the length of the string. If the function fails, the return value is -1.

**Remarks**

**DscIdGet** returns the identification name of the iDSC board. The DscId is unrelated to the DscAddress. The default identification names for the iDSC boards on a system are iDsc0, iDsc1, ..., etc. A maximum of 14 iDSC boards may run on one system.

**See Also**

**DscIdSet**
The **DscIdSet** function sets the identification name of the iDSC board.

```c
int __stdcall DscIdSet(
    HDSC hDsc,           // iDSC board handle.
    const char *pszId    // Identification name.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **pszId**
  - Pointer to an identification name for the iDSC board.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscIdSet** allows a user to assign a unique identification name to the iDSC board.

The default identification names for the iDSC boards on a system are **iDsc0**, **iDsc1**, ..., etc. A maximum of 14 iDSC boards may run on one system.

**See Also**

**DscIdGet**
**DscInputRangeGet**

The `DscInputRangeGet` function gets the input range of the iDSC board.

```c
int __stdcall DscInputRangeGet(
    HDSC hDsc  // iDSC board handle.
);
```

**Parameters**

- `hDsc`: Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the input range. If the function fails, the return value is 0.

**Remarks**

`DscInputRangeGet` returns the input range on the iDSC board. The return value will be either 5000 to represent +/- 5 Volts or 10000 to represent +/- 10 Volts.

**See Also**

`DscInputRangeSet`
**DsclnputRangeSet**

The `DsclnputRangeSet` function sets the input range of the iDSC board.

```c
int __stdcall DsclnputRangeSet(
    HDSC hDsc,            // iDSC board handle.
    int iInputRange       // Input range.
);
```

**Parameters**

- `hDsc` Handle of the iDSC board.
- `iInputRange` Input range to set.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DsclnputRangeSet` configures the input range on the iDSC board. A user can select either +/- 5 Volts or +/- 10 Volts.

To use this property the voltages must be specified as absolute values in millivolts. For example, 5000 represents +/- 5 Volts and 10000 represents +/- 10 Volts.

If the user selects an invalid input range, the function will fail and return an error code of 0. The error can be retrieved using `DsclnLastErrorTextGet`.

**See Also**

- `DsclnputRangeGet`
DscLastErrorTextGet

The `DscLastErrorTextGet` function gets the last error message that occurred in the DSCIO DLL.

```c
int __stdcall DscLastErrorTextGet(
    int iSize, // Size of pszError buffer.
    const char *pszError // Pointer to buffer of characters.
);
```

**Parameters**

- `iSize`
  - Size of the `pszError` buffer that stores the last error message. The buffer is allocated by the application.

- `pszError`
  - Pointer to a buffer that stores the last error message. The buffer is allocated by the application.

**Return Values**

The function always returns the length of the string. If the function returns 0, it could mean that the string is empty or `iSize` is invalid.

**Remarks**

`DscLastErrorTextGet` returns the last error message that occurred in the DSCIO DLL. An error occurs in the DSCIO DLL when a function call fails. To use `DscLastErrorTextGet` correctly, it must be called immediately after the function of interest.

If a function call does not fail, the last error message is empty. If a function call fails, the last error message is updated by the last function call that failed. The last error message will stay the same until another function call fails. To clear the last error message, use the `DscLastErrorTextSet` function.

**See Also**

`DscLastErrorTextSet`
**DscLastErrorTextSet**

The *DscLastErrorTextSet* function updates the last error message.

```c
int __stdcall DscLastErrorTextSet(
    const char *pszError
    // Error message
);
```

**Parameters**

*pszError*

Pointer to an error message.

**Return Values**

The function always returns 1.

**Remarks**

*DscLastErrorTextSet* allows the user to update the last error message. However, if a function fails in the DSCIO DLL, the function will update the last error message. *DscLastErrorTextSet* is only useful if the user has a reason to clear or change the last error message.

**See Also**

*DscLastErrorTextGet*
DscMasterGet

The DscMasterGet function gets the handle of a Master iDSC board when invoked on a Slave iDSC board. DscMasterGet is only useful in a Master/Slave Configuration.

HDSC __stdcall DscMasterGet(
    HDSC hDscSlave // Slave iDSC board handle.
);

Parameters
hDscSlave
Handle of the Slave iDSC board.

Return Values
If the function succeeds, the return value is the Master iDSC board handle. If the function fails, the return value is a NULL handle (of value 0).

Remarks
DscMasterGet gives access to all Master iDSC boards on a system when invoked on a Slave iDSC board in a Master/Slave Configuration. If DscMasterGet is invoked on a Master or Normal iDSC board, the return value is a NULL handle.

See Also
DscMasterSet, DscOperateModeGet, DscOperateModeSet, DscRemoteMasterSet
**DscMasterSet**

The `DscMasterSet` function connects a Slave iDSC board to a Master iDSC board. `DscMasterSet` is only useful in a Master/Slave Configuration.

```c
int __stdcall DscMasterSet(
    HDSC hDscSlave, // Slave iDSC board handle.
    HDSC hDscMaster  // Slave Master handle to set.
);
```

**Parameters**
- `hDscSlave` Handle of the Slave iDSC board.
- `hDscMaster` Handle of the Master iDSC board to set to the Slave iDSC board.

**Return Values**
If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
`DscMasterSet` connects a Slave iDSC board to a Master iDSC board in a Master/Slave configuration. The only way to specify the iDSC board as a master or slave is by invoking `DscMasterSet` using two unique iDSC board handles.

When `DscMasterSet` completes, the specified slave becomes a Slave iDSC board and the specified master becomes a Master iDSC board. `DscMasterSet` will automatically update the operational state of the iDSC board in `DscOperateModeGet`. If non-unique iDSC board handles are used, the Master and Slave iDSC boards are not connected.

**See Also**
- `DscMasterGet`, `DscOperateModeGet`, `DscOperateModeSet`, `DscRemoteMasterSet`
DscMemoryUsed

The function **DscMemoryUsed** displays the used memory on the iDSC board.

```c
int __stdcall DscMemoryUsed(
    HDSC hDsc, // iDSC board handle.
);
```

**Parameters**

* **hDsc**
  - Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the amount of memory used, expressed in tenths of a percent. If the function fails, the return value is 0.

**Remarks**

**DscMemoryUsed** displays the used memory on the iDSC board in tenths of a percent. As an example, if **DscMemoryUsed** returns 42, it means that 4.2% of memory is used.

**DscMemoryUsed** is useful in determining whether or not the iDSC board will be able to sustain a particular sample rate without overflowing.
**DscOnSystemErrorSet**

The **DscOnSystemErrorSet** function runs when the system encounters a failure.

```c
int __stdcall DscOnSystemErrorSet(
    HDSC hDsc, // iDSC board handle.
    TProcSystemErrorStdcall *Proc, // Pointer to function.
    void *pInfo // Pointer to additional information.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **Proc**
  - Pointer to function with the format **TProcSystemErrorStdcall**.

- **pInfo**
  - Pointer to additional information to satisfy type calling conventions.

**Return Values**

If the function succeeds, the return value is the pointer to a null-terminated message text string. If the function fails, the return value is the pointer to a null string.

**Remarks**

**DscOnSystemErrorSet** is useful in determining if the system has encountered a failure since it provides a way to access the errors. **DscOnSystemErrorSet** runs when **DscSystemErrorProcess** is invoked. **DscSystemErrorProcess** is automatically invoked during **DscBufferGetEx** if there are errors from the iDSC. The user can also call **DscSystemErrorProcess** and invoke **DscOnSystemErrorSet** if errors from the iDSC are suspected.

An example of a system error is an input channel pipe overflow when the sample rate of the iDSC board is too high for the PC to keep up.

**See Also**

**DscSystemErrorProcess, TProcSystemErrorStdcall**
**DscOperateModeGet**

The `DscOperateModeGet` function gets the operational mode of the iDSC board. Valid operational modes are Normal, Master, or Slave.

```c
int __stdcall DscOperateModeGet(
    HDSC hDsc);  // iDSC board handle.
```

**Parameters**
- `hDsc` Handle of the iDSC board.

**Return Values**
- If the function succeeds, the return value is the operate mode. If the function fails, the return value is -1.
- If the operate mode is Normal, the return value is the constant `Dsc_Normal`.

**Remarks**

`DscOperateModeGet` informs a user of the operational mode of the iDSC board. The values returned for the operational mode correspond to the constants below.
- `Dsc_Normal`, `Dsc_Master`, `Dsc_Slave`

**See Also**
- `DscMasterGet`, `DscMasterSet`, `DscOperateModeSet`, `DscRemoteMasterSet`
**DscOperateModeSet**

The **DscOperateModeSet** function sets the operational mode of the iDSC board. Valid operational modes are Normal, Master, or Slave.

```c
int __stdcall DscOperateModeSet(
    HDSC hDsc,  // iDSC board handle.
    int iOperateMode  // Selected operate mode.
);
```

**Parameters**
- **hDsc**
  - Handle of the iDSC board.
- **iOperateMode**
  - Operate mode to select. Use only the constant **Dsc_Normal**.

**Return Values**
- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
- **DscOperateModeSet** is provided to allow the user to select a Normal iDSC board. **DscOperateModeSet** cannot be used to specify Master and Slave iDSC boards. Instead, **DscMasterSet** should be used to set up Master and Slave iDSC boards. **DscMasterSet** will automatically update the mode to the correct state.
- The constants below simplify selecting the operate mode.
  - **Dsc_Normal**, **Dsc_Master**, **Dsc_Slave**
- The following call may be used to set the operational mode to Normal.
  ```c
  DscOperateModeSet(hDsc, Dsc_Normal);
  ```

**See Also**
- **DscMasterGet**, **DscMasterSet**, **DscOperateModeGet**, **DscRemoteMasterSet**
DscPinEnabledCount

The **DscPinEnabledCount** function returns the number of enabled input pins on the iDSC board.

```c
int __stdcall DscPinEnabledCount(
    HDSC hDsc  // iDSC board handle.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the number of enabled input pins. If the function fails, the return value is 0.

**Remarks**

**DscPinEnabledCount** returns the number of enabled input pins on the iDSC board. It is related to the **DscPinEnabledGet** and **DscPinEnabledSet** functions. Valid values for the enabled input pin count are in the range one to eight.

**See Also**

**DscPinEnabledGet, DscPinEnabledSet**
**DscPinEnabledGet**

The **DscPinEnabledGet** function gets the enabled input pins on the iDSC board.

```c
int __stdcall DscPinEnabledGet(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the enabled input pins. If the function fails, the return value is zero since at least one input pin must be enabled.

**Remarks**

**DscPinEnabledGet** returns the enabled input pins as an integer value. Only the lowest eight bits (bit 0 through bit 7) are used since there are a total of eight input pin options, \(A_0\), \(A_1\), \(A_2\), \(A_3\), \(A_4\), \(A_5\), \(A_6\), and \(A_7\). \(A_0\) is bit 0, \(A_1\) is bit 1, \(A_2\) is bit 2, \(A_3\) is bit 3, \(A_4\) is bit 4, \(A_5\) is bit 5, \(A_6\) is bit 6, and \(A_7\) is bit 7 of the returned integer value.

When the input pins are enabled, the corresponding bits are set. When the input pins are disabled, the corresponding bits are cleared. To find out the number of enabled input pins, use the **DscPinEnabledCount** function.

**See Also**

**DscPinEnabledCount, DscPinEnabledSet**
**DscPinEnabledSet**

The **DscPinEnabledSet** function sets the enabled input pins on the iDSC board.

```c
int __stdcall DscPinEnabledSet(
    HDSC hDsc,               // iDSC board handle.
    int iPinEnabled          // Input pins to enable.
);                        
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.
- **iPinEnabled**
  - Input pins to enable.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscPinEnabledSet** enables or disables the input pins using the **iPinEnabled** integer value. Only the lowest eight bits (bit 0 through bit 7) are used since there are a total of eight input pin options, A0, A1, A2, A3, A4, A5, A6, and A7. A0 is bit 0, A1 is bit 1, A2 is bit 2, A3 is bit 3, A4 is bit 4, A5 is bit 5, A6 is bit 6, and A7 is bit 7 of **iPinEnabled**.

When the input pins are enabled, the corresponding bits are set. When the input pins are disabled, the corresponding bits are cleared. To find out the number of enabled input pins, use the **DscPinEnabledCount** function.

The constants below simplify enabling the input pins.

- DscPin_A0, DscPin_A1, DscPin_A2, DscPin_A3, DscPin_A4, DscPin_A5, DscPin_A6, DscPin_A7

The following example is used to enable input pins A1, A3 and A6.

```c
DscPinEnabledSet(hDsc, DscPin_A1|DscPin_A3|DscPin_A6);
```
See Also

DscPinEnabledCount, DscPinEnabledGet
**DscPinToFilterMapGet**

The **DscPinToFilterMapGet** function gets the mapping of a filter index to an input pin index.

```c
int __stdcall DscPinToFilterMapGet(
    HDSC hDsc,        // iDSC board handle.
    int iPinIndex     // Input pin index.
);
```

**Parameters**

- **hDsc**: Handle of the iDSC board.
- **iPinIndex**: Input pin index of interest. Valid input pin indices are 0 through 7.

**Return Values**

If the function succeeds, the return value is the filter index. If the function fails, the return value is -1.

If the input pin of interest is mapped to filter index 0, the return value is 0.

**Remarks**

**DscPinToFilterMapGet** returns the filter index that is mapped to an input pin index. The default mapping is: input pin index 0 to filter index 0, input pin index 1 to filter index 1, ..., input pin index 7 to filter index 7.

Valid input pin indices are 0 through 7, which correspond to input pins A0 through A7. Valid filter indices are 0 through 7. If an input pin index is invalid, the function will fail.

**See Also**

*DscConfigDialogShow, DscPinToFilterMapSet*
DscPinToFilterMapSet

The **DscPinToFilterMapSet** function sets the mapping of a filter index to an input pin index.

```c
int __stdcall DscPinToFilterMapSet(
    HDSC hDsc, // iDSC board handle.
    int iPinIndex, // Input pin index.
    int iFilterIndex // Filter index to map.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.
- **iPinIndex**
  - Input pin index of interest. Valid input pin indices are 0 through 7.
- **iFilterIndex**
  - Filter index to map to the input pin index. Valid filter indices are 0 through 7.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscPinToFilterMapSet** associates a filter index to a corresponding input pin index.

Valid input pin indices are 0 through 7, which correspond to input pins A0 through A7. Valid filter indices are 0 through 7. If an input pin index or filter index is invalid, the function will fail.

**See Also**

- **DscConfigDialogShow**, **DscPinToFilterMapGet**
DscRemoteMasterGet

The DscRemoteMasterGet function gets the remote master state of the Master/Slave Configuration.

```c
int __stdcall DscRemoteMasterGet(
    HDSC hDsc        // iDSC board handle.
);
```

Parameters

- `hDsc` Handle of the iDSC board.

Return Values

- If the function succeeds, the return value is 0 for disabled and 1 for enabled. If the function fails, the return value is -1.

Remarks

DscRemoteMasterGet returns the remote master state of the master/slave configuration. If DscRemoteMasterGet is 0, the remote master state is disabled. If DscRemoteMasterGet is 1, the remote master state is enabled.

The remote master state enables the user to synchronize multiple iDSCs across PCs. It only works if the MSXB 045 hardware is present. Please refer to the MSXB 045 hardware manual for more description on the hardware.

See Also

DscMasterSet, DscOperateModeGet, DscRemoteMasterSet
**DscRemoteMasterSet**

The **DscRemoteMasterSet** function sets the remote master state of the Master/Slave Configuration.

```c
int __stdcall DscRemoteMasterSet(
    HDSC hDsc, // iDSC board handle.
    int iRemoteMaster // Remote master state.
);
```

**Parameters**
- **hDsc**
  Handle of the iDSC board.
- **iRemoteMaster**
  Remote master state.

**Return Values**
- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
- **DscRemoteMasterSet** sets the remote master state of the master/slave configuration. If `iRemoteMaster` is 0, the remote master state is disabled. If `iRemoteMaster` is 1, the remote master state is enabled.

The remote master state enables the user to synchronize multiple iDSCs across PCs. It only works if the MSXB 045 hardware is present. Please refer to the MSXB 045 hardware manual for more description on the hardware.

**See Also**
- **DscMasterGet, DscOperateModeGet, DscRemoteMasterSet**
DscRunning

The **DscRunning** function returns the state of the iDSC board.

```c
int __stdcall DscRunning(
    HDSC hDsc,               // iDSC board handle.
);                        //
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is 0 for not running and 1 for running. If the function fails, the return value is -1.

**Remarks**

**DscRunning** specifies whether the iDSC is running. Once **DscStartAcquiring** is invoked, **DscRunning** will return 1. Once **DscStopAcquiring** is invoked, **DscRunning** will return 0.

**See Also**

[DscHardwareStop, DscStartAcquiring, DscStopAcquiring]
**DscSampleRateGet**

The **DscSampleRateGet** function gets the effective sampling rate per channel on the iDSC board.

```c
int __stdcall DscSampleRateGet(HDSC hDsc);  // iDSC board handle.
```

**Parameters**
- **hDsc**
  Handle of the iDSC board.

**Return Values**
- If the function succeeds, the return value is the sampling rate. If the function fails, the return value is 0.

**Remarks**
- **DscSampleRateGet** returns the effective sampling rate for each channel on the iDSC board in units of samples per second. There are two effective sampling rates in the highest octave and lowest octave, and three effective sampling rates in every other octave.

Note: See **DscSampleRateSet** for the valid sampling rates.

**See Also**
- **DscConfigDialogShow, DscSampleRateSet**
DscSampleRateSet

The `DscSampleRateSet` function sets the effective sampling rate per channel on the iDSC board.

```c
int __stdcall DscSampleRateSet(
    HDSC hDsc,                // iDSC board handle.
    int iSampleRate           // Sample rate.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iSampleRate`  
  Effective sample rate to set.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscSampleRateSet` sets the effective sampling rate for each channel on the iDSC board in units of samples per second. The table below displays the valid sample rates arranged in octaves. There are two effective sampling rates in the highest octave.

<table>
<thead>
<tr>
<th>Octave</th>
<th>Sample Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102400, 76800</td>
</tr>
<tr>
<td>2</td>
<td>51200, 38400</td>
</tr>
<tr>
<td>3</td>
<td>25600, 19200</td>
</tr>
<tr>
<td>4</td>
<td>12800, 9600, 7680</td>
</tr>
<tr>
<td>5</td>
<td>6400, 5120, 4800</td>
</tr>
<tr>
<td>6</td>
<td>3200, 3072, 2560, 2400, 2048, 1920</td>
</tr>
<tr>
<td>7</td>
<td>1600, 1536, 1280, 1200, 1024, 960</td>
</tr>
<tr>
<td>8</td>
<td>800, 768, 640, 600, 512, 480</td>
</tr>
<tr>
<td>9</td>
<td>400, 384, 320, 300, 256, 240</td>
</tr>
<tr>
<td>10</td>
<td>200, 192, 160, 150, 128, 120</td>
</tr>
</tbody>
</table>

153600
If an invalid sample rate is selected, the function will automatically select the closest larger sample rate. The function is always successful in this case.

When the sampling rate is changed, the \textit{iSharpness}, \textit{fCutoffSlope}, and \textit{fAttenuation} members of \texttt{TFilterParam} will return their default values when used with \texttt{DscFilterParametersGet}. Only \textit{fCutoffFreq} will retain its previously set value if the set value is valid. If the previously set value of \textit{fCutoffFreq} is out of range, then \textit{fCutoffFreq} will also return the default value.

\textbf{See Also}  
\texttt{DscConfigDialogShow}, \texttt{DscSampleRateGet}
DscScansDiscarded

The **DscScansDiscarded** function returns the number of scans thrown away since **DscCommandsLoad**.

```cpp
int __stdcall DscScansDiscarded(
    HDSC hDsc,                   // iDSC board handle.
    __int64 *i64Discarded       // Address to receive the scans.
);
```

**Parameters**
- **hDsc**
  - Handle of the iDSC board.
- **i64Discarded**
  - Pointer to a 64-bit integer that receives the scans discarded. The __int64 variable is allocated by the application.

**Return Values**
- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
- **DscScansDiscarded** informs the user of the scans discarded since **DscCommandsLoad**. This information is useful in calculating timing delays.
- A scan is a set of enabled input pins. For example, if there are five enabled input pins, a scan consists of five samples, if there are two enabled input pins, a scan consists of two samples.

**See Also**
- **DscCommandsLoad**
DscServerDiskLogBytes

The **DscServerDiskLogBytes** function returns the number of bytes logged to disk by the server.

```c
int __stdcall DscServerDiskLogBytes(
    HDSC hDsc, // IDSC board handle.
    TDscIoInt64 *i64Bytes // Address to receive logged bytes.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **i64Bytes**
  - Pointer to a **TDscIoInt64** structure that receives the number of bytes logged.
  - The **TDscIoInt64** variable is allocated by the application.

**Return Values**

- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

- **DscServerDiskLogBytes** informs the user of the number of bytes logged to disk by the server. It is a **TDscIoInt64** structure. It should be called after **DscStartAcquiring** is invoked if **DscServerDiskLogEnabledGet** is 1.
- **DscServerDiskLogBytes** will return 0 if no data has been logged to disk.

**See Also**

- **DscServerDiskLogEnabledGet**, **DscServerDiskLogEnabledSet**,
- **DscServerDiskLogConfigGet**, **DscServerDiskLogConfigSet**
DscServerDiskLogConfigGet

The `DscServerDiskLogConfigGet` function gets the server disk log configuration through `TServerDiskLogConfig`.

```c
int __stdcall DscServerDiskLogConfigGet(
    HDSC hDsc,            // iDSC board handle.
    TServerDiskLogConfig *pServerDiskLogConfig,  // Address of structure.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `pServerDiskLogConfig`  
  Pointer to a `TServerDiskLogConfig` structure that receives the server disk log configuration. `TServerDiskLogConfig` must be initialized using `DscStructPrepare`.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscServerDiskLogConfigGet` returns the server disk log configuration in the `TServerDiskLogConfig` structure. `TServerDiskLogConfig` must be initialized using `DscStructPrepare` before invoking `DscServerDiskLogConfigGet` or the function will fail.

The `pszFileName` field of `TServerDiskLogConfig` must be initialized to point to the user allocated buffer. The size of the user allocated buffer must be specified using the `dwFileNameSize` field. It must include an extra space for the null terminator.

If `dwFileNameSize` is 0, the file name is not returned in `pszFileName`. If `dwFileNameSize` is 1, only the null terminator is returned in `pszFileName`. 
See Also
DscServerDiskLogConfigSet, TServerDiskLogConfig
**DscServerDiskLogConfigSet**

The `DscServerDiskLogConfigSet` function sets the server disk log configuration through `TServerDiskLogConfig`.

```c
int __stdcall DscServerDiskLogConfigSet(
    HDSC hDsc, // iDSC board handle.
    const TServerDiskLogConfig *pServerDiskLogConfig, // Address of structure.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **pServerDiskLogConfig**
  - Pointer to a `TServerDiskLogConfig` structure that passes the server disk log configuration. `TServerDiskLogConfig` must be initialized using `DscStructPrepare`.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscServerDiskLogConfigSet` modifies the server disk log configuration through the `TServerDiskLogConfig` structure. `TServerDiskLogConfig` must be initialized using `DscStructPrepare` before invoking `DscServerDiskLogConfigSet` or the function will fail.

The `pszFileName` field of `TServerDiskLogConfig` must be initialized to point to the user allocated and initialized buffer. The `dwFileNameSize` field is not used.

It is recommended that the user invokes `DscServerDiskLogConfigGet` to get the server disk log configuration defaults, updates the pertinent fields, and then invokes `DscServerDiskLogConfigSet` to set the new desired values.

The DAPcell Local or DAPcell server will start a disk logging session when `DscStartAcquiring` is invoked only if the following has been done:
1) Under Windows Control Panel | Data Acquisition Processor | Disk I/O tab | Disk Logging:
   - Set the Default Path to a valid path (or valid paths) on the server PC
   - Set the Permission to Restricted or Normal
   - Select the Save button before closing the dialog, or changes will be lost

   For more information on Default Path and Permission, refer to the DAP Service documentation by selecting the Help button, or going to the main documentation reference on the DAPtools CD.

2) Set a valid filename for pszFileName of TServerDiskLogConfig.
3) Enable DscServerDiskLogEnabledSet.

The disk logging session will end when DscStopAcquiring is invoked.

The Accel32 server does not support server disk logging sessions.

See Also
DscServerDiskLogConfigGet, DscServerDiskLogBytes,
DscServerDiskLogEnabledSet, TServerDiskLogConfig
DscServerDiskLogEnabledGet

The `DscServerDiskLogEnabledGet` function gets the state of the server disk log option.

```c
int __stdcall DscServerDiskLogEnabledGet(
    HDSC hDsc,           // iDSC board handle.
);
```

**Parameters**

- `hDsc`
  - Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is 0 for disabled and 1 for enabled. If the function fails, the return value is -1.

**Remarks**

`DscServerDiskLogEnabledGet` returns the state of the server disk log option. If `DscServerDiskLogEnabledGet` is 0, the server will not log any data to disk when `DscStartAcquiring` is invoked. If `DscServerDiskLogEnabledGet` is 1, the server will log data to disk when `DscStartAcquiring` is invoked.

**See Also**

`DscServerDiskLogEnabledSet`, `DscServerDiskLogConfigSet`
**DscServerDiskLogEnabledSet**

The `DscServerDiskLogEnabledSet` function sets the state of the server disk log option.

```c
int __stdcall DscServerDiskLogEnabledSet(
    HDSC hDsc,         // iDSC board handle.
    int iSdlEnabled   // Server disk log state.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iSdlEnabled`  
  Server disk log state.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscServerDiskLogEnabledSet` sets the state of the server disk log option. If `iSdlEnabled` is 1, the server will log data to disk when `DscStartAcquiring` is invoked. If `iSdlEnabled` is 0, the server will not log any data to disk when `DscStartAcquiring` is invoked.

**See Also**

`DscServerDiskLogEnabledGet`, `DscServerDiskLogConfigSet`
**DscServerDiskLogFileNameGet**

The `DscServerDiskLogFileNameGet` function gets the file name used for server disk logging.

```c
int __stdcall DscServerDiskLogFileNameGet(
    HDSC hDsc, // iDSC board handle.
    int iSize,  // Size of pszFileName buffer.
    const char *pszFileName // Pointer to buffer of characters.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **iSize**
  - Size of the `pszFileName` buffer that stores the file name. The buffer is allocated by the application.

- **pszFileName**
  - Pointer to a buffer that stores the file name. The buffer is allocated by the application.

**Return Values**

If the function succeeds, the return value is the length of the string. If the function fails, the return value is -1.

**Remarks**

`DscServerDiskLogFileNameGet` returns the name of the primary disk log file and the name of a possible mirror disk log file, if mirror logging is enabled. The default is an empty string.

**See Also**

`DscServerDiskLogFileNameSet`, `TServerDiskLogConfig`
DscServerDiskLogFileNameSet

The **DscServerDiskLogFileNameSet** function sets the file name to use for server disk logging.

```c
int __stdcall DscServerDiskLogFileNameSet(
    HDSC hDsc,  // iDSC board handle.
    const char *pszFileName  // File name.
);
```

**Parameters**

- **hDsc**  
  Handle of the iDSC board.

- **pszFileName**  
  Pointer to a file name to use for server disk logging.

**Return Values**

- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscServerDiskLogFileNameSet** specifies the name of the primary disk log file and the name of a possible mirror disk log file, if mirror logging is enabled. The default is an empty string. You can also set the file name through **pszFileName** in **TServerDiskLogConfig**.

Mirror logging is enabled by selecting the **DscDlfMirrorLog** flag of **dwFlags** in **TServerDiskLogConfig**. Multiple file names are separated by semi-colons. Currently, only one mirror file is allowed. Both files must be on the same side of the DAPcell Local/DAPcell service (the PC application side or the iDSC side).

**See Also**

**DscServerDiskLogFileNameGet, TServerDiskLogConfig**
DscSlaveCount

The DscSlaveCount function returns the number of slaves attached to a Master iDSC board. It is only useful in a Master/Slave Configuration.

```c
int __stdcall DscSlaveCount(
    HDSC hDscMaster // Master iDSC board handle.
);
```

Parameters
- **hDscMaster**  
  Handle of the Master iDSC board.

Return Values
- If the function succeeds, the return value is the number of slaves. If the function fails, the return value is -1.
- If there are no slaves attached to the master, the return value is 0. If DscSlaveCount is invoked on a Slave or Normal iDSC board the return value is also 0.

Remarks
- DscSlaveCount should only be called on a Master iDSC board. It returns the slave count, and not the slave index, of a Master iDSC board. For example, if the slave count is 1, then the slave index is 0. The slave index in used in the DscSlaveHandle function.

See Also
- DscMasterGet, DscMasterSet
**DscSlaveHandle**

The `DscSlaveHandle` function returns the handle of a Slave iDSC board that is attached to a Master iDSC board, given a slave index. It is only useful in a Master/Slave Configuration.

```c
HDSC __stdcall DscSlaveHandle(
    HDSC hDscMaster,  // Master iDSC board handle.
    int iSlaveIndex   // Slave index.
);
```

**Parameters**

- **hDscMaster**  
  Handle of the Master iDSC board.

- **iSlaveIndex**  
  Slave index of interest. Valid slave indices are 0 through `DscSlaveCount` - 1.

**Return Values**

If the function succeeds, the return value is the Slave iDSC board handle for the given slave index. If the function fails, the return value is a NULL handle (of value 0).

**Remarks**

`DscSlaveHandle` should only be called on a Master iDSC board. When passed a valid slave index, it returns the corresponding slave handle connected to the Master iDSC board.

Valid slave indices are 0 through `DscSlaveCount` - 1. To get the handles of all slaves attached to a Master iDSC board, the user could use a `for` loop that goes from 0 to `DscSlaveCount` - 1.

The function will fail if `DscSlaveHandle` is invoked on a Slave or Normal iDSC board or if the user passes an invalid slave index to `DscSlaveHandle`.

**See Also**

`DscMasterGet`, `DscMasterSet`
**DscStartAcquiring**

The **DscStartAcquiring** function starts the data acquiring process. Data will start showing up at the PC.

```c
int __stdcall DscStartAcquiring(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**

- **hDsc**
  Handle of the iDSC board.

**Return Values**

- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

- **DscStartAcquiring** starts the sampling process, which causes data to start showing up at the PC. **DscStartAcquiring** will force a **DscCalibrate** if it cannot find the saved calibration values, and a **DscCommandsLoad** if the iDSC board configuration or filter designs have changed. If the iDSC board configuration or filter designs have changed, the user may see a delay of two times **DscGroupDelay** seconds.

- Once **DscStartAcquiring** is invoked, the user can use **DscBufferGetEx** to read blocks of data into buffers.

**See Also**

- **DscBufferGetEx, DscCalibrate, DscCommandsLoad, DscStopAcquiring**
**DscStopAcquiring**

The *DscStopAcquiring* function stops the data acquiring process. Data will stop showing up at the PC.

```c
int __stdcall DscStopAcquiring( 
    HDSC hDsc // iDSC board handle.
);
```

### Parameters

- **hDsc**
  - Handle of the iDSC board.

### Return Values

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

### Remarks

*DscStopAcquiring* stops the sampling process. It is called after
*DscStartAcquiring* to stop data from showing up at the PC.

### See Also

*DscStartAcquiring*
**DscStructPrepare**

The **DscStructPrepare** function prepares structures for use with functions.

```c
int __stdcall DscStructPrepare(
    void *pvStruct,            // Address of structure to initialize.
    unsigned long ulSize      // Size of structure.
);
```

**Parameters**

- **pvStruct**
  Pointer to structure.

- **ulSize**
  Size of structure.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscStructPrepare** initializes the `iInfoSize` field and zeroes out all other fields of structures. It should be used with structures that have an `iInfoSize` field before the other fields of the structures are initialized. Since **DscStructPrepare** initializes the `iInfoSize` field, there is no need to initialize it separately.

**See Also**

- `TBufferGetEx`
- `TFilterParam`
- `TXbPinConfig`
DscSystemErrorProcess

The DscSystemErrorProcess function processes error messages from the iDSC.

```c
int __stdcall DscSystemErrorProcess(
    HDSC hDsc // iDSC board handle.
);
```

Parameters

hDsc
Handle of the iDSC board.

Return Values

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remarks

DscSystemErrorProcess processes error messages from the iDSC if they exist, and then invokes DscOnSystemErrorSet. DscSystemErrorProcess is automatically invoked by DscBufferGetEx. The user can also call DscSystemErrorProcess and invoke DscOnSystemErrorSet if errors from the iDSC are suspected.

See Also

DscOnSystemErrorSet
**DscTcEnabledCount**

The `DscTcEnabledCount` function returns the number of enabled timing channels on the iDSC board.

```c
int __stdcall DscTcEnabledCount(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**

- `hDsc` Handle of the iDSC board.

**Return Values**

- If the function succeeds, the return value is the number of enabled timing channels.
- If the function fails, the return value is -1.

**Remarks**

`DscTcEnabledCount` returns the number of enabled timing channels on the iDSC board. It is related to the `DscTcEnabledGet` and `DscTcEnabledSet` functions. Valid values for the enabled timing channel count are in the range zero to two.

**See Also**

`DscTcEnabledGet`, `DscTcEnabledSet`
**DscTcEnabledGet**

The **DscTcEnabledGet** function gets the enabled timing channels on the iDSC board.

```c
int __stdcall DscTcEnabledGet(
    HDSC hDsc
    // iDSC board handle.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

**Return Values**

- If the function succeeds, the return value is the enabled timing channels. If the function fails, the return value is -1.

**Remarks**

**DscTcEnabledGet** returns the enabled timing channels as an integer value. Only the lowest three bits (bit 0 through bit 2) are used since there are a total of three timing channel options, **Tc0** (Timing Channel 0), **Tc1** (Timing Channel 1), and **TcWidth32** (32-bit width). **Tc0** is bit 0, **Tc1** is bit 1, and **TcWidth32** is bit 2 of the returned integer value.

When the timing channels are enabled, the corresponding bits are set. When the timing channels are disabled, the corresponding bits are cleared. To find out the number of enabled timing channels, use the **DscTcEnabledCount** function.

**See Also**

**DscTcEnabledCount**, **DscTcEnabledSet**, **DscTcMaximum**, **DscTcWidth**
**DscTcEnabledSet**

The **DscTcEnabledSet** function sets the enabled timing channels on the iDSC board.

```
int __stdcall DscTcEnabledSet(
    HDSC hDsc,       // iDSC board handle.
    int iTcEnabled   // Timing channels to enable.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **iTcEnabled**
  - Timing channels to enable.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscTcEnabledSet** enables or disables the timing channels using the **iTcEnabled** integer value. Only the lowest three bits (bit 0 through bit 2) are used since there are a total of three timing channel options, **Tc0** (Timing Channel 0), **Tc1** (Timing Channel 1), and **TcWidth32** (32-bit width). **Tc0** is bit 0, **Tc1** is bit 1, and **TcWidth32** is bit 2 of **iTcEnabled**.

When the timing channels are enabled, the corresponding bits are set. When the timing channels are disabled, the corresponding bits are cleared. To find out the number of enabled timing channels, use the **DscTcEnabledCount** function.

**Tc0** enables Timing Channel 0 and **Tc1** enables Timing Channel 1. **TcWidth32** forces the timing channel width to be 4 bytes (32-bits) regardless of the sample rate. If **TcWidth32** is not enabled, the timing channel width can be either 2 bytes or 4 bytes depending on the sample rate. To find out the actual timing channel width, use the **DscTcWidth** function.

The constants below simplify enabling the timing channels.

- **DscTc_Tc0**
- **DscTc_Tc1**
- **DscTc_TcWidth32**
The following example is used to enable Timing Channel 1 with 4 bytes for the timing channel width.

```c
DscTcEnabledSet(hDsc, DscTc_Tc1|DscTc_TcWidth32);
```

**See Also**

-DscTcEnabledCount, DscTcEnabledGet, DscTcMaximum, DscTcWidth
DscTcMaximum

The DscTcMaximum function returns the maximum timing channel value based on the sample rate.

```c
int __stdcall DscTcMaximum(
    HDSC hDsc  // iDSC board handle.
);
```

Parameters

hDsc
Handle of the iDSC board.

Return Values

If the function succeeds, the return value is the maximum timing channel value. If the function fails, the return value is 0.

Remarks

DscTcMaximum returns the maximum timing channel value which is dependent on the sample rate. DscTcMaximum can range from 128 to 2457600. The wide range of DscTcMaximum forces DscTcWidth to be either 2 bytes or 4 bytes.

See Also

DscTcEnabledGet, DscTcEnabledSet, DscTcWidth
**DscTcWidth**

The **DscTcWidth** function returns the width, in bytes, of timing channel values.

```c
int __stdcall DscTcWidth(HDSC hDsc); // iDSC board handle.
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

**Return Values**

- If the function succeeds, the return value is the timing channel width. If the function fails, the return value is 0.

**Remarks**

- **DscTcWidth** returns the width, in bytes, of timing channel values. It returns either 2 for 16-bit values or 4 for 32-bit values.

- **DscTcWidth** is dependent on the sample rate. The timing channel width for sample rates 8 s/s to 600 s/s are always 4 bytes. The timing channel width for sample rates 640 s/s to 153600 s/s are 2 bytes by default. The timing channel width for these higher sample rates can be forced to 4 bytes by setting **TcWidth32** in the **DscTcEnabledSet** function.

**See Also**

- **DscTcMaximum**, **DscTcEnabledGet**, **DscTcEnabledSet**
DscTransferFunctionGet

The DscTransferFunctionGet function is used to plot the filter response.

```c
int __stdcall DscTransferFunctionGet(
    HDSC hDsc,          // iDSC board handle.
    int iFilterIndex,   // Filter index.
    int iLength,        // Number of data points.
    double *pdBuffer    // Buffer to receive data.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iFilterIndex`  
  Filter index of interest. Valid filter indices are 0 through 7.

- `iLength`  
  Number of data points to return.

- `pdBuffer`  
  Buffer for storing the transfer function data points. The buffer must be large enough to accommodate the requested `iLength` data points.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

DscTransferFunctionGet plots the filter responses of the filter designs. There can be up to eight filter response plots since there are up to eight filter designs.

**See Also**

DscUnitStepLengthGet, DscUnitStepGet
The **DscUnitStepGet** function plots the unit step responses of the filter designs.

```c
int __stdcall DscUnitStepGet(
    HDSC hDsc, // iDSC board handle.
    int iFilterIndex, // Filter index.
    int iLength, // Number of data points.
    double *pdBuffer // Buffer to receive data.
);
```

**Parameters**
- **hDsc**
  - Handle of the iDSC board.
- **iFilterIndex**
  - Filter index of interest. Valid filter indices are 0 through 7.
- **iLength**
  - Number of data points to return.
- **pdBuffer**
  - Buffer for storing the unit step data points. The buffer must be large enough to accommodate the requested `iLength` data points.

**Return Values**
If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**
- **DscUnitStepGet** plots the unit step responses of the filter designs. The **DscUnitStepLengthGet** function should be used to determine the `iLength` parameter. There can be up to eight unit step response plots since there are up to eight filter designs.

**See Also**
- **DscTransferFunctionGet, DscUnitStepLengthGet**
DscUnitStepLengthGet

The **DscUnitStepLengthGet** function returns the number of data points in the unit step response.

```c
int __stdcall DscUnitStepLengthGet(
    HDSC hDsc,       // iDSC board handle.
    int iFilterIndex  // Filter index.
);
```

**Parameters**
- **hDsc**
  - Handle of the iDSC board.
- **iFilterIndex**
  - Filter index of interest. Valid filter indices are 0 through 7.

**Return Values**
- If the function succeeds, the return value is the number of data points. If the function fails, the return value is 0.

**Remarks**
- **DscUnitStepLengthGet** returns the number of data points in the unit step response. The function should be used to determine the `iLength` parameter of **DscUnitStepGet**.

**See Also**
- **DscTransferFunctionGet**, **DscUnitStepGet**
DscXbCalibrate

The DscXbCalibrate function performs calibration on the external board.

```c
int __stdcall DscXbCalibrate(
    HDSC hDsc  // iDSC board handle.
    int iSampleRate  // Sample rate to use for calibration.
);```

Parameters

- **hDsc**
  Handle of the iDSC board.

- **iSampleRate**
  Sample rate to use for external board calibration. A sample rate of 100 s/s is recommended.

Return Values

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remarks

DscXbCalibrate is used to calibrate the external board. It is advisable to not exceed a sample rate of 1024 s/s or the calibration values may be unstable. DscXbCalibrate works only if DscXbEnabledGet is true, otherwise a failure occurs.

A higher sample rate allows DscXbCalibrate to execute faster but the calibration values are less accurate. A lower sample rate executes slower but the calibration values are more accurate. The recommended value of 100 s/s works well.

DscXbCalibrate is not automatically invoked if the external board calibration values are not found. The user has to explicitly call DscXbCalibrate to calibrate the external board.

See Also

DscXbEnabledGet, DscXbEnabledSet
DscXbEnabledGet

The DscXbEnabledGet function gets the state of the external board.

```c
int __stdcall DscXbEnabledGet(
    HDSC hDsc // iDSC board handle.
);```

Parameters

- `hDsc` Handle of the iDSC board.

Return Values

If the function succeeds, the return value is 0 for disabled and 1 for enabled. If the function fails, the return value is -1.

Remarks

DscXbEnabledGet returns the state of the external board, whether enabled or disabled. If DscXbEnabledGet is 0, the external board is disabled and not visible in DscConfigDialogShow. If DscXbEnabledGet is 1, the external board is enabled and visible in DscConfigDialogShow.

See Also

DscXbEnabledSet
DscXbEnabledSet

The `DscXbEnabledSet` function sets the external board state.

```c
int __stdcall DscXbEnabledSet(
    HDSC hDsc, // iDSC board handle.
    int iXbEnabled // External board state.
);
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

- **iXbEnabled**
  - External board state, whether enabled or disabled.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscXbEnabledSet` sets the external board state, whether enabled or disabled. If `iXbEnabled` is 0, the external board is disabled and not visible in `DscConfigDialogShow`. If `iXbEnabled` is 1, the external board is enabled and visible in `DscConfigDialogShow`.

To access the external board, `iXbEnabled` must be 1 so that the configuration information is sent to the external board. Also `DscXbCalibrate` will work only if `iXbEnabled` is true.

**See Also**

`DscXbEnabledGet`, `DscXbCalibrate`
**DscXbPinConfigGet**

The `DscXbPinConfigGet` function gets the external board pin configuration associated with a pin index. The pin configuration includes the input type, input range, input offset, input offset range, and output excitation.

```c
int __stdcall DscXbPinConfigGet(
    HDSC hDsc,          // iDSC board handle.
    int iPinIndex,      // Pin index.
    TXbPinConfig *pXbPinConfig   // Address of structure.
);
```

**Parameters**

- **hDsc**
  Handle of the iDSC board.

- **iPinIndex**
  Pin index of interest. Valid pin indices are 0 through 7.

- **pXbPinConfig**
  Pointer to a `TXbPinConfig` structure that receives the pin configuration.
  `TXbPinConfig` must be initialized using `DscStructPrepare`.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscXbPinConfigGet` returns the pin configuration for an associated pin index in the `TXbPinConfig` structure. `TXbPinConfig` must be initialized using `DscStructPrepare` before invoking `DscXbPinConfigGet` or the function will fail.

Valid pin indices are 0 through 7. If the pin index is invalid, the function will fail.

**See Also**

- `DscConfigDialogShow`, `DscXbPinConfigSet`, `TXbPinConfig`
The `DscXbPinConfigSet` function sets the pin configuration associated with a pin index. The pin configuration includes the input type, input range, input offset, input offset range, and output excitation.

```c
int __stdcall DscXbPinConfigSet(
    HDSC hDsc, // iDSC board handle.
    int iPinIndex, // Pin index.
    const TXbPinConfig *pXbPinConfig // Address of structure.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iPinIndex`  
  Pin index of interest. Valid pin indices are 0 through 7.

- `pXbPinConfig`  
  Pointer to a `TXbPinConfig` structure that passes the pin configuration. `TXbPinConfig` must be initialized using `DscStructPrepare`.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscXbPinConfigSet` modifies the pin configuration for an associated pin index through the `TXbPinConfig` structure. `TXbPinConfig` must be initialized using `DscStructPrepare` before invoking `DscXbPinConfigSet` or the function will fail.

The `iInputType`, `fInputRange`, `fInputOffset`, and `fOutputExcitation` members must be set to the new desired values. `fInputOffsetRange` cannot be set since it is a read only property. Note that the pin configuration can be graphically set in `DscConfigDialogShow`.

Valid pin indices are 0 through 7. If the pin index is invalid, the function will fail.
See Also
DscConfigDialogShow, DscXbPinConfigGet, TXbPinConfig
The **DscGroupAddOne** function adds one iDSC to the iDSC group.

```c
int __stdcall DscGroupAddOne(
    HDSCGROUP hDscGroup, // iDSC group handle.
);
```

**Parameters**
- **hDscGroup**
  - Handle of the group of iDSC boards.

**Return Values**
- If the function succeeds, the return value is the index of the added iDSC. If the function fails, the return value is -1.

**Remarks**
- **DscGroupAddOne** adds one iDSC to the iDSC group and returns the index of the added iDSC. The index of the iDSC is the **DscGroupCount** minus one. The maximum number of iDSCs in the iDSC group is 64 which means that the maximum index is 63. When **DscGroupAddOne** is invoked, the **DscGroupCount** increases by one.

  As an example, **DscGroupAddOne** returns 3. Therefore, the newly added iDSC index is 3 and the **DscGroupCount** is 4.

**See Also**
- **DscGroupDeleteOne, DscGroupCount**
**DscGroupConfigDialogShow**

The **DscGroupConfigDialogShow** function displays modal dialog screens for graphical configuration of multiple iDSC boards.

```c
int __stdcall DscGroupConfigDialogShow(
    HDSCGROUP hDscGroup, // iDSC group handle.
);
```

**Parameters**

- **hDscGroup**
  - Handle of the group of iDSC boards.

**Return Values**

If the user selects the OK button, the return value is IDOK (of value 1). If the user selects the Cancel button, the return value is IDCANCEL (of value 2). If the function fails, the return value is 0.

**Remarks**

**DscGroupConfigDialogShow** simplifies the configuration of multiple iDSC boards with a graphical interface. When the user expands the iDSC Boards tree, there are options for changing the Address and Mode. When the user selects the DSC name with the right click button, there are several more options like External Board Calibrate, External Board Enable, Raw Data, Remote Master, Server Disk Log, and Copy and Paste. The Copy and Paste feature allows the user to copy the entire configuration of an iDSC board to another iDSC board.

For each iDSC board, the **Input Screen** provides a quick method for selecting the sample rate, selecting the input range, mapping the input pins to selected filter designs, and enabling or disabling the input pins.

The **Filter Design Screen** allows easy manipulation of filter parameters like sharpness, low cutoff frequency, low cutoff slope, high cutoff frequency, high cutoff slope, and attenuation, either by entering a valid number or by adjusting a slider. When the user selects the right click button, there are several more options like Crosshair Track, Y Display (linear, linear zoom, log, log zoom, and unit step), Defaults Load, Copy, and Paste. The Copy (Ctrl-C) and Paste (Ctrl-V) feature copies the filter parameters from one filter design tab to another.
The figure below displays the configuration of a group of four iDSC boards. DSC0 and DSC1 are configured as Independent, while DSC2 is the Master of DSC3.

See Also
DscGroupAddOne, DscGroupDeleteOne, DscGroupCount
DscGroupConfigRead

The DscGroupConfigRead function reads iDSC group information from memory.

```c
int __stdcall DscGroupConfigRead(
    HDSCGROUP hDscGroup,  // iDSC group handle.
    int iSize,            // Size of memory allocated.
    void *pvBuffer        // Pointer to memory buffer.
);
```

Parameters

- **hDscGroup**
  - Handle of the group of iDSC boards.

- **iSize**
  - Size of the memory buffer for the iDSC group information. The memory buffer is allocated by the application.

- **pvBuffer**
  - Pointer to the memory buffer that stores the iDSC group information. The memory buffer is allocated by the application.

Return Values

- If the function succeeds, the return value is the number of bytes read from memory.
- If the function fails, the return value is 0.

Remarks

DscGroupConfigRead reads iDSC group information in binary format from memory. The information in memory is probably retrieved from a disk file.

The iDSC group information read includes the number of iDSCs boards, address, mode, sample rate, input range, input pin to filter mappings, enabled input pins, and details of the filter designs.

iSize is the value that was previously used as the iSize parameter of DscGroupConfigWrite when the iDSC group information was stored to memory. The user cannot use DscGroupConfigWriteSize for iSize in DscGroupConfigRead because it will not return the correct size. DscGroupConfigWriteSize is provided only for the DscGroupConfigWrite operation.
See Also

DscGroupConfigWrite, DscGroupConfigWriteSize
DscGroupConfigWrite

The DscGroupConfigWrite function writes iDSC group information to memory.

```c
int __stdcall DscGroupConfigWrite(
    HDSCGROUP hDscGroup,  // iDSC group handle.
    int iSize,             // Size of memory allocated.
    void *pvBuffer        // Pointer to memory buffer.
);
```

Parameters

- **hDscGroup**
  - Handle of the group of iDSC boards.

- **iSize**
  - Size of the memory buffer for the iDSC group information. The memory buffer is allocated by the application.

- **pvBuffer**
  - Pointer to the memory buffer that stores the iDSC group information. The memory buffer is allocated by the application.

Return Values

- If the function succeeds, the return value is the number of bytes written to memory.
- If the function fails, the return value is 0.

Remarks

DscGroupConfigWrite writes iDSC group information in binary format to memory. The information in memory can then be written to a disk file.

The iDSC group information written includes the number of iDSCs boards, address, mode, sample rate, input range, input pin to filter mappings, enabled input pins, and details of the filter designs.

*iSize* is the size of the memory buffer allocated and must be at least DscGroupConfigWriteSize. *iSize* should be stored by the program to be used later as the *iSize* parameter of DscGroupConfigRead.
See Also

DscGroupConfigRead, DscGroupConfigWriteSize
**DscGroupConfigWriteSize**

The `DscGroupConfigWriteSize` function returns the minimum number of bytes to allocate for `DscGroupConfigWrite`.

```c
int __stdcall DscGroupConfigWriteSize(
    HDSCGROUP hDscGroup,  // iDSC group handle.
);
```

**Parameters**
- `hDscGroup` Handle of the group of iDSC boards.

**Return Values**
- If the function succeeds, the return value is the minimum number of bytes to allocate. If the function fails, the return value is 0.

**Remarks**
- `DscGroupConfigWriteSize` must be used to determine the minimum number of bytes to allocate for the `iSize` parameter of `DscGroupConfigWrite`. It cannot be used as the `iSize` parameter of `DscGroupConfigRead`.

**See Also**
- `DscGroupConfigWrite`, `DscGroupConfigRead`
**DscGroupCount**

The **DscGroupCount** function returns the number of iDSCs in the iDSC group.

```c
int __stdcall DscGroupCount(
    HDSCGROUP hDscGroup, // iDSC group handle.
);
```

**Parameters**
- **hDscGroup**
  - Handle of the group of iDSC boards.

**Return Values**
- If the function succeeds, the return value is the number of iDSCs in the iDSC group. If the function fails, the return value is -1.

**Remarks**
- **DscGroupCount** returns the number of iDSCs in the iDSC group. **DscGroupCount** increases by one when **DscGroupAddOne** is invoked and decreases by one when **DscGroupDeleteOne** is invoked. The maximum number of iDSCs in the iDSC group is 64.

**See Also**
- **DscGroupAddOne**, **DscGroupDeleteOne**
**DscGroupDeleteOne**

The **DscGroupDeleteOne** function deletes one iDSC from the iDSC group.

```c
int __stdcall DscGroupDeleteOne(
   HDSCGROUP hDscGroup,       // iDSC group handle.
);
```

**Parameters**

- **hDscGroup**
  - Handle of the group of iDSC boards.

**Return Values**

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

**DscGroupDeleteOne** deletes one iDSC from the iDSC group. The maximum number of iDSCs in the iDSC group is 64. When **DscGroupDeleteOne** is invoked, the **DscGroupCount** decreases by one.

**See Also**

**DscGroupAddOne, DscGroupCount**
The **DscGroupDsc** function allows access to a specific iDSC board using the iDSC index.

```c
HDSC __stdcall DscGroupDsc(
    HDSCGROUP hDscGroup,      // iDSC group handle.
    int iDscIndex,            // iDSC board index.
);
```

**Parameters**
- **hDscGroup**
  - Handle of the group of iDSC boards.
- **iDscIndex**
  - iDSC board index of interest. Valid iDSC board indices are 0 through **DscGroupCount** - 1.

**Return Values**
- If the function succeeds, the return value is the iDSC board handle for the given iDSC board index. If the function fails, the return value is a NULL handle (of value 0).
- The iDSC board handle should not be stored, instead **DscGroupDsc** should be called repeatedly to access a specific iDSC board.

**Remarks**
- **DscGroupDsc** allows access to a specific iDSC board using the iDSC board index. Valid iDSC board indices are 0 through **DscGroupCount** - 1.
- The iDSC board handle should not be stored because it is destroyed and recreated in functions like **DscGroupConfigDialogShow**, **DscGroupDeleteOne**, **DscGroupAddOne**, etc. **DscGroupDsc** should be called repeatedly to access a specific iDSC board.

**See Also**
- **DscGroupConfigDialogShow**, **DscGroupCount**
DscGroupHandleClose

The `DscGroupHandleClose` function releases a handle previously opened with `DscGroupHandleOpen`.

```c
int __stdcall DscGroupHandleClose(
    HDSCGROUP hDscGroup,  // iDSC group handle to close.
);
```

**Parameters**

- `hDscGroup`  
  Handle of the group of iDSC boards to close. It must be a handle previously opened by `DscGroupHandleOpen`.

**Return Values**

- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscGroupHandleClose` terminates communication with the iDSC group. `DscGroupHandleClose` is the last function called by any application ready to end communication with the iDSC group. Once the iDSC group communication has ended, the user can no longer access a specific iDSC board through `DscGroupDsc`.

**See Also**

`DscGroupHandleOpen`, `DscGroupDsc`
**DscGroupHandleOpen**

The **DscGroupHandleOpen** function returns a group handle to the target iDSC group.

```c
HDSCGROUP __stdcall DscGroupHandleOpen(
);
```

**Return Values**

If the function succeeds, the return value is an open handle to the specified target. If the function fails, the return value is a NULL handle (of value 0).

**Remarks**

**DscGroupHandleOpen** initiates communication with the iDSC group. **DscGroupHandleOpen** is the first function called by any application ready to begin communication with the iDSC group. It returns a group handle to the target iDSC group.

A group handle is a 32-bit value that references the group of iDSC boards. This handle is used in all **DscGroupXxxx** services to reference the appropriate iDSC group.

Once the iDSC group communication has began, the user can access a specific iDSC board through **DscGroupDsc**.

**See Also**

**DscGroupHandleClose, DscGroupDsc**
Obsolete Interface

The functions listed below are obsolete. They are discussed in greater detail in the following pages.

<table>
<thead>
<tr>
<th>Category</th>
<th>Dsc Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom command services</td>
<td>DscDaplCCDownloadGet</td>
</tr>
<tr>
<td></td>
<td>DscDaplCCDownloadSet</td>
</tr>
<tr>
<td></td>
<td>DscDaplCCListGet</td>
</tr>
<tr>
<td></td>
<td>DscDaplCCListLengthGet</td>
</tr>
<tr>
<td></td>
<td>DscDaplCCListSet</td>
</tr>
<tr>
<td></td>
<td>DscDaplCCStackSizeGet</td>
</tr>
<tr>
<td></td>
<td>DscDaplCCStackSizeSet</td>
</tr>
</tbody>
</table>
DAPL Custom Command Support

Using the DAPL custom command interface

To define DAPL custom commands, the user should use `DscDaplCCListSet`. Each line of the custom command list is a custom command filename. Once a single or a list of custom commands have been defined for the iDSC using `DscDaplCCListSet`, the custom commands will be downloaded to the iDSC when the user invokes `DscCommandsLoad` or `DscStartAcquiring` if the download code in `DscDaplCCDownloadSet` is set to enabled.

If the default stack size of 1024 bytes is not sufficient for the custom commands, the user can change the stack size using `DscDaplCCStackSizeSet`. The stack size should only be changed after the custom commands have been defined using `DscDaplCCListSet` but before `DscCommandsLoad` or `DscStartAcquiring` is invoked.

If the user wants to retrieve the defined custom commands, `DscDaplCCListLengthGet` along with `DscDaplCCListGet` should be used. `DscDaplCCDownloadGet` is used to get the state of the download code. `DscDaplCCStackSizeGet` is used to get the stack size of a custom command.

Below is a C/C++ example:

```c
HdSc hDsc;
hDsc = DscHandleOpen("\\.\Dap0"); // Open iDSC handle.
DscD aplCCListSet(hDsc, // Define two custom commands.
    "c:\\t1.bin\r\n",
    "c:\\t2.bin\r\n");
DscDaplCCStackSizeSet(hDsc, // Set t1.bin stack
    0, 2048); // size to 2048 bytes.
DscDaplCCStackSizeSet(hDsc, // Set t2.bin stack
    1, 2048); // size to 2048 bytes.
DscDaplCCDownloadSet(hDsc, 1); // Enable custom command download.
DscStartAcquiring(hDsc); // Send custom command list to iDSC.
//... (other function calls) ...
```
DscDaplCCDownloadGet

The DscDaplCCDownloadGet function gets the download code that indicates if downloading the DAPL custom command list is enabled or disabled.

```c
int __stdcall DscDaplCCDownloadGet(HDSC hDsc) // iDSC board handle.
```

**Parameters**

- **hDsc**
  - Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the download code. The download code is 0 if downloading the custom command list is disabled and 1 if downloading the custom command list is enabled. If the function fails, the return value is -1.

**Remarks**

DscDaplCCDownloadGet returns the download code that indicates if downloading the DAPL custom command list is enabled or disabled.

If the download code is enabled, the DAPL custom command list is downloaded to the iDSC when the user invokes DscCommandsLoad or DscStartAccquiring. If the download code is disabled, the DAPL custom command list is not downloaded to the iDSC.

The user should only have to download the DAPL custom command list to the iDSC once at startup.

**See Also**

DscDaplCCDownloadSet
The `DscDaplCCDownloadSet` function sets the download code that indicates if downloading the DAPL custom command list is enabled or disabled.

```c
int __stdcall DscDaplCCDownloadSet(
    HDSC hDsc,           // iDSC board handle.
    int iDownload        // Download code.
);
```

**Parameters**

- `hDsc`  
  Handle of the iDSC board.

- `iDownload`  
  Download code. Set it to 0 to disable downloading or 1 to enable downloading.

**Return Values**

- If the function succeeds, the return value is 1. If the function fails, the return value is 0.

**Remarks**

`DscDaplCCDownloadSet` sets the download code that indicates if downloading the DAPL custom command list is enabled or disabled.

If the download code is enabled, the DAPL custom command list is downloaded to the iDSC when the user invokes `DscCommandsLoad` or `DscStartAccquiring`. If the download code is disabled, the DAPL custom command list is not downloaded to the iDSC.

The user should only have to download the DAPL custom command list to the iDSC once at startup.

**See Also**

`DscDaplCCDownloadGet`
**DscDaplCCListGet**

The `DscDaplCCListGet` function gets the DAPL custom command list defined for the iDSC.

```c
int __stdcall DscDaplCCListGet(
    HDSC hDsc,    // iDSC board handle.
    int iSize,    // Size of `pszCCList` buffer.
    const char *pszCCList    // Pointer to buffer of characters.
);
```

### Parameters

- **hDsc**
  - Handle of the iDSC board.

- **iSize**
  - Size of the `pszCCList` buffer for the DAPL custom command list. The buffer is allocated by the application.

- **pszCCList**
  - Pointer to a buffer that stores the DAPL custom command list. The buffer is allocated by the application.

### Return Values

If the function succeeds, the return value is the length of the string. If the function fails, the return value is -1.

### Remarks

`DscDaplCCListGet` returns the DAPL custom command list defined for the iDSC. Each custom command in the list is delimited by a carriage-return and line-feed.

The DAPL custom command list is sent to the iDSC when the user invokes `DscCommandsLoad` or `DscStartAcquiring` if the download code in `DscDaplCCDownloadSet` is enabled.

### See Also

`DscDaplCCListLengthGet`, `DscDaplCCListSet`
**DscDaplICCListLengthGet**

The *DscDaplICCListLengthGet* function gets the length of the DAPL custom command list defined for the iDSC.

```c
int __stdcall DscDaplICCListLengthGet(
    HDSC hDsc // iDSC board handle.
);
```

**Parameters**

- `hDsc`
  - Handle of the iDSC board.

**Return Values**

If the function succeeds, the return value is the length of the buffer for the DAPL custom command list. If the function fails, the return value is 0.

**Remarks**

*DscDaplICCListLengthGet* returns the length of the buffer for the DAPL custom command list defined for the iDSC. This includes an extra character at the end for the null-terminator.

*DscDaplICCListLengthGet* allows the user to determine the size of buffer to allocate when using the *DscDaplICCListGet* function.

**See Also**

*DscDaplICCListGet*, *DscDaplICCListSet*
DscDaplCCListSet

The DscDaplCCListSet function defines the DAPL custom command list for the iDSC.

```c
int __stdcall DscDaplCCListSet(
    HDSC hDsc, // iDSC board handle.
    const char *pszCCList // Custom command list.
);
```

Parameters

- **hDsc**: Handle of the iDSC board.
- **pszCCList**: Pointer to the DAPL custom command list.

Return Values

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remarks

DscDaplCCListSet allows the user to define a DAPL custom command list for the iDSC. Each custom command in the list must be delimited by a carriage-return, line-feed, or both.

The DAPL custom command list is sent to the iDSC when the user invokes DscCommandsLoad or DscStartAcquiring if the download code in DscDaplCCDownloadSet is enabled.

See Also

DscDaplCCListGet
**DscDapICCCStackSizeGet**

The **DscDapICCCStackSizeGet** function gets the stack size of the custom command identified by an index.

```c
int __stdcall DscDapICCCStackSizeGet(
    HDSC hDsc,  // iDSC board handle.
    int iIndex   // Index of custom command.
);
```

**Parameters**
- `hDsc`: Handle of the iDSC board.
- `iIndex`: Index of the custom command. Valid numbers are from 0 to (n-1), where n is the number of custom commands in the list.

**Return Values**
If the function succeeds, the return value is the stack size. If the function fails, the return value is 0. The minimum value allowed for the stack size is 1024 bytes.

**Remarks**
**DscDapICCCStackSizeGet** returns the stack size of the custom command identified by an index from the list of custom commands. If there is only one custom command in the list, the index is 0.

**See Also**
**DscDapICCCStackSizeSet**
The `DscDaplCCStackSizeSet` function sets the stack size of the custom command identified by an index.

```c
int __stdcall DscDaplCCStackSizeSet(  
    HDSC hDsc,  // iDSC board handle.  
    int iIndex, // Index of custom command.  
    int iStackSize  // Stack size.  
    );
```

**Parameters**
- **hDsc**
  - Handle of the iDSC board.
- **iIndex**
  - Index of the custom command. Valid numbers are from 0 to (n-1), where n is the number of custom commands in the list.
- **iStackSize**
  - Stack size to set. The default is 1024.

**Return Values**
- If the function succeeds, the return value is 1. If the function fails, the return value is 0. The minimum value allowed for the stack size is 1024 bytes.

**Remarks**
- `DscDaplCCStackSizeSet` sets the stack size of the custom command identified by an index from the list of custom commands. If there is only one custom command in the list, the index is 0.
- If the required stack size of the custom command is larger than 1024 bytes, the stack size must be set after the DAPL custom command list has been defined using `DscDaplCCListSet`. The DAPL custom command list is then sent to the iDSC with the new stack size when the user invokes `DscCommandsLoad` or `DscStartAcquiring` if the download code in `DscDaplCCDownloadSet` is enabled.
See Also
DscDaplCCStackSizeGet
The Digital Signal Conditioning Board Component (DSCC) Programmer’s Interface provides the link between an application and the iDSC board.

The DSCC interface supports the Delphi 5, Delphi 6, Delphi 7, C++Builder 5, and C++Builder 6 development environments. When the DSC Component Library is installed, the Dsc and DscGroup components will appear on the Microstar tab with the label DSC and DSCs. Installing the Component Library describes installing the DSC Component Library in detail.

The Object and Type Summary provides a summary of all of the objects and types. The Property, Method, and Event Summary provides a summary of all of the supported properties, methods, and events.
DSCC Interface Examples

There are several examples located in the `<InstallDir>\Examples\Pascal` directory for Delphi which demonstrate the use of the DSCC interface. Before running the examples, verify that the iDSC board and iDSC board software are properly installed by running DSCview. Exit DSCview before running the DSCC interface examples.

**BinLog.dpr**

BinLog.dpr shows how to log binary data to disk. It allows the user to design filters and configure the iDSC board using the `ConfigDialogShow` method. When the iDSC board is started, the selected log file is opened and data logging starts. When the iDSC board is stopped, the selected log file is closed and data logging stops.

**Dvm.dpr**

Dvm.dpr shows how to display voltage measurements. It transfers blocks of data from the iDSC board into a data buffer. It displays the 0th index from the data buffer, assuming a 5V input signal.

**Graph.dpr**

Graph.dpr shows how to display data in a graph. It allows the user to design filters and configure the iDSC board using the `ConfigDialogShow` method. When the iDSC board is started, graphing starts. When the iDSC board is stopped, graphing stops.
Installing the Component Library

The DSC Component Library supports the Delphi and C++Builder development environments. It consists of two components; the Dsc and DscGroup components. The Dsc component configures a single DSC and the DscGroup component configures multiple DSCs.

Before you install the DSC Component Library, make sure that you have installed the Accel32/DAPcell server. The DAPIO32.DLL that is shipped with the Accel32/DAPcell server is needed for the DSC Component Library installation.

Delphi 5, Delphi 6, Delphi 7

The following steps outline the process of incorporating the DSC Component Library into Delphi 5, Delphi 6, and Delphi 7. It involves installing the design time package and configuring the search path.

If you build with run time packages, copy DscLibXX.Xpl to your application directory.

A. Installing the design time package

1) Select the Component|Install Packages... menu item.

2) Select Add..., enter <InstallDir>\Comp\DelphiX\DscLibXX.Xpl as the file name, and select Open.

<InstallDir> is the installation directory for the iDSC software.

\DelphiX\DscLibXX.Xpl is
\Delphi5\DscLib50.bpl for Delphi 5,
\Delphi6\DscLib60.bpl for Delphi 6,
\Delphi7\DscLib70.bpl for Delphi 7.

3) Select OK.

4) The Dsc and DscGroup components are available on the Microstar tab of the component palette with the label DSC and DSCs.

B. Configuring the search path
1) Select the **Tools**|**Environment Options...** menu item.

2) Select the **Library** tab, add `<InstallDir>`\Comp\DelphiX to the library path, and select **OK**.
C++Builder 5, C++Builder 6

The following steps outline the process of incorporating the DSC Component Library into C++Builder 5 and C++Builder 6. It involves installing the design time package and configuring the search path.

If you build with run time packages, copy DscLibXX.bpl to your application directory.

A. Installing the design time package

1) Select the Component|Install Packages... menu item.

2) Select Add..., enter <InstallDir>\Comp\CBuildX\DscLibXX.bpl as the file name, and select Open.

   <InstallDir> is the installation directory for the iDSC software.

   \CBuildX\DscLibXX.bpl is
   \CBuild5\DscLib50.bpl for C++Builder 5
   \CBuild6\DscLib60.bpl for C++Builder 6.

3) Select OK.

4) The Dsc and DscGroup components are available on the Microstar tab of the component palette with the label DSC and DSCs.

B. Configuring the search path

1) Select the File|Close All menu item if any projects are opened.

2) Select the Tools|Environment Options... menu item.

3) Select the Library tab, add <InstallDir>\Comp\CBuildX to the library path, and select OK.
Creating a DSCC Interface Application

Before you create a DSCC Interface application, make sure that you have installed the Accel32/DAPcell server. The DAPIO32.DLL that is shipped with the Accel32/DAPcell server is needed for any application that uses the DSCC Interface.

The following steps describe the functions required for creating a simple application with the DSCC interface using the Dsc component.

1) Select the Dsc component from the Microstar tab of the Component Palette and drop the component onto the application form.

2) Configure the iDSC board at design time by selecting the ConfigDialog property from the Object Inspector. This allows the user to set the sample rate, enable/disable channels, design filters, etc. through a graphical interface.

3) Start data acquisition by invoking the StartAcquiring method.

4) Get binary data from the iDSC board by invoking the BufferGetEx method.

5) Stop data acquisition by invoking the StopAcquiring method.

The steps above describe a complete application. To configure the iDSC board at run time, the user must invoke the ConfigDialogShow method.

More complicated applications that use DAPL text and DAPL custom commands are also available with the DSCIO interface.
Universal Naming Convention

DSCC addresses the iDSC board using the Universal Naming Convention (UNC). A UNC name consists of two parts, the machine name and the iDSC board name. A UNC name begins with two backslashes, and the parts of the name are separated by a single backslash as shown below.

```
\<Machine name>\<iDSC board name>
```

A local machine is denoted by a period. A remote machine is represented by its unique network machine name. Only the DAPcell implementation of the iDSC board supports remote machine names. All other implementations support only local machine names.

iDSC board names are predefined as Dap0, Dap1, ..., Dap(N-1) where N is the number of iDSC boards installed on the system. The maximum value for N is 14.

The UNC name is used with the Address property of the DSCC interface. Using `\.Dap0` as an example, ```\``` denotes the local machine and `\Dap0` denotes the name of the iDSC board. If connected to a remote machine through DAPcell named PC45 that contains Dap0, then Address is `\PC45\Dap0`. 
Master/Slave Configuration

To synchronize multiple iDSC boards on a system, you must designate one iDSC board as a master unit and the other iDSC boards as slave units. From iDSC board software, use the Master property to setup the Master/Slave Configuration. From iDSC board hardware, use a special cable to setup the Master/Slave Configuration.

Since iDSC boards share a sampling clock, the special cable distributes the sampling clock. The Synchronization Connector is described in the Hardware Architecture chapters of this document.

Apart from connecting the cable, one iDSC board has to be configured as a Master iDSC board by using the Master property. Basically, setting up a Master or Slave iDSC board means associating a slave to a master through this property.

The supported properties and methods for Master/Slave Configuration are listed below.

Master property
OperateMode property
RemoteMaster property
SlaveCount property
SlaveName method

The master and slave units must use the same effective sample rate. Therefore, the sample rate should only be changed on a Master iDSC board using the SampleRate property. Changing the sample rate on a Slave iDSC board will have no effect. The Slave iDSC board will continue using the sample rate of its master.

If two independent iDSC boards are running at different sample rates, and the user decides to set up the iDSC boards in a Master/Slave configuration, the Slave iDSC board will use the sample rate of the Master iDSC board.

In a Master/Slave Configuration, the system services listed below should only be invoked on a Master iDSC board. If these methods are invoked on a Slave iDSC board, there will be an exception.

Calibrate method
CommandsLoad method
StartAcquiring method
StopAcquiring method
XbCalibrate method
ConfigDialogShow of TDscGroup allows easy graphical configuration of masters and slaves.
DAPL Support

Writing DAPL

The DSC system writes the filtered data to pipe pDscData for use in DAPL. Pipe pDscData has interleaved data only from the enabled channels. Before the user can perform further processing on the data, the SEPARATE command is recommended for separating the data from pipe pDscData into the correct number of pipes. If the user only has five channels enabled out of the maximum of eight, the user should separate the interleaved data into five pipes. Once the user has completed processing the data, the results must be merged to $BINOUT.

The restricted DAPL interface requires that input and output procedures, and START and STOP commands, not be included in the DAPL listing. The iDSC can only be started and stopped using the StartAcquiring and StopAcquiring methods.

The example below shows the form of the first few lines in DAPL. Pipe pDscData has interleaved data from five enabled channels.

    PIPES P0, P1, P2, P3, P4
    PDEF A
    SEPARATE(pDscData, P0, P1, P2, P3, P4)
    ...
    ...(further processing)
    ...
    ...(must write output to $BINOUT)
    END

P0, P1, P2, P3, P4 are the number of enabled channels, which is five in this example.

Using the DAPL Interface

To write custom DAPL text, the user should use the DaplText property. Once the DAPL text has been defined for the iDSC using DaplText, the DAPL text will be sent to the iDSC when the user invokes CommandsLoad or StartAcquiring.

If the user wants to change the already defined DAPL text, the DaplText property must be updated with the new DAPL text followed by either a CommandsLoad or StartAcquiring to activate the new DAPL text.
If the user wants to retrieve the defined DAPL text, invoke the `DaplText` property.

Below is a Delphi example, where `Dsc1` of type `TDsc` is already on the form:

```delphi
Dsc1.DaplText.Text := 'PIPES P0, P1, P2, P3, P4' +
   'PIPES R0, R1, R2, R3, R4' +
   'PDEF A' +
   '  SEPARATE(pDscData, P0, P1, P2, P3, P4)' +
   '  FFT(5, 9, 4, P0, R0)' +
   '  FFT(5, 9, 4, P1, R1)' +
   '  FFT(5, 9, 4, P2, R2)' +
   '  FFT(5, 9, 4, P3, R3)' +
   '  FFT(5, 9, 4, P4, R4)' +
   '  MERGE(R0, R1, R2, R3, R4, $BINOUT)' +
   '  END';

Dsc1.StartAcquiring; // Send DAPL text.
```

//...(other function calls) ...
```
## Object and Type Summary

Special objects and types are used to define some properties, methods, and events of the DSCC interface. The special objects and types, and the properties and methods they support are displayed below.

<table>
<thead>
<tr>
<th>Object</th>
<th>Supports</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TDsc</code> object</td>
<td>iTDsc Board, <code>Master</code> property</td>
</tr>
<tr>
<td><code>TDscGroup</code> object</td>
<td>iTDsc Board Group</td>
</tr>
<tr>
<td><code>TExternalBoard</code> object</td>
<td>iTExternalBoard property</td>
</tr>
<tr>
<td><code>TFilterDesign</code> object</td>
<td>iTFilterDesign property</td>
</tr>
<tr>
<td><code>TServerDiskLog</code> object</td>
<td>iTServerDiskLog property</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Supports</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TBufferGetEx</code> type</td>
<td><code>BufferGetEx</code> method</td>
</tr>
<tr>
<td><code>TDscIoInt64</code> type</td>
<td><code>ServerDiskLogBytes</code> property</td>
</tr>
<tr>
<td><code>TFilterParam</code> type</td>
<td><code>FilterParametersGet</code> method</td>
</tr>
<tr>
<td><code>TServerDiskLogConfig</code> type</td>
<td><code>ServerDiskLogConfigGet</code> method</td>
</tr>
<tr>
<td><code>TXbPinConfig</code> type</td>
<td><code>XbPinConfigGet</code> method</td>
</tr>
<tr>
<td></td>
<td><code>XbPinConfigSet</code> method</td>
</tr>
</tbody>
</table>
**TDsc object**

The **TDsc** object defines the behavior of the iDSC board and the **Master** property.

**Remarks**

**TDsc** is the underlying object of the iDSC board. **TDsc** supports the properties, methods, and events below.

**Properties**

- **Address** property
- **BufferGetEnabled** property
- **ConfigDialogOptions** property
- **Dap1Text** property
- **ExternalBoard** property
- **FilterDesign** property
- **GroupDelay** property
- **InputRange** property
- **Master** property
- **MemoryUsed** property
- **OperateMode** property
- **PinEnabled** property
- **PinEnabledCount** property
- **RemoteMaster** property
- **Running** property
- **SampleRate** property
- **ScansDiscarded** property
- **ServerDiskLog** property
- **ServerDiskLogBytes** property
- **ServerDiskLogEnabled** property
- **SlaveCount** property
- **TcEnabled** property
- **TcEnabledCount** property
- **TcMaximum** property
- **TcWidth** property
- **XbEnabled** property

**Methods**

- **BufferAvail** method
BufferGet method
BufferGetEx method
Calibrate method
CommandsLoad method
ConfigDialogShow method
HardwareStop method
SlaveName method
StartAcquiring method
StopAcquiring method
StructPrepare method
SystemErrorProcess method
XbCalibrate method

Events

OnCalibrateProgress event
OnHardwareDelayChange event
OnInputRangeUpdate event
OnPinEnabledUpdate event
OnSystemError event
**TDscGroup object**

The **TDscGroup** object defines the behavior of the iDSC group.

**Remarks**

**TDscGroup** is the underlying object of the iDSC group. The iDSC group consists of a list of iDSC boards that are of type **TDsc**. The specific iDSC board can be accessed through the array property **Dsc**.

**TDscGroup** supports the properties, methods, and events below.

**Properties**

- **Count** property
- **Dsc** property

**Methods**

- **AddOne** method
- **ConfigDialogShow** method
- **DeleteOne** method

**Events**

- **OnAfterNumDscChange** event
- **OnBeforeNumDscChange** event
**TExternalBoard object**

The `TExternalBoard` object defines the behavior of the `ExternalBoard` property.

**Remarks**

`TExternalBoard` supports the properties and methods below.

**Properties**

- `InputOffset` property
- `InputOffsetRange` property
- `InputRange` property
- `InputType` property
- `OutputExcitation` property

**Methods**

- `XbPinConfigGet` method
- `XbPinConfigSet` method
**TFilterDesign object**

The **TFilterDesign** object defines the behavior of the **FilterDesign** property.

**Remarks**

**TFilterDesign** supports the properties and methods below.

**Properties**

- **Attenuation** property
- **CutoffFreqHigh** property
- **CutoffFreqLow** property
- **CutoffSlopeHigh** property
- **CutoffSlopeLow** property
- **FilterName** property
- **FilterType** property
- **PinToFilterMap** property
- **Sharpness** property

**Methods**

- **FilterIndex** method
- **FilterParametersGet** method
- **FilterParametersSet** method
- **TransferFunctionGet** method
- **UnitStepGet** method
- **UnitStepLengthGet** method
**TServerDiskLog object**

The `TServerDiskLog` object defines the behavior of the `ServerDiskLog` property.

**Remarks**

`TServerDiskLog` supports the properties and methods below.

**Properties**

- `BlockSize` property
- `FileFlagsAttributes` property
- `FileName` property
- `FileShareMode` property
- `Flags` property
- `MaxCount` property
- `OpenFlags` property

**Methods**

- `ServerDiskLogConfigGet` method
- `ServerDiskLogConfigSet` method
**TBufferGetEx type**

The `TBufferGetEx` type defines the behavior of `BufferGetEx`.

**Declaration**

```
TBufferGetEx = packed record
  iInfoSize: integer; // Size of this record.
  iMinBytes: integer; // Minimum number of bytes to get.
  iMaxBytes: integer; // Maximum number of bytes to get.
  iTimeWait: integer; // Longest time to wait for data.
  iTimeOut: integer; // Longest total time for operation.
  iBytesMultiple: integer; // Bytes to get is a multiple of this.
end;
```

**Members**

- `iInfoSize`  
  Size of this information record.

- `iMinBytes`  
  Minimum number of bytes to get. It can be zero or any positive integer.

- `iMaxBytes`  
  Maximum number of bytes to get. `iMaxBytes` must be greater than or equal to `iMinBytes`.

- `iTimeWait`  
  Longest time in milliseconds that the get operation can be blocked waiting for data. If no data shows up in that amount of time, the operation should be aborted.

- `iTimeOut`  
  Longest time in milliseconds that the get operation should complete. If it fails to complete in that amount of time, the operation is aborted. When this member is specified, it takes precedence over `iTimeWait`. This member is ignored if its value is zero.

- `iBytesMultiple`  
  The number of bytes to get is always a multiple of `iBytesMultiple`.

**Remarks**

`TBufferGetEx` is used in `BufferGetEx`. `TBufferGetEx` should be initialized using `StructPrepare` or `BufferGetEx` will fail.
Each member of **TBufferGetEx** must be initialized to the appropriate value before being passed to **BufferGetEx**. The member **iMinBytes** must be greater than or equal to zero, and the member **iMaxBytes** must be greater than or equal to **iMinBytes**. If **iMinBytes** is zero, **BufferGetEx** will never block even when there are no data available.

A zero value of **iBytesMultiple** is treated the same as one. The value of **iBytesMultiple** cannot be larger than the maximum pipe buffer size on the PC side (converted to bytes) minus 1024, or minus the iDSC side blocking size (converted to bytes), whichever is larger; otherwise, the first condition causes an error. The second condition is not checked and may cause a deadlock. It is the application's responsibility to guarantee that it never happens.

Both **iMinBytes** and **iMaxBytes** must be an integral multiple of the **iBytesMultiple** value; otherwise, an error occurs.

**See Also**
**BufferGetEx** method, **StructPrepare** method
TDscIoInt64 type

The TDscIoInt64 type defines the behavior of the ServerDiskLogBytes property and the i64MaxCount parameter of TServerDiskLogConfig. TDscIoInt64 represents a 64-bit integer type.

Declaration

{$IFDEF DscIoNoInt64}
TDscIoInt64 = packed record
    dwLowPart: DWORD; // Low 32-bits of 64-bit integer type.
    dwHighPart: DWORD; // High 32-bits of 64-bit integer type.
end;
{$ELSE}
TDscIoInt64 = int64;
{$ENDIF}

Remarks

TDscIoInt64 supports the Delphi 5, Delphi 6, and Delphi 7, and C++Builder 5, and C++Builder 6 development environments.

Example

The following examples show how to initialize the i64MaxCount parameter of TServerDiskLogConfig to a value of 4294967296. The first example is for environments that support the int64 data type, while the second example is for environments that do not support the int64 data type.

Dsc1 is of type TDsc.

// Example 1: int64 type is supported
var
    sdlc: TServerDiskLogConfig;
begin
    Dsc1.StructPrepare(sdlc, SizeOf(sdlc));
    sdlc.i64MaxCount := 4294967296;
    ...
end;
// Example 2: int64 type is not supported

var
    sdlc: TServerDiskLogConfig;
begin
    Dsc1.StructPrepare(sdlc, SizeOf(sdlc));
    sdlc.i64MaxCount.dwLowPart := 0;
    sdlc.i64MaxCount.dwHighPart := 1;
    ...
end:

See Also

TServerDiskLogConfig type, ServerDiskLogBytes property
The **TFilterParam** type defines the behavior of the **FilterParametersGet** method and **FilterParametersSet** method.

**Declaration**

```delphi
TFilterParam = packed record
  iInfoSize: integer; // Size of this record.
  achName: array[0..63] of char; // Filter name.
  iFilterType: integer; // Filter type.
  iSharpness: integer; // Filter sharpness.
  fCutoffFreqLow: single; // Filter low cutoff frequency.
  fCutoffSlopeLow: single; // Filter low cutoff slope.
  fCutoffFreqHigh: single; // Filter high cutoff frequency.
  fCutoffSlopeHigh: single; // Filter high cutoff slope.
  fAttenuation: single; // Filter attenuation.
end;
```

**Members**

- **iInfoSize**
  
  Size of this information record.

- **achName**
  
  Name of the filter. The name is restricted to 63 characters plus one null-terminated character. The default is **FD0**, **FD1**, **FD2**, **FD3**, **FD4**, **FD5**, **FD6**, and **FD7** for each of the eight filters.

- **iFilterType**
  
  Type of the filter, either lowpass or bandpass. These constants simplify selecting the filter type: **Dsc_LowPass**, **Dsc_BandPass**. The default is lowpass.

- **iSharpness**
  
  Sharpness of the filter specified as an odd number. Valid numbers are in the range 37 to 255, depending on the sample rate.

  The default is 37 for sample rate 153600 s/s, 119 for sample rate 102400 s/s, 95 for sample rate 76800 s/s, 195 for sample rates of 51200 s/s, 1024 s/s, 2048 s/s, and 1024 s/s and 137 for all other sample rates.

- **fCutoffFreqLow**
  
  Low cutoff frequency of the filter specified in Hertz. Valid numbers are in the range 2% to 80% of the Nyquist frequency (0.02 * Nyquist frequency to 0.8 *
Nyquist frequency). The Nyquist frequency is half the sample rate. The default is half the Nyquist frequency.

**fCutoffSlopeLow**

Low cutoff slope of the filter specified as a fraction of the Nyquist frequency, with the Nyquist frequency normalized to 1.0. Valid numbers are in the range 0.0 to 0.8. 0.0 corresponds to 0% of the Nyquist frequency and 0.8 corresponds to 80% of the Nyquist frequency. The Nyquist frequency is half the sample rate.

The default is 0.0 for sample rates of 153600 s/s, 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, and 1024 s/s and 0.04 for all other sample rates.

**fCutoffFreqHigh**

High cutoff frequency of the filter specified in Hertz. Valid numbers are in the range 2% to 80% of the Nyquist frequency (0.02 * Nyquist frequency to 0.8 * Nyquist frequency). The Nyquist frequency is half the sample rate. The default is three-quarters the Nyquist frequency.

**fCutoffSlopeHigh**

High cutoff slope of the filter specified as a fraction of the Nyquist frequency, with the Nyquist frequency normalized to 1.0. Valid numbers are in the range 0.0 to 0.8. 0.0 corresponds to 0% of the Nyquist frequency and 0.8 corresponds to 80% of the Nyquist frequency. The Nyquist frequency is half the sample rate.

The default is 0.0 for sample rates of 153600 s/s, 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, and 1024 s/s and 0.04 for all other sample rates.

**fAttenuation**

Attenuation of the filter in the stopband region. Valid numbers are in the range 6.0 to 12.0. The larger the number, the more attenuation in the stopband, but this will further attenuate the frequencies near cutoff.

The default is 10.2 for sample rate 153600 s/s, 9.4 for sample rate 76800 s/s, 9.8 for sample rates of 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, and 1024 s/s and 9.0 for all other sample rates.

**Remarks**

_TFilterParam_ is used in _FilterParametersGet_ and _FilterParametersSet_. _TFilterParam_ should be initialized using _StructPrepare_ or both methods will fail.

(FilterParametersGet will return the filter parameters in the iSharpness, fCutoffFreqLow, fCutoffSlopeLow, fCutoffFreqHigh, fCutoffSlopeHigh, and fAttenuation members.)
Before invoking FilterParametersSet, the user must set the iSharpness, fCutoffFreqLow, fCutoffSlopeLow, fCutoffFreqHigh, fCutoffSlopeHigh, and fAttenuation members to the new desired values.

See Also
FilterParametersGet method, FilterParametersSet method, StructPrepare method
**TServerDiskLogConfig type**

The `TServerDiskLogConfig` type defines the behavior of `ServerDiskLogConfigGet` and `ServerDiskLogConfigSet`.

**Declaration**

```
TServerDiskLogConfig = packed record
  iInfoSize: integer; // Size of this record.
  dwFlags: DWORD; // Logging behavior flags.
  pszFileName: PChar; // File name.
  dwFileNameSize: DWORD; // Size of file name.
  dwFileShareMode: DWORD; // File share properties.
  dwOpenFlags: DWORD; // File open options.
  dwFileFlagsAttributes: DWORD; // File attributes.
  dwBlockSize: DWORD; // Size of block to write.
  i64MaxCount: TDscIoInt64; // File maximum count.
end;
```

**Members**

*iInfoSize*
Size of this information record.

*dwFlags*
Flags to control disk logging behavior. The constants below simplify selecting `dwFlags`. The default is `DscDlfServerSide` and `DscDlfFlushBefore`.

*DscDlfServerSide*
Log on the same side of the network connection as the iDSC. If not specified, logging will take place on the application (client) side of the network connection.

*DscDlfFlushBefore*
Flush the input data pipe before beginning the logging session. Default action is to not flush the pipes before logging.

*DscDlfFlushAfter*
Flush the input pipe after the logging session has terminated. Default action is to not flush the pipes after logging.

*DscDlfMirrorLog*
Enable mirror logging. Mirror logging creates a copy of the logged data in another file.
DscDlfAppendData
Allow new data to be appended to an existing file. Only the
DscOfOpenAlways and DscOfOpenExisting flags of the dwOpenFlags
member can be used for appending.

DscDlfBlockTransfer
Open the file with no intermediate buffering or caching and access the file
in a special way that is highly dependent on the target disk attributes to
improve performance. This transfer mode adds overhead to slow rate
transfer with small buffers. It should only be used when necessary with a
very large dwBlockSize value (such as 1048576 and above). If this
option is selected, dwBlockSize is automatically set to 1048576.

pszFileName
Name of the primary disk log file and name of a possible mirror disk log file, if
mirror logging is enabled. The default is an empty string. The size of the user
allocated buffer must be specified using dwFileNameSize when used with
ServerDiskLogConfigGet.

Mirror logging is enabled by selecting the DscDlfMirrorLog flag of the
dwFlags member. Multiple file names are separated by semi-colons. Currently,
only one mirror file is allowed. Both files must be on the same side of the
DAPcell Local/DAPcell service (the PC application side or the iDSC side).

dwFileNameSize
Size of the user allocated buffer to store the file name specified by
pszFileName. The size must include an extra space for the null terminator. This
field is important in ServerDiskLogConfigGet. It is not used in
ServerDiskLogConfigSet.

If dwFileNameSize is 0, the file name is not returned in pszFileName. If
dwFileNameSize is 1, only the null terminator is returned in pszFileName.

dwFileShareMode
File share mode of the disk log file. The constants below simplify selecting
dwFileShareMode. The default is DscFsmRead.

DscFsmNone The file cannot be used by another process.
DscFsmRead The file can be read by another process.
DscFsmWrite The file can be written to by another process.
DscFsmReadWrite The file can be read and written to by another process.

dwOpenFlags
File open options of the disk log file. The constants below simplify selecting
dwOpenFlags. The default is DscOfCreateAlways.
DscOfCreateNew
Create a new file. Creation fails if the file already exists.

DscOfCreateAlways
Create a new file. If the file already exists, it is overwritten.

DscOfOpenAlways
Open an existing file. If the file does not exist, it will be created.

DscOfOpenExisting
Open an existing file without resetting permissions. Opening fails if the file does not exist.

dwFileFlagsAttributes
File attributes of the disk log file. The constants below simplify selecting dwFileFlagsAttributes. The default is DscFfaAttributeNormal.

<table>
<thead>
<tr>
<th>常数名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>DscFfaAttributeNormal</td>
<td>No special attributes.</td>
</tr>
<tr>
<td>DscFfaAttributeEncrypted</td>
<td>The data in the file is encrypted.</td>
</tr>
<tr>
<td>DscFfaFlagWriteThrough</td>
<td>Write through any intermediate caching and go directly to disk.</td>
</tr>
<tr>
<td>DscFfaFlagSequentialScan</td>
<td>Can be used to optimize the transfer of large blocks of data. Most applications will not need this flag.</td>
</tr>
</tbody>
</table>

dwBlockSize
Minimum amount of data, in bytes, to write to the disk log file at one time. This field is provided for disk transfer optimization. The default is 8192.

i64MaxCount
Maximum number of bytes to log. The default is 0, which causes logging to continue indefinitely until StopAcquiring is invoked. It is a TdscIoInt64 type.

Remarks
TServerDiskLogConfig is used in ServerDiskLogConfigGet and ServerDiskLogConfigSet. TServerDiskLogConfig should be initialized using StructPrepare or both functions will fail.

ServerDiskLogConfigGet will return the server disk log configuration in the dwFlags, pszFileName, dwFileShareMode, dwOpenFlags, dwFileFlagsAttributes, dwBlockSize, and i64MaxCount members of TServerDiskLogConfig.

To use ServerDiskLogConfigSet it is recommended that the user invokes ServerDiskLogConfigGet to get the server disk log configuration defaults, updates the pertinent fields, and then invokes ServerDiskLogConfigSet.
See Also
  StructPrepare method, ServerDiskLogConfigGet method,
  ServerDiskLogConfigSet method
**TXbPinConfig type**

The `TXbPinConfig` type defines the behavior of the `XbPinConfigGet` method and `XbPinConfigSet` method.

### Declaration

```pascal
TXbPinConfig = packed record
  iInfoSize: integer; // Size of this record.
iInputType: integer; // Input type.
fInputRange: single; // Input range voltage.
fInputOffset: single; // Input offset voltage.
fInputOffsetRange: single; // Input offset range voltage.
fOutputExcitation: single; // Output excitation voltage.
end;
```

### Members

- **iInfoSize**
  Size of this information record.

- **iInputType**
  Type of input signal, DC coupling, AC coupling or excitation. These constants simplify selecting the input type: `DscXb_DCCoupling`, `DscXb_ACCoupling`, `DscXb_Excitation`. The default is `DscXb_DCCoupling`.

- **fInputRange**
  Input range of the signal specified in Volts. Please note that this input range is different from `InputRange`, and only works if `InputRange` is set to +/- 5V.

  If you select 0.5 the input range is +/- 500 mV, if you select 2.0 the input range +/- 2V. If you specify an invalid input range, the input range will not change from its previous setting. The default is 10.0.

  Valid input ranges are:
  
  - 0.01, 0.02, 0.05,
  - 0.1, 0.2, 0.5,
  - 1.0, 2.0, 5.0,
  - 10.0

- **fInputOffset**
  Input offset of the signal specified in Volts. The input offset must be within the range of the input offset range. If you specify an invalid input offset, the input offset will not change from its previous setting. The default is 0.0.
**fInputOffsetRange**
Input offset range of the signal returned in Volts. The input offset range is determined by the input range. For example, if the input range is 0.5 then the input offset range is 2.5 for +/-2.5 V, if the input range is 2.0 then the input offset range is 1.0 for +/- 1V. This is a read only property that is dependent on the input range and you cannot specify it.

Valid input offset ranges are:

- 0.5 when the input range is 0.01
- 1.0 when the input range is 0.02
- 2.5 when the input range is 0.05
- 0.5 when the input range is 0.1
- 1.0 when the input range is 0.2
- 2.5 when the input range is 0.5
- 1.0 when the input range is 1.0
- 1.0 when the input range is 2.0
- 5.0 when the input range is 5.0
- 5.0 when the input range is 10.0

**fOutputExcitation**
Output excitation signal specified in Volts. If you specify an invalid output excitation, the output excitation will not change from its previous setting. The default is 0.0.

Valid output excitation ranges are:

- 0.0, 1.0, 2.0, 5.0, 10.0

**Remarks**

`TXbPinConfig` is used in `XbPinConfigGet` and `XbPinConfigSet`. `TXbPinConfig` should be initialized using `StructPrepare` or both functions will fail.

`XbPinConfigGet` will return the pin configuration in the `iInputType`, `fInputRange`, `fInputOffset`, `fInputOffsetRange`, and `fOutputExcitation` members of `TXbPinConfig`.

Before invoking `XbPinConfigSet`, the `iInputType`, `fInputRange`, `fInputOffset`, and `fOutputExcitation` members of `TXbPinConfig` must be set to the new desired values. `fInputOffsetRange` cannot be set since it is a read only property.

**See Also**

`StructPrepare` method, `XbPinConfigGet` method, `XbPinConfigSet` method
# Property, Method, and Event Summary

The DSCC interface provides a complete set of properties, methods, and events for communicating with the iDSC board. Each property, method, and event falls into one of several categories.

<table>
<thead>
<tr>
<th>Category</th>
<th>Tdsc Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical services</td>
<td>ConfigDialogOptions property</td>
</tr>
<tr>
<td></td>
<td>ConfigDialogShow method</td>
</tr>
<tr>
<td>Filter design services</td>
<td>FilterDesign property</td>
</tr>
<tr>
<td>Communication services</td>
<td>Calibrate method</td>
</tr>
<tr>
<td></td>
<td>CommandsLoad method</td>
</tr>
<tr>
<td></td>
<td>StartAcquiring method</td>
</tr>
<tr>
<td></td>
<td>StopAcquiring method</td>
</tr>
<tr>
<td>I/O services</td>
<td>BufferAvail method</td>
</tr>
<tr>
<td></td>
<td>BufferGet method</td>
</tr>
<tr>
<td></td>
<td>BufferGetEnabled property</td>
</tr>
<tr>
<td></td>
<td>BufferGetEx method</td>
</tr>
<tr>
<td>System services</td>
<td>Address property</td>
</tr>
<tr>
<td></td>
<td>GroupDelay property</td>
</tr>
<tr>
<td></td>
<td>HardwareStop method</td>
</tr>
<tr>
<td></td>
<td>InputRange property</td>
</tr>
<tr>
<td></td>
<td>MemoryUsed property</td>
</tr>
<tr>
<td></td>
<td>PinEnabled property</td>
</tr>
<tr>
<td></td>
<td>PinEnabledCount property</td>
</tr>
<tr>
<td></td>
<td>Running property</td>
</tr>
<tr>
<td></td>
<td>SampleRate property</td>
</tr>
<tr>
<td></td>
<td>ScansDiscarded property</td>
</tr>
<tr>
<td></td>
<td>StructPrepare method</td>
</tr>
<tr>
<td>Master/slave services</td>
<td>Master property</td>
</tr>
<tr>
<td></td>
<td>OperateMode property</td>
</tr>
<tr>
<td></td>
<td>RemoteMaster property</td>
</tr>
<tr>
<td></td>
<td>SlaveCount property</td>
</tr>
<tr>
<td></td>
<td>SlaveName method</td>
</tr>
</tbody>
</table>

DSC Component Programmer's Interface
<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAPL services</td>
<td><strong>Dap1Text</strong> property</td>
</tr>
<tr>
<td>External board services</td>
<td><strong>ExternalBoard</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>XbCalibrate</strong> method</td>
</tr>
<tr>
<td></td>
<td><strong>XbEnabled</strong> property</td>
</tr>
<tr>
<td>Server disk log services</td>
<td><strong>ServerDiskLog</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>ServerDiskLogBytes</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>ServerDiskLogEnabled</strong> property</td>
</tr>
<tr>
<td>Timing channel services</td>
<td><strong>TcEnabled</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>TcEnabledCount</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>TcMaximum</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>TcWidth</strong> property</td>
</tr>
<tr>
<td>Error handling services</td>
<td><strong>SystemErrorProcess</strong> method</td>
</tr>
<tr>
<td>Events</td>
<td><strong>OnCalibrateProgress</strong> event</td>
</tr>
<tr>
<td></td>
<td><strong>OnHardwareDelayChange</strong> event</td>
</tr>
<tr>
<td></td>
<td><strong>OnInputRangeUpdate</strong> event</td>
</tr>
<tr>
<td></td>
<td><strong>OnPinEnabledUpdate</strong> event</td>
</tr>
<tr>
<td></td>
<td><strong>OnSystemError</strong> event</td>
</tr>
<tr>
<td>Category</td>
<td><strong>TDscGroup Services</strong></td>
</tr>
<tr>
<td>Graphical services</td>
<td><strong>ConfigDialogShow</strong> method</td>
</tr>
<tr>
<td>Configuration services</td>
<td><strong>AddOne</strong> method</td>
</tr>
<tr>
<td></td>
<td><strong>Count</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>DeleteOne</strong> method</td>
</tr>
<tr>
<td></td>
<td><strong>Dsc</strong> property</td>
</tr>
<tr>
<td>Events</td>
<td><strong>OnAfterNumDscChange</strong> event</td>
</tr>
<tr>
<td></td>
<td><strong>OnBeforeNumDscChange</strong> event</td>
</tr>
<tr>
<td>Category</td>
<td><strong>TExternalBoard Services</strong></td>
</tr>
<tr>
<td>External board properties</td>
<td><strong>InputOffset</strong> property</td>
</tr>
<tr>
<td></td>
<td><strong>InputOffsetRange</strong> property</td>
</tr>
</tbody>
</table>
InputRange property
InputType property
OutputExcitation property

External board utilities  XbPinConfigGet method
XbPinConfigSet method

Category  TFilterDesign Services

Filter design properties  Attenuation property
CutoffFreqHigh property
CutoffFreqLow property
CutoffSlopeHigh property
CutoffSlopeLow property
FilterName property
FilterType property
PinToFilterMap property
Sharpness property

Filter design utilities  FilterIndex method
FilterParametersGet method
FilterParametersSet method
TransferFunctionGet method
UnitStepGet method
UnitStepLengthGet method

Category  TServerDiskLog Services

Server disk log properties  BlockSize property
FileFlagsAttributes property
FileName property
FileShareMode property
Flags property
MaxCount property
OpenFlags property

Server disk log utilities  ServerDiskLogConfigGet method
ServerDiskLogConfigSet method
The following figure displays available design time properties with the `TDsc` object from the Delphi 7.0 development environment.

![Object Inspector](image)

### DSC Component Programmer’s Interface
**AddOne method**

The *AddOne* method adds one iDSC to the iDSC group.

**Applies To**
TDscGroup

**Declaration**

```
function AddOne: integer;
```

**Return Values**

The return value is the index of the added iDSC or *Count* - 1.

**Remarks**

*AddOne* adds one iDSC to the iDSC group and returns the index of the added iDSC. The index of the iDSC is the *Count* minus one. The maximum number of iDSCs in the iDSC group is 64 which means that the maximum index is 63. When *AddOne* is invoked, the *Count* increases by one.

As an example, *AddOne* returns 3. Therefore, the newly added iDSC index is 3 and the *Count* is 4.

**See Also**

*DeleteOne* method, *Count* property, *ConfigDialogShow* method
**Address property**

The *Address* property specifies the machine name and iDSC board name. The name returned is in **UNC** format.

**Applies To**

TDesc

**Declaration**

property Address: string;

**Default**

\\.\Dap0

**Access Restrictions**

None

**Remarks**

*Address* consists of two portions, the machine name and the iDSC board name. The naming method is based on the **UNC**.

Using `\\.\Dap0` as an example, `\.` denotes the local machine and `\Dap0` is the name of the iDSC board. If you are connected to a remote machine through DAPcell named PC45, that contains `Dap0`, then *Address* is `\\PC45\Dap0`.

**See Also**

UNC
**Attenuation property**

The **Attenuation** property is an array property that specifies the attenuation of the filter.

**Applies To**

*TFilterDesign*

**Declaration**

```pascal
property Attenuation[iFilterIndex: integer]: single;
```

**Parameters**

*iFilterIndex*

Filter index of interest. Valid filter indices are 0 through 7.

**Default**

Dependent on SampleRate

- 153600 s/s: 10.2
- 76800 s/s: 9.4
- 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, 1024 s/s: 9.8
- All others: 9.0

**Access Restrictions**

Run time only

**Remarks**

**Attenuation** specifies the filter response in the stopband region. Valid numbers are in the range 6.0 to 12.0. The larger the number, the more attenuation in the stopband, but this will further attenuate the frequencies near cutoff.

Valid filter indices are 0 through 7. If the filter index is not valid, the exception ‘Filter index is out of range’ is raised.

**Attenuation** is a **TFilterDesign** property.
See Also

`ConfigDialogShow` method
**BlockSize property**

The **BlockSize** property specifies the size of block to write.

**Applies To**

- **TServerDiskLog**

**Declaration**

```
property BlockSize: DWORD;
```

**Default**

8192

**Access Restrictions**

None

**Remarks**

**BlockSize** specifies the minimum amount of data, in bytes, to write to the disk log file at one time. This field is provided for disk transfer optimization.

**See Also**

- **ServerDiskLog** property, **TServerDiskLog** object
**BufferAvail method**

The **BufferAvail** method gets the number of bytes available for reading from the iDSC board.

### Applies To

**TDsc**

### Declaration

```pascal
function BufferAvail: integer;
```

### Return Values

- If the function succeeds, the return value is the number of bytes available for reading.
- If the function fails, the return value is -1.
- If there are no data available, the return value is 0.

### Remarks

**BufferAvail** returns the number of bytes already buffered. An application is safe to read that number from the iDSC board without being blocked.

For efficient data transfer, it is best to use **BufferGetEx** with a time wait, and avoid using **BufferAvail**. **BufferGetEx** returns the actual number of bytes read which lets the application know what data have been transferred.

### See Also

**BufferGet** method, **BufferGetEx** method
**BufferGet method**

The **BufferGet** method reads a block of data from the iDSC board.

**Applies To**

**TDsc**

**Declaration**

```pascal
function BufferGet(
  iBytes: integer;  // Number of bytes to read.
  iTimeWait: integer;  // Longest time to wait for data.
  var Buffer: integer;  // Buffer to receive data.
): integer;
```

**Parameters**

- **iBytes**
  Number of bytes to read.

- **iTimeWait**
  Specifies the longest time in milliseconds that the get operation can be blocked waiting for data. If no data shows up in that amount of time, the operation should be aborted.

- **Buffer**
  Buffer for storing the data from the iDSC board.

**Return Values**

If the function succeeds, the return value is the number of bytes read. If the function fails, the return value is -1 or an exception is raised. If there are no data available, the return value is 0.

**Remarks**

**BufferGet** attempts to read all the requested **iBytes** from the iDSC board. If all the requested **iBytes** are not available for **iTimeWait** milliseconds, it returns with the number of bytes read so far.

This method is useful for displaying the acquired and filtered data in graphs, tables, etc.
See Also

BufferGetEx method
**BufferGetEnabled property**

The `BufferGetEnabled` property enables or disables the `BufferGet` method and `BufferGetEx` method.

**Applies To**

`TDsc`

**Declaration**

```pascal
property BufferGetEnabled: boolean;
```

**Default**

True

**Access Restrictions**

Run time only

**Remarks**

`BufferGetEnabled` allows enabling or disabling the `BufferGet` method and `BufferGetEx` method. If `BufferGetEnabled` is true, `BufferGet` and `BufferGetEx` are enabled. If `BufferGetEnabled` is false, `BufferGet` and `BufferGetEx` are disabled, and invoking those methods will return -1.

`BufferGetEnabled` is useful if another PC wants to access the iDSC data from the PC with the iDSC through networking. The networking capability is only available with the DAPcell Local and DAPcell servers. It is not available with the Accel32 server.

**See Also**

`BufferGet` method, `BufferGetEx` method
**BufferGetEx method**

The `BufferGetEx` method reads a block of data from the iDSC board, using the `TBufferGetEx` record to control its behavior.

**Applies To**

`TDsc`

**Declaration**

```pascal
function BufferGetEx(
    const bgeBufferGetEx: TBufferGetEx;  // Information record.
    var Buffer  // Buffer to receive data.
): integer;
```

**Parameters**

- `bgeBufferGetEx` is a record that passes the appropriate parameters.
- `TBufferGetEx` must be initialized using `StructPrepare`.
- `Buffer` is the buffer for storing the data from the iDSC board.

**Return Values**

If the function succeeds, the return value is the number of bytes read. If the function fails, an exception is raised. If there are no data available, the return value is 0. If `CommandsLoad` or `StartAcquiring` has not been invoked, the return value is -1.

**Remarks**

`BufferGetEx` is an extended version of `BufferGet`. It allows a minimum request count, `iMinBytes`, and a maximum request count, `iMaxBytes`, specification. Both `iMinBytes` and `iMaxBytes` are passed into this function through the `TBufferGetEx` record. Both `iMinBytes` and `iMaxBytes` are in bytes and must be an integral multiple of `iBytesMultiple`. The function reads at least `iMinBytes` bytes of data. Once it reads enough data to cover `iMinBytes`, the function reads all available data up to `iMaxBytes` without waiting. The actual bytes returned is always an integral multiple of `iBytesMultiple`.

---

DSC Component Programmer’s Interface 285
Before \textit{iMinBytes} is covered, the function will be blocked waiting for data if the target pipe becomes empty. In this case, the two members of the \texttt{TBufferGetEx} record, \textit{iTimeWait} and \textit{iTimeOut}, determine the behavior of the function. If \textit{iMinBytes} is not covered in \textit{iTimeOut} milliseconds, or if no data are available for \textit{iTimeWait} milliseconds, the function returns immediately. The return value is then the number of bytes actually read up to the point where the operation is aborted. It can be zero or any integral multiple of \textit{iBytesMultiple} less than \textit{iMinBytes}. An application can check the return value to determine if a time-out has occurred.

\textbf{See Also}

\texttt{BufferGet} method
Calibrate method

The `Calibrate` method performs calibration on the iDSC board.

Applies To
TDsc

Declaration
procedure Calibrate;

Remarks
`Calibrate` calibrates the iDSC board for DC gain and offset, and saves the calibration values.

`Calibrate` is automatically invoked when a user selects `StartAcquiring` for the first time. The next time a user uses the component, the saved calibration values are reused.

If the calibration values are not found, calibration is automatically re-invoked. A user can force recalibration by calling `Calibrate`.

See Also
`StartAcquiring` method
**CommandsLoad method**

The **CommandsLoad** method downloads the configuration commands to the iDSC board and performs the necessary configuration for filtering.

**Applies To**

TDsc

**Declaration**

```delphi
procedure CommandsLoad;
```

**Remarks**

**CommandsLoad** configures the iDSC board with the appropriate programs and coefficients. The filter designs are used internally by **CommandsLoad** to calculate and download the appropriate commands.

If the iDSC board configuration or filter designs change and **StartAcquiring** is selected, **StartAcquiring** automatically invokes **CommandsLoad** and downloads new commands to the iDSC board. Data will show up at the PC two times **GroupDelay** seconds later.

If the iDSC board configuration or filter designs change and **CommandsLoad** is selected before **StartAcquiring**, data will show up immediately at the PC. Data that show up immediately are actually data that was sampled **GroupDelay** seconds ago.

**See Also**

**GroupDelay** property, **StartAcquiring** method
**ConfigDialogOptions property**

The `ConfigDialogOptions` property allows manipulating the display options of `ConfigDialogShow`.

**Applies To**

`TDesc`

**Declaration**

```pascal
property ConfigDialogOptions: TConfigDialogOptions;
```

```pascal
TConfigDialogOptions = set of TConfigDialogOption;
```

```pascal
TConfigDialogOption = (
    cdoInputScreenHide, cdoFDScreenHide, cdoTcHide,
    cdoFD0Hide, cdoFD1Hide, cdoFD2Hide, cdoFD3Hide,
    cdoFD4Hide, cdoFD5Hide, cdoFD6Hide, cdoFD7Hide );
```

**Default**

`[cdoTcHide]`

**Access Restrictions**

None

**Remarks**

`ConfigDialogOptions` specifies the display options of `ConfigDialogShow`. It is of type `TConfigDialogOptions`, which is a set of `TConfigDialogOption`. `ConfigDialogOptions` can be manipulated using the `set` operators.

`ConfigDialogOptions` specifies one or more of the following options.

- `cdoInputScreenHide` Hide the Input Screen.
- `cdoFDScreenHide` Hide the Filter Design Screen.
- `cdoTcHide` Hide the timing channels selection.
- `cdoFD0Hide` Hide filter design index 0.
- `cdoFD1Hide` Hide filter design index 1.
- `cdoFD2Hide` Hide filter design index 2.
- `cdoFD3Hide` Hide filter design index 3.
Hide filter design index 4.

Hide filter design index 5.

Hide filter design index 6.

Hide filter design index 7.

See Also

`ConfigDialogShow` method
**ConfigDialogShow method**

The `ConfigDialogShow` method displays modal dialog screens for graphical configuration of the inputs and filter designs.

**Applies To**

`TDsc`

**Declaration**

```pascal
function ConfigDialogShow: integer;
```

**Return Values**

If the user selects the OK button when changes are made, the return value is `mrOk` (of value 1). If the user selects the OK button when no changes are made, the return value is `mrIgnore` (of value 5). If the user selects the Cancel button, the return value is `mrCancel` (of value 2).

**Remarks**

`ConfigDialogShow` simplifies the configuration of the input and filter design process with a graphical interface. The Input Screen provides a quick method for selecting the sample rate, selecting the input range, mapping the input pins to selected filter designs, and enabling or disabling the input pins.
The figure below displays the **Input Screen**.

![Table of filter design parameters](image)

The **Filter Design Screen** allows easy manipulation of filter parameters like sharpness, low cutoff frequency, low cutoff slope, high cutoff frequency, high cutoff slope, and attenuation, either by entering a valid number or by adjusting a slider. When the user selects the right click button, there are several more options like **Crosshair Track, Y Display** (linear, linear zoom, log, log zoom, and unit step), **Defaults Load,** **Copy,** and **Paste.** The **Copy (Ctrl-C)** and **Paste (Ctrl-V)** feature copies the filter parameters from one filter design tab to another.
The figure below displays the Filter Design Screen.

A user can also design filters using the run-time design techniques available. These include `FilterParametersGet` and `FilterParametersSet`.

The Input Screen, Filter Design Screen, timing channels selection, and individual filter designs can be hidden using `ConfigDialogOptions`.

See Also
- `ConfigDialogOptions` property,
- `FilterParametersGet` method,
- `FilterParametersSet` method
**ConfigDialogShow method**

The `ConfigDialogShow` method displays modal dialog screens for graphical configuration of multiple iDSC boards.

**Applies To**

`TDscGroup`

**Declaration**

```pascal
function ConfigDialogShow: integer;
```

**Return Values**

If the user selects the OK button, the return value is IDOK (of value 1). If the user selects the Cancel button, the return value is IDCANCEL (of value 2).

**Remarks**

`ConfigDialogShow` simplifies the configuration of multiple iDSC boards with a graphical interface. When the user expands the iDSC Boards tree, there are options for changing the Address and Mode. When the user selects the DSC name with the right click button, there are several more options like External Board Calibrate, External Board Enable, Raw Data, Remote Master, Server Disk Log, and Copy and Paste. The Copy and Paste feature allows the user to copy the entire configuration of an iDSC board to another iDSC board.

For each iDSC board, the Input Screen provides a quick method for selecting the sample rate, selecting the input range, mapping the input pins to selected filter designs, and enabling or disabling the input pins.

The Filter Design Screen allows easy manipulation of filter parameters like sharpness, low cutoff frequency, low cutoff slope, high cutoff frequency, high cutoff slope, and attenuation, either by entering a valid number or by adjusting a slider. When the user selects the right click button, there are several more options like Crosshair Track, Y Display (linear, linear zoom, log, log zoom, and unit step), Defaults Load, Copy, and Paste. The Copy (Ctrl-C) and Paste (Ctrl-V) feature copies the filter parameters from one filter design tab to another.
The figure below displays the configuration of a group of four iDSC boards. DSC0 and DSC1 are configured as Independent, while DSC2 is the Master of DSC3.

### Configuration: DSC2

<table>
<thead>
<tr>
<th>Input</th>
<th>Filter Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Rate</td>
<td>12000 Hz</td>
</tr>
<tr>
<td>Channel Count</td>
<td>0</td>
</tr>
<tr>
<td>Group Delay</td>
<td>0.0031888 sec</td>
</tr>
<tr>
<td>Input Range</td>
<td>±5 V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Pin</th>
<th>Filter Name</th>
<th>Cutoff Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>FC0</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A1</td>
<td>FC1</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A2</td>
<td>FC2</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A3</td>
<td>FC3</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A4</td>
<td>FC4</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A5</td>
<td>FC5</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A6</td>
<td>FC6</td>
<td>3200 Hz</td>
</tr>
<tr>
<td>A7</td>
<td>FC7</td>
<td>3200 Hz</td>
</tr>
</tbody>
</table>

See Also
- `AddOne` method, `DeleteOne` method
**Count property**

The *Count* property returns the number of iDSCs in the iDSC group.

**Applies To**

*TDscGroup*

**Declaration**

```property
property Count: integer;
```

**Access Restrictions**

Read only; Run time only

**Remarks**

*Count* returns the number of iDSCs in the iDSC group. *Count* increases by one when *AddOne* is invoked and decreases by one when *DeleteOne* is invoked. The maximum number of iDSCs in the iDSC group is 64.

**See Also**

*AddOne* method, *DeleteOne* method
The **CutoffFreqHigh** property is an array property that specifies the high cutoff frequency of the filter.

### Applies To

**TFilterDesign**

### Declaration

```pascal
property CutoffFreqHigh[iFilterIndex: integer]: single;
```

### Parameters

**iFilterIndex**
- Filter index of interest. Valid filter indices are 0 through 7.

### Default

Three-quarters the Nyquist frequency

### Access Restrictions

Run time only

### Remarks

**CutoffFreqHigh** determines the ideal high cutoff response which the filter is designed to approximate. It is specified in Hertz. Valid numbers are in the range 2% to 80% of the Nyquist frequency (0.02 * Nyquist frequency to 0.8 * Nyquist frequency). The Nyquist frequency is half the sample rate.

Valid filter indices are 0 through 7. If the filter index is not valid, the exception ‘Filter index is out of range’ is raised.

**CutoffFreqHigh** is a **TFilterDesign** property.

### See Also

**CutoffFreqLow** property, **ConfigDialogShow** method
**CutoffFreqLow property**

The **CutoffFreqLow** property is an array property that specifies the low cutoff frequency of the filter.

**Applies To**

**TFilterDesign**

**Declaration**

```property```
CutoffFreqLow[iFilterIndex: integer]: single;
```property```

**Parameters**

**iFilterIndex**

Filter index of interest. Valid filter indices are 0 through 7.

**Default**

Half the Nyquist frequency

**Access Restrictions**

Run time only

**Remarks**

**CutoffFreqLow** determines the ideal low cutoff response which the filter is designed to approximate. It is specified in Hertz. Valid numbers are in the range 2% to 80% of the Nyquist frequency (0.02 * Nyquist frequency to 0.8 * Nyquist frequency). The Nyquist frequency is half the sample rate.

Valid filter indices are 0 through 7. If the filter index is not valid, the exception 'Filter index is out of range' is raised.

**CutoffFreqLow** is a **TFilterDesign** property.

**See Also**

**CutoffFreqHigh** property, **ConfigDialogShow** method
**CutoffSlopeHigh property**

The *CutoffSlopeHigh* property is an array property that specifies the high cutoff slope of the filter.

**Applies To**

*TFilterDesign*

**Declaration**

```pascal
property CutoffSlopeHigh[iFilterIndex: integer]: single;
```

**Parameters**

*iFilterIndex*

Filter index of interest. Valid filter indices are 0 through 7.

**Default**

Dependent on SampleRate

- 153600 s/s, 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, 1024 s/s: 0.0
- All others: 0.04

**Access Restrictions**

Run time only

**Remarks**

*CutoffSlopeHigh* determines the ideal high cutoff response which the filter is designed to approximate. It is specified as a fraction of the Nyquist frequency, with the Nyquist frequency normalized to 1.0. Valid numbers are in the range 0.0 to 0.8. 0.0 corresponds to 0% of the Nyquist frequency and 0.8 corresponds to 80% of the Nyquist frequency. The Nyquist frequency is half the sample rate.

Valid filter indices are 0 through 7. If the filter index is not valid, the exception ‘Filter index is out of range’ is raised.

*CutoffSlopeHigh* is a *TFilterDesign* property.
See Also
CutoffSlopeLow property, ConfigDialogShow method
**CutoffSlopeLow property**

The **CutoffSlopeLow** property is an array property that specifies the low cutoff slope of the filter.

**Applies To**

*TFilterDesign*

**Declaration**

```delphi
property CutoffSlopeLow[iFilterIndex: integer]: single;
```

**Parameters**

*iFilterIndex*

Filter index of interest. Valid filter indices are 0 through 7.

**Default**

Dependent on SampleRate

- 153600 s/s, 102400 s/s, 51200 s/s, 10240 s/s, 2048 s/s, 1024 s/s: 0.0
- All others: 0.04

**Access Restrictions**

Run time only

**Remarks**

**CutoffSlopeLow** determines the ideal low cutoff response which the filter is designed to approximate. It is specified as a fraction of the Nyquist frequency, with the Nyquist frequency normalized to 1.0. Valid numbers are in the range 0.0 to 0.8. 0.0 corresponds to 0% of the Nyquist frequency and 0.8 corresponds to 80% of the Nyquist frequency. The Nyquist frequency is half the sample rate.

Valid filter indices are 0 through 7. If the filter index is not valid, the exception ‘Filter index is out of range’ is raised.

**CutoffSlopeLow** is a **TFilterDesign** property.
See Also

*CutoffSlopeHigh* property, *ConfigDialogShow* method
**DaplText property**

The **DaplText** property defines the DAPL text for the iDSC.

**Applies To**

**TDsc**

**Declaration**

```pascal
property DaplText: TStringList;
```

**Access Restrictions**

Read only; Run time only

**Remarks**

**DaplText** allows the user to define custom DAPL text for the iDSC. Each line of DAPL text must be delimited by a carriage-return, line-feed, or both.

The DAPL text is sent to the iDSC when the user invokes **CommandsLoad** or **StartAcquiring**.

**See Also**

**CommandsLoad** method, **StartAcquiring** method
DeleteOne method

The `DeleteOne` method deletes one iDSC from the iDSC group.

**Applies To**

`TDscGroup`

**Declaration**

```delphi
function DeleteOne: integer;
```

**Return Values**

The return value is the last index of the remaining iDSCs or `Count` - 1.

**Remarks**

`DeleteOne` deletes one iDSC from the iDSC group and returns the last index of the remaining iDSCs. The index of the iDSC is the `Count` minus one. The maximum number of iDSCs in the iDSC group is 64 which means that the maximum index is 63. When `DeleteOne` is invoked, the `Count` decreases by one.

As an example, `DeleteOne` returns 3. Therefore, the last index of the remaining iDSCs is 3 and the `Count` is 4.

**See Also**

`AddOne` method, `Count` property, `ConfigDialogShow` method
**Dsc property**

The **Dsc** property is an array property that allows access to a specific iDSC board.

**Applies To**

**TDscGroup**

**Declaration**

```pascal
property Dsc[i: integer]: TDsc;
```

**Parameters**

`i`

iDSC board index of interest. Valid iDSC board indices are 0 through `Count - 1`.

**Access Restrictions**

Read only; Run time only

**Remarks**

**Dsc** allows access to a specific iDSC board using the iDSC board index. Valid iDSC board indices are 0 through `Count - 1`.

The **Dsc** property should not be stored because it is destroyed and recreated in methods like **ConfigDialogShow**, **DeleteOne**, **AddOne**, etc. The **Dsc** property should be called repeatedly to access a specific iDSC board.

**Dsc** is of type **TDsc**.

**See Also**

**ConfigDialogShow** method, **Count** property, **TDsc** object
**ExternalBoard property**

The **ExternalBoard** property allows access to the external board configuration through the **TExternalBoard** object.

**Applies To**

**TDsc**

**Declaration**

```delphi
property ExternalBoard: TExternalBoard;
```

**Access Restrictions**

Read only; Run time only

**Remarks**

**ExternalBoard** provides access to the external board properties like **InputOffset**, **InputOffsetRange**, **InputRange**, **InputType**, and **OutputExcitation**.

It also provides access to the external board methods like **XbPinConfigGet** and **XbPinConfigSet**. If **ExternalBoard** is not addressed, these properties and methods are inaccessible.

**See Also**

**TExternalBoard** object
**FileFlagsAttributes property**

The `FileFlagsAttributes` property specifies the file attributes.

**Applies To**

`TServerDiskLog`

**Declaration**

```pascal
property FileFlagsAttributes: TFileFlagsAttributes;

TFileFlagsAttributes = set of TFileFlagsAttributesEnum;

TFileFlagsAttributesEnum = ( 
  ffaAttributeNormal,
  ffaAttributeEncrypted,
  ffaFlagWriteThrough,
  ffaFlagSequentialScan
);
```

**Default**

`[ffaAttributeNormal]`

**Access Restrictions**

None

**Remarks**

`FileFlagsAttributes` specifies the file attributes of the disk log file. It is of type `TFileFlagsAttributes`, which is a set of `TFileFlagsAttributesEnum`. `FileFlagsAttributes` can be manipulated using the set operators.

`FileFlagsAttributes` specifies one or more of the following options.

- **ffaAttributeNormal**: No special attributes.
- **ffaAttributeEncrypted**: The data in the file is encrypted.
- **ffaFlagWriteThrough**: Write through any intermediate caching and go directly to disk.
- **ffaFlagSequentialScan**: Can be used to optimize the transfer of large blocks of data. Most applications will not need this flag.
See Also

ServerDiskLog property, TServerDiskLog object
**FileName property**

The *FileName* property specifies the file name.

**Applies To**  
*TServerDiskLog*

**Declaration**

```pascal
property FileName: string;
```

**Default**

Empty string

**Access Restrictions**

None

**Remarks**

*FileName* specifies the name of the primary disk log file and the name of a possible mirror disk log file, if mirror logging is enabled. The default is an empty string.

Mirror logging is enabled by selecting the *dlfMirrorLog* flag of the *Flags* property. Multiple file names are separated by semi-colons. Currently, only one mirror file is allowed. Both files must be on the same side of the DAPcell Local/DAPcell service (the PC application side or the iDSC side).

**See Also**

*ServerDiskLog* property, *TServerDiskLog* object
**FileShareMode property**

The **FileShareMode** property specifies the file share properties.

**Applies To**

TServerDiskLog

**Declaration**

```pascal
property FileShareMode: TFileShareMode;
TFileShareMode = set of TFileShareModeEnum;
TFileShareModeEnum = (
    fsmNone,
    fsmRead,
    fsmWrite,
    fsmReadWrite
);
```

**Default**

[fsmRead]

**Access Restrictions**

None

**Remarks**

**FileShareMode** specifies the file share mode of the disk log file. It is of type TFileShareMode, which is a set of TFileShareModeEnum. **FileShareMode** can be manipulated using the set operators.

**FileShareMode** specifies one or more of the following options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsmNone</td>
<td>The file cannot be used by another process.</td>
</tr>
<tr>
<td>fsmRead</td>
<td>The file can be read by another process.</td>
</tr>
<tr>
<td>fsmWrite</td>
<td>The file can be written to by another process.</td>
</tr>
<tr>
<td>fsmReadWrite</td>
<td>The file can be read and written to by another process.</td>
</tr>
</tbody>
</table>

**See Also**

ServerDiskLog property, TServerDiskLog object
**FilterDesign property**

The `FilterDesign` property specifies the currently active filter design through the `TFilterDesign` object.

**Applies To**

`TDsc`

**Declaration**

```
property FilterDesign: TFilterDesign;
```

**Access Restrictions**

Read only; Run time only

**Remarks**

`FilterDesign` provides access to the filter design properties like `Attenuation`, `CutoffFreqHigh`, `CutoffFreqLow`, `CutoffSlopeHigh`, `CutoffSlopeLow`, `FilterName`, `FilterType`, `PinToFilterMap`, and `Sharpness`.

It also provides access to the filter design methods like `FilterIndex`, `FilterParametersGet`, `FilterParametersSet`, `TransferFunctionGet`, `UnitStepGet`, and `UnitStepLengthGet`. If `FilterDesign` is not addressed, these properties and methods are inaccessible.

**See Also**

`TFilterDesign` object
**FilterIndex method**

The FilterIndex method returns the filter index given a filter name.

**Applies To**

TFilterDesign

**Declaration**

```pascal
function FilterIndex(
    sFilterName: string // Filter name.
): integer;
```

**Parameters**

sFilterName

Filter name string.

**Return Values**

If the function succeeds, the return value is the filter index. If the function fails, the return value is -1. For example, if `sFilterName` is an invalid name, failure occurs.

If the filter name of interest is associated with filter index 0, the return value is 0.

**Remarks**

FilterIndex determines the filter index associated with a particular filter name. Since there are a maximum of eight filter designs, valid filter indices are 0 through 7.

The filter index is useful in methods like FilterParametersGet and FilterParametersSet.

FilterIndex is a TFilterDesign method.

**See Also**

FilterParametersGet method, FilterParametersSet method, TFilterDesign object
**FilterName property**

The **FilterName** property is an array property that specifies the filter name associated with a filter index.

**Applies To**

`TFilterDesign`

**Declaration**

```pascal
property FilterName[iFilterIndex: integer]: string;
```

**Parameters**

- **iFilterIndex**
  Filter index of interest. Valid filter indices are 0 through 7.

**Default**

`FD0, FD1, FD2, FD3, FD4, FD5, FD6, FD7`

**Access Restrictions**

Run time only

**Remarks**

*FilterName* allows the user to specify a unique filter name for an associated filter index. If an already existing filter name is assigned, the existing filter name will not change. The filter name is restricted to 63 characters.

Valid filter indices are 0 through 7. If a filter index is invalid, the exception ‘Filter index is out of range’ is raised.

*FilterName* is a `TFilterDesign` property.

**See Also**

`ConfigDialogShow` method
FilterParametersGet method

The FilterParametersGet method gets the filter parameters associated with a filter index. The filter parameters include the name, type, sharpness, cutoff frequency, cutoff slope, and attenuation.

Applies To

TFilterDesign

Declaration

procedure FilterParametersGet(
    iFilterIndex: integer; // Filter index.
    var pFilterParam: TFilterParam // Information record.
); 

Parameters

iFilterIndex
    Filter index of interest. Valid filter indices are 0 through 7.

pFilterParam
    A TFilterParam record that receives the filter parameters. TFilterParam must be initialized using StructPrepare.

Remarks

FilterParametersGet returns the filter parameters for an associated filter index in the TFilterParam record. TFilterParam must be initialized using StructPrepare before invoking FilterParametersGet or the method will fail. Valid filter indices are 0 through 7. If the filter index is invalid, the exception ‘Filter index is out of range’ is raised.

FilterParametersGet is a TFilterDesign method.

See Also

ConfigDialogShow method, FilterParametersSet method, TFilterParam type
**FilterParametersSet method**

The FilterParametersSet method sets the filter parameters associated with a filter index. The filter parameters include the name, type, sharpness, cutoff frequency, cutoff slope, and attenuation.

**Applies To**

TFilterDesign

**Declaration**

```
procedure FilterParametersSet(
  iFilterIndex: integer; // Filter index.
);
```

**Parameters**

- **iFilterIndex**
  Filter index of interest. Valid filter indices are 0 through 7.

- **pFilterParam**
  A TFilterParam record that passes the filter parameters. TFilterParam must be initialized using StructPrepare.

**Remarks**

FilterParametersSet modifies the filter parameters for an associated filter index. All members of TFilterParam must be set before invoking FilterParametersSet.

TFilterParam must be initialized using StructPrepare before invoking FilterParametersSet or the method will fail. The achName, iFilterType, iSharpness, fCutoffFreqLow, fCutoffSlopeLow, fCutoffFreqHigh, fCutoffSlopeHigh, and fAttenuation members of TFilterParam must be set to the new desired values. Note that the filter parameters can be graphically set in ConfigDialogShow.

Valid filter indices are 0 through 7. If the filter index is invalid, the exception ‘Filter index is out of range’ is raised.

FilterParametersSet is a TFilterDesign method.
See Also

`ConfigDialogShow` method, `FilterParametersGet` method, `TFilterParam` type
**FilterType property**

The `FilterType` property is an array property that specifies the filter type associated with a filter index.

**Applies To**

`TFilterDesign`

**Declaration**

```pascal
property FilterType[iFilterIndex: integer]: TFilterType;
TFilterType = (  
  fLowPass,          // Lowpass filter.  
  fBandPass          // Bandpass filter.  
);  
```

**Parameters**

`iFilterIndex`

Filter index of interest. Valid filter indices are 0 through 7.

**Default**

`fLowPass`

**Access Restrictions**

Run time only

**Remarks**

`FilterType` allows the user to specify the filter type for an associated filter index. The filter type is `fLowPass` for a lowpass filter and `fBandPass` for a bandpass filter.

Valid filter indices are 0 through 7. If a filter index is invalid, the exception ‘Filter index is out of range’ is raised.

`FilterType` is a `TFilterDesign` property.
See Also

ConfigDialogShow method
Flags property

The Flags property controls the disk logging behavior.

Applies To
TServerDiskLog

Declaration
property Flags: TDiskLogFlags;

TDiskLogFlags = set of TDiskLogFlagsEnum;

TDiskLogFlagsEnum = ( dlfServerSide, dlfFlushBefore, dlfFlushAfter, dlfMirrorLog, dlfAppendData, dlfBlockTransfer );

Default
[dlfServerSide, dlfFlushBefore]

Access Restrictions
None

Remarks
Flags controls the disk logging behavior. It is of type TDiskLogFlags, which is a set of TDiskLogFlagsEnum. Flags can be manipulated using the set operators.

Flags specifies one or more of the following options.

dlfServerSide
Log on the same side of the network connection as the iDSC. If not specified, logging will take place on the application (client) side of the network connection.
dlfFlushBefore
Flush the input data pipe before beginning the logging session. Default action is to not flush the pipes before logging.

dlflFlushAfter
Flush the input pipe after the logging session has terminated. Default action is to not flush the pipes after logging.

dlflMirrorLog
Enable mirror logging. Mirror logging creates a copy of the logged data in another file.

dlflAppendData
Allow new data to be appended to an existing file. Only the ofOpenAlways and ofOpenExisting flags of the OpenFlags member can be used for appending.

dlflBlockTransfer
Open the file with no intermediate buffering or caching and access the file in a special way that is highly dependent on the target disk attributes to improve performance. This transfer mode adds overhead to slow rate transfer with small buffers. It should only be used when necessary with a very large BlockSize value (such as 1048576 and above). If this option is selected, BlockSize is automatically set to 1048576.

See Also
ServerDiskLog property, OpenFlags property, BlockSize property,
TServerDiskLog object
**GroupDelay property**

The `GroupDelay` property returns the group delay in seconds for all of the filter designs.

**Applies To**

`TDsc`

**Declaration**

```property
GroupDelay: single;
```

**Default**

Dependent on sample rate

**Access Restrictions**

Read only, Run time only

**Remarks**

`GroupDelay` informs a user of the group delay (in seconds) through all the filter designs. The group delay is the amount of time to wait before data start showing up at the PC, and does not include the filter designs of disabled input pins.

If the iDSC board configuration or filter designs change and `StartAcquiring` is selected, `StartAcquiring` automatically invokes `CommandsLoad` and downloads new commands to the iDSC board. Data will show up at the PC two times `GroupDelay` seconds later.

If the iDSC board configuration or filter designs change and `CommandsLoad` is selected before `StartAcquiring`, data will show up immediately at the PC. Data that show up immediately are actually data that were sampled `GroupDelay` seconds ago.

**See Also**

`Calibrate` method, `CommandsLoad` method, `StartAcquiring` method
**HardwareStop method**

The `HardwareStop` method stops the hardware on the iDSC board.

**Applies To**

TDsc

**Declaration**

```delphi
procedure HardwareStop;
```

**Remarks**

`HardwareStop` stops and resets the hardware on the iDSC board.

**See Also**

Running property
**InputOffset property**

The `InputOffset` property is an array property that configures the input offset voltage on the external board.

**Applies To**

`TExternalBoard`

**Declaration**

```
property InputOffset[iPinIndex: integer]: single;
```

**Parameters**

`iPinIndex`

Pin index of interest. Valid pin indices are 0 through 7.

**Default**

0.0

**Access Restrictions**

Run time only

**Remarks**

`InputOffset` specifies the input offset voltage on the external board. The input offset of the signal is specified in Volts. The input offset must be within the range of the input offset range. If you specify an invalid input offset, the input offset will not change from its previous setting. The default is 0.0.

Valid pin indices are 0 through 7. If the pin index is not valid, the exception ‘Pin index is out of range’ is raised.

`InputOffset` is a `TExternalBoard` property.

**See Also**

`TXbPinConfig` type
**InputOffsetRange property**

The **InputOffsetRange** property is an array property that configures the input offset range voltage on the external board.

**Applies To**

**TExternalBoard**

**Declaration**

```pascal
class TExternalBoard
begin

  property InputOffsetRange[iPinIndex: integer]: single;

end;
```

**Parameters**

- **iPinIndex**
  
  Pin index of interest. Valid pin indices are 0 through 7.

**Default**

Dependent of the input range

**Access Restrictions**

Read only; Run time only

**Remarks**

The **InputOffsetRange** specifies the input offset range voltage on the external board. The input offset range of the signal is specified in Volts. The input offset range is determined by the input range. For example, if the input range is 0.5 then the input offset range is 2.5 for +/-2.5 V, if the input range is 2.0 then the input offset range is 1.0 for +/- 1V. This is a read only property that is dependent on the input range and you cannot specify it.

Valid input offset ranges are:

- 0.5 when the input range is 0.01
- 1.0 when the input range is 0.02
- 2.5 when the input range is 0.05
- 0.5 when the input range is 0.1
- 1.0 when the input range is 0.2
- 2.5 when the input range is 0.5
- 1.0 when the input range is 1.0
1.0 when the input range is 2.0
5.0 when the input range is 5.0
5.0 when the input range is 10.0

Valid pin indices are 0 through 7. If the pin index is not valid, the exception ‘Pin index is out of range’ is raised.

`InputOffsetRange` is a `TExternalBoard` property.

**See Also**

`TXbPinConfig` type
InputRange property

The InputRange property configures the input range voltage on the iDSC board.

Applies To
TDsc

Declaration
property InputRange: integer;

Default
5000

Access Restrictions
None

Remarks
InputRange configures the input range voltage on the iDSC board. A user can select either +/- 5 Volts or +/- 10 Volts.

To use this property the voltages must be specified as absolute values in millivolts. For example, 5000 represents +/- 5 Volts and 10000 represents +/- 10 Volts.

If the input range is not valid, the exception ‘Input range of xxx is invalid’ is raised where xxx represents the invalid input range.

See Also
ConfigDialogShow method
InputRange property

The InputRange property is an array property that configures the input range voltage on the external board.

Applies To
TEexternalBoard

Declaration
property InputRange[iPinIndex: integer]: single;

Parameters
iPinIndex
Pin index of interest. Valid pin indices are 0 through 7.

Default
10.0

Access Restrictions
Run time only

Remarks
InputRange specifies the input range voltage on the external board. The input range of the signal is specified in Volts. For example, if you select 0.5 the input range is +/- 500 mV, if you select 2.0 the input range +/- 2V. If you specify an invalid input range, the input range will not change from its previous setting. The default is 10.0.

Valid input ranges are:
0.01, 0.02, 0.05,
0.1, 0.2, 0.5,
1.0, 2.0, 5.0,
10.0

Valid pin indices are 0 through 7. If the pin index is not valid, the exception ‘Pin index is out of range’ is raised.

InputRange is a TExternalBoard property.
See Also

TXbPinConfig type
**InputType property**

The *InputType* property is an array property that configures the input type on the external board.

**Applies To**

*TExternalBoard*

**Declaration**

```pascal
property InputType[iPinIndex: integer]: TInputType;
```

```pascal
TInputType = (  
  itDCCoupling,  // DC coupling.
  itACCoupling,  // AC coupling.
  itExcitation,  // Excitation voltage.
);  
```

**Parameters**

*iPinIndex*

Pin index of interest. Valid pin indices are 0 through 7.

**Default**

*itDCCoupling*

**Access Restrictions**

Run time only

**Remarks**

*InputType* specifies the input type on the external board. The type of input signal includes DC coupling, AC coupling or excitation.

**See Also**

*ExternalBoard* property
Master property

The **Master** property specifies the Master iDSC board through the **TDsc** object. It is only useful in a Master/Slave Configuration.

**Applies To**

**TDsc**

**Declaration**

**property** Master: TDsc;

**Default**

Nil

**Access Restrictions**

None

**Remarks**

**Master** allows a user to designate one or more Master iDSC boards on a system. When the Master iDSC boards are designated, the Slave iDSC boards are automatically defined. **Master** will automatically update the operational state of the iDSC board in **OperateMode**.

For example, imagine two iDSC boards on a system, DSC1 and DSC2. To make DSC1 a Master iDSC board and DSC2 a Slave iDSC board, select DSC2 and hook up DSC1 as its **Master**. DSC1 is now a Master and DSC2 is now a Slave. The **OperateMode** for DSC1 is **omMaster** and for DSC2 is **omSlave**.

**See Also**

**OperateMode** property, **RemoteMaster** property, **TDsc** object
MaxCount property

The MaxCount property specifies the file maximum count.

Applies To
TServerDiskLog

Declaration

```plaintext
property MaxCount: TDscIoInt64;

TDscIoInt64 = packed record // For environments that do not
  dwLowPart: DWORD; // support 64-bit integers.
  dwHighPart: DWORD;
end;

TDscIoInt64 = int64; // For environments that support
// 64-bit integers.
```

Default

0

Access Restrictions

None

Remarks

MaxCount specifies the maximum number of bytes to log. The default is 0, which
causes logging to continue indefinitely until StopAcquiring is invoked.

See Also

ServerDiskLog property, TServerDiskLog object
MemoryUsed property

The MemoryUsed property displays the used memory on the iDSC board.

Applies To

TDsc

Declaration

property MemoryUsed: integer;

Access Restrictions

Read only; Run time only

Remarks

MemoryUsed displays the used memory on the iDSC board in tenths of a percent. As an example, if MemoryUsed returns 42, it means that 4.2% of memory is used. MemoryUsed is useful in determining whether or not the iDSC board will be able to sustain a particular sample rate without overflowing.
**OnAfterNumDscChange event**

The `OnAfterNumDscChange` event runs after the number of iDSC boards change.

**Applies To**

`TDscGroup`

**Declaration**

```plaintext
property OnAfterNumDscChange: TNotifyEvent;

TNotifyEvent = procedure (Sender: TObject) of object;
```

**Remarks**

`OnAfterNumDscChange` is useful for executing user specific code after the number of iDSC boards change, whether the number increases or decreases. The user can increase the number of iDSCs using `AddOne` and decrease the number of iDSCs using `DeleteOne`.

**See Also**

`OnBeforeNumDscChange` event, `TDscGroup` type
**OnBeforeNumDscChange event**

The *OnBeforeNumDscChange* event runs before the number of iDSC boards change.

**Applies To**

*TDscGroup*

**Declaration**

```pascal
property OnBeforeNumDscChange: TNotifyEvent;

TNotifyEvent = procedure (Sender: TObject) of object;
```

**Remarks**

*OnBeforeNumDscChange* is useful for executing user specific code before the number of iDSC boards change, whether the number increases or decreases. The user can increase the number of iDSCs using *AddOne* and decrease the number of iDSCs using *DeleteOne*.

**See Also**

*OnAfterNumDscChange* event, *TDscGroup* type
OnCalibrateProgress event

The **OnCalibrateProgress** event runs while the **Calibrate** method is executing to inform the user of the progress of calibration.

**Applies To**

TDsc

**Declaration**

property OnCalibrateProgress: TProcCalibrateProgressEvent;

TProcCalibrateProgressEvent = procedure (Sender: TObject; iProgress: integer) of object;

**Remarks**

**OnCalibrateProgress** is useful in determining the progress of calibration while the **Calibrate** method is executing. **OnCalibrateProgress** is of type TProcCalibrateProgressEvent. The parameter, **iProgress**, specifies the progress of iDSC calibration. The user can use the **iProgress** parameter in a progress bar, status bar, etc. to display the progress of calibration.

The **OnCalibrateProgress** event runs while the **Calibrate** method executes. **iProgress** is measured as a percentage from 0 to 100. A value of 0 means that calibration has not begun. A value of 100 means that calibration is complete.

**See Also**

**Calibrate** method
OnHardwareDelayChange event

The OnHardwareDelayChange event runs if the hardware delay constant changes.

Applies To

TDsc

Declaration

property OnHardwareDelayChange: TProcEvent;

TProcEvent = procedure (Sender: TObject) of object;

Remarks

OnHardwareDelayChange is useful in determining if the hardware delay constant has changed. The new hardware delay constant can then be displayed correctly.

The OnHardwareDelayChange event runs when the iDSC is first initialized, which happens the first time Calibrate, XbCalibrate, CommandsLoad, or StartAcquiring is called.

See Also

StartAcquiring method
**OnInputRangeUpdate event**

The **OnInputRangeUpdate** event runs when the input range is updated.

**Applies To**

**TDsc**

**Declaration**

```pascal
property OnInputRangeUpdate: TProcInputRangeUpdateEvent;

TProcInputRangeUpdateEvent = procedure (
    Sender: TObject;
    iInputRange: integer
) of object;
```

**Remarks**

**OnInputRangeUpdate** is useful in determining when the input range has changed. The new input range can then be used correctly. **OnInputRangeUpdate** is of type **TProcInputRangeUpdateEvent**. The parameter, **iInputRange**, specifies the supported input range of the iDSC.

The **OnInputRangeUpdate** event runs when the user changes the input range using **ConfigDialogShow** or **InputRange**.

Different models of the iDSC support different ranges, so it is advisable to check **InputRange** after setting it. **InputRange** returns the same value as **iInputRange**.

**See Also**

**InputRange** property
**OnPinEnabledUpdate event**

The **OnPinEnabledUpdate** event runs when the enabled input pins are updated.

**Applies To**

**TDsc**

**Declaration**

```pascal
property OnPinEnabledUpdate: TProcPinEnabledUpdateEvent;

TProcPinEnabledUpdateEvent = procedure (  
  Sender: TObject;  
  NewPins: TInputPins  
) of object;
```

**Remarks**

**OnPinEnabledUpdate** is useful in determining when the enabled input pins have changed. The new enabled input pins can then be used correctly. **OnPinEnabledUpdate** is of type **TProcPinEnabledUpdateEvent**. The parameter, **NewPins**, specifies the enabled input pins.

The **OnPinEnabledUpdate** event runs when the user enables or disables an input pin using **ConfigDialogShow** or **PinEnabled**.

**See Also**

**PinEnabled** property
**OnSystemError event**

The *OnSystemError* event runs when the system encounters a failure.

**Applies To**
TDsc

**Declaration**

```delphi
property OnSystemError: TProcSystemErrorEvent;

TProcSystemErrorEvent = procedure (Sender: TObject;
   sError: string)
  of object;
```

**Remarks**

*OnSystemError* is useful in determining if the system has encountered a failure since it provides a way to access the errors. *OnSystemError* is of type TProcSystemErrorEvent. The parameter, *sError*, displays an error message when a system error occurs. The user can use the *sError* parameter to display the error message in an error handling routine.

The *OnSystemError* event runs when the *SystemErrorProcess* method is invoked. *SystemErrorProcess* is automatically invoked during *BufferGetEx* if there are errors from the iDSC. The user can also call the *SystemErrorProcess* method and invoke the *OnSystemError* event if errors from the iDSC are suspected.

An example of a system error is an input channel pipe overflow when the sample rate of the iDSC board is too high for the PC to keep up.

**See Also**

*SystemErrorProcess* method
**OpenFlags property**

The *OpenFlags* property specifies the file open options.

**Applies To**

*TServerDiskLog*

**Declaration**

```pascal
property OpenFlags: TOpenFlags;

TOpenFlags = (
    ofCreateNew,
    ofCreateAlways,
    ofOpenAlways,
    ofOpenExisting
);
```

**Default**

*ofCreateAlways*

**Access Restrictions**

None

**Remarks**

*OpenFlags* specifies the file open options of the disk log file. It is of type *TOpenFlags*.

*OpenFlags* specifies one of the following options.

- **ofCreateNew**
  - Create a new file. Creation fails if the file already exists.
- **ofCreateAlways**
  - Create a new file. If the file already exists, it is overwritten.
- **ofOpenAlways**
  - Open an existing file. If the file does not exist, it will be created.
- **ofOpenExisting**
  - Open an existing file without resetting permissions. Opening fails if the file does not exist.
See Also

ServerDiskLog property, TServerDiskLog object
**OperateMode property**

The `OperateMode` property specifies the operational mode of the iDSC board. Valid operational modes are Normal, Master, or Slave.

### Applies To

`TDsc`

### Declaration

```pascal
property OperateMode: TOperateMode;

TOperateMode = (  
  omNormal,       // Normal iDSC board.  
  omMaster,       // Master iDSC board.  
  omSlave         // Slave iDSC board.  
);
```

### Default

`omNormal`

### Access Restrictions

None

### Remarks

`OperateMode` specifies the operational mode of the iDSC board. It is of type `TOperateMode`. `OperateMode` supports three operational modes: `omNormal`, `omMaster`, and `omSlave`. However, it is only provided to allow the user to select a Normal iDSC board.

`OperateMode` cannot be used to specify Master and Slave iDSC boards. Instead, the `Master` property should be used to set up Master and Slave iDSC boards. `Master` will automatically update `OperateMode` to the correct state of either `omMaster` or `omSlave`.

If the master has to support synchronization across PCs, `RemoteMaster` must be set.

---

342 DSC Component Programmer's Interface
See Also

Master property, RemoteMaster property
**OutputExcitation property**

The `OutputExcitation` property is an array property that configures the output excitation voltage on the external board.

**Applies To**

`TExternalBoard`

**Declaration**

```pascal
property OutputExcitation[iPinIndex: integer]: single;
```

**Parameters**

`iPinIndex`

Pin index of interest. Valid pin indices are 0 through 7.

**Default**

0.0

**Access Restrictions**

Run time only

**Remarks**

`OutputExcitation` specifies the output excitation voltage on the external board. The output excitation of the signal is specified in Volts. If you specify an invalid output excitation, the output excitation will not change from its previous setting. The default is 0.0.

Valid output excitation ranges are:

0.0, 1.0, 2.0, 5.0, 10.0

Valid pin indices are 0 through 7. If the pin index is not valid, the exception ‘Pin index is out of range’ is raised.

`OutputExcitation` is a `TExternalBoard` property.

**See Also**

`TXbPinConfig` type
**PinEnabled property**

The `PinEnabled` property specifies the set of enabled input pins on the iDSC board.

**Applies To**

`TDsc`

**Declaration**

```properties
property PinEnabled: TInputPins;
TInputPins = set of TInputPin;
TInputPin = (A0, A1, A2, A3, A4, A5, A6, A7);
```

**Default**

`[A0, A1, A2, A3, A4, A5, A6, A7]`

**Access Restrictions**

None

**Remarks**

`PinEnabled` specifies the enabled input pins. It is of type `TInputPins`, which is a set of `TInputPin`. `PinEnabled` can be manipulated using the `set` operators.

`PinEnabled` is configured from a `set` of eight input pin options `[A0, A1, A2, A3, A4, A5, A6, A7]`. Empty sets `[]` of input pins are not allowed. To find out the number of enabled input pins, use the `PinEnabledCount` property.

**See Also**

`PinEnabledCount` property
**PinEnabledCount property**

The `PinEnabledCount` property returns the number of enabled input pins on the iDSC board.

**Applies To**

TDsc

**Declaration**

```pascal
property PinEnabledCount: integer;
```

**Default**

Eight

**Access Restrictions**

Read only; Run time only

**Remarks**

`PinEnabledCount` returns the number of enabled input pins on the iDSC board. It is related to the `PinEnabled` property. Valid values for the enabled input pin count are in the range one to eight.

**See Also**

`PinEnabled` property
**PinToFilterMap property**

The *PinToFilterMap* property is an array property that specifies the mapping of a filter index to an input pin index.

**Applies To**

*TFilterDesign*

**Declaration**

```
property PinToFilterMap[iPinIndex: integer]: integer;
```

**Default**

- *A0* mapped to *FD0*
- *A1* mapped to *FD1*
- *A2* mapped to *FD2*
- *A3* mapped to *FD3*
- *A4* mapped to *FD4*
- *A5* mapped to *FD5*
- *A6* mapped to *FD6*
- *A7* mapped to *FD7*, where *A* is the input pin, *FD* is the filter index

**Access Restrictions**

Run time only

**Remarks**

*PinToFilterMap* associates a filter index to a corresponding input pin index. The same filter design may be applied to several input pins.

Valid input pin indices are 0 through 7, which correspond to input pins *A0* through *A7*. Valid filter indices are 0 through 7. If an input pin index is invalid, the exception ‘Pin index is out of range’ is raised.

*PinToFilterMap* is a *TFilterDesign* property.

**See Also**

*ConfigDialogShow* method
**RemoteMaster property**

The `RemoteMaster` property allows synchronization across PCs.

**Applies To**

`TDsc`

**Declaration**

```pascal
property RemoteMaster: boolean;
```

**Default**

`False`

**Access Restrictions**

Run time only

**Remarks**

`sRemoteMaster` enables the user to synchronize multiple iDSCs across PCs. It only works if the MSXB 045 hardware is present. Please refer to the MSXB 045 hardware manual for more description on the hardware.

**See Also**

`Master` property, `OperateMode` property
**Running property**

The `Running` property returns the state of the iDSC board.

**Applies To**

`TDsc`

**Declaration**

```plaintext
property Running: boolean;
```

**Default**

False

**Access Restrictions**

Read only, Run time only

**Remarks**

`Running` specifies whether the iDSC is running. Once `StartAcquiring` is invoked, `Running` will return true. Once `StopAcquiring` is invoked, `Running` will return false.

**See Also**

`HardwareStop` method, `StartAcquiring` method, `StopAcquiring` method
**SampleRate property**

The **SampleRate** property specifies the effective sampling rate per channel on the iDSC board.

**Applies To**

**TDsc**

**Declaration**

```delphi
gtype SampleRate: integer;
```

**Default**

12800

**Access Restrictions**

None

**Remarks**

**SampleRate** specifies the effective sampling rate in samples per second per channel. The table below displays the valid sample rates arranged in octaves. There are two effective sampling rates in the highest octave.

<table>
<thead>
<tr>
<th>102400</th>
<th>153600</th>
</tr>
</thead>
<tbody>
<tr>
<td>51200</td>
<td>76800</td>
</tr>
<tr>
<td>25600</td>
<td>19200</td>
</tr>
<tr>
<td>12800</td>
<td>9600</td>
</tr>
<tr>
<td>6400</td>
<td>4800</td>
</tr>
<tr>
<td>3200</td>
<td>2400</td>
</tr>
<tr>
<td>1600</td>
<td>1200</td>
</tr>
<tr>
<td>800</td>
<td>600</td>
</tr>
<tr>
<td>400</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>150</td>
</tr>
<tr>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
</tr>
<tr>
<td>25</td>
<td>16</td>
</tr>
</tbody>
</table>

DSC Component Programmer's Interface
If an invalid sample rate is selected, the property will automatically select the closest larger sample rate.

When the user changes the `SampleRate`, the `Sharpness`, `CutoffSlopeLow`, and `Attenuation` properties of `TFilterDesign` will resort to their defaults. Only `CutoffFreqLow` will retain its previously set value if the set value is valid. If the previously set value of `CutoffFreqLow` is out of range, then `CutoffFreqLow` will also resort to its default.

**See Also**

`ConfigDialogShow` method
**ScansDiscarded property**

The `ScansDiscarded` property returns the number of scans thrown away since `CommandsLoad`.

**Applies To**

`TDsc`

**Declaration**

```
property ScansDiscarded: TDoubleLong;
```

**Access Restrictions**

Read only; Run time only

**Remarks**

`ScansDiscarded` is useful in determining the number of scans discarded since `CommandsLoad`. This information is useful in calculating timing delays.

A scan is a set of enabled input pins. For example, if there are five enabled input pins, a scan consists of five samples, if there are two enabled input pins, a scan consists of two samples.

**See Also**

`CommandsLoad` method
ServerDiskLog property

The ServerDiskLog property allows access to the server disk log configuration through the TServerDiskLog object.

Applies To
TDsc

Declaration
property ServerDiskLog: TServerDiskLog;

Access Restrictions
Read only

Remarks
ServerDiskLog provides access to the server disk log configuration properties like BlockSize, FileFlagsAttributes, FileName, FileShareMode, Flags, MaxCount, and OpenFlags.

It also provides access to the server disk log configuration methods like ServerDiskLogConfigGet and ServerDiskLogConfigSet. If ServerDiskLog is not addressed, these properties and methods are inaccessible.

See Also
TServerDiskLog object, ServerDiskLogEnabled property
ServerDiskLogBytes property

The ServerDiskLogBytes property returns the number of bytes logged to disk by the server.

Applies To

TDsc

Declaration

property ServerDiskLogBytes: TDscIoInt64;

Access Restrictions

Read only, Run time only

Remarks

ServerDiskLogBytes returns the number of bytes logged to disk by the server. It is a TDscIoInt64 type. It should be called after StartAcquiring is invoked if ServerDiskLogEnabled is true. ServerDiskLogBytes will return 0 if no data has been logged to disk.

See Also

ServerDiskLog property, ServerDiskLogEnabled property
**ServerDiskLogConfigGet method**

The `ServerDiskLogConfigGet` method gets the server disk log configuration through `TServerDiskLogConfig`.

### Applies To
- `TServerDiskLog`

### Declaration
```pascal
procedure ServerDiskLogConfigGet(  
);```

### Parameters
- `pServerDiskLog`
  - A `TServerDiskLogConfig` record that receives the server disk log configuration. `TServerDiskLogConfig` must be initialized using `StructPrepare`.

### Remarks
- `ServerDiskLogConfigGet` returns the server disk log configuration in the `TServerDiskLogConfig` record. `TServerDiskLogConfig` must be initialized using `StructPrepare` before invoking `ServerDiskLogConfigGet` or the function will fail.

  The `pszFileName` field of `TServerDiskLogConfig` must be initialized to point to the user allocated buffer. The size of the user allocated buffer must be specified using the `dwFileNameSize` field. It must include an extra space for the null terminator.

  If `dwFileNameSize` is 0, the file name is not returned in `pszFileName`. If `dwFileNameSize` is 1, only the null terminator is returned in `pszFileName`.

### See Also
- `ServerDiskLogConfigSet` method, `ServerDiskLogEnabled` property, `TServerDiskLogConfig` type
**ServerDiskLogConfigSet method**

The `ServerDiskLogConfigSet` method sets the server disk log configuration through `TServerDiskLogConfig`.

**Applies To**

`TServerDiskLog`

**Declaration**

```plaintext
procedure ServerDiskLogConfigSet(
);
```

**Parameters**

`pServerDiskLogConfig`

A `TServerDiskLogConfig` record that passes the server disk log configuration. `TServerDiskLogConfig` must be initialized using `StructPrepare`.

**Remarks**

`ServerDiskLogConfigSet` modifies the server disk log configuration through the `TServerDiskLogConfig` record. `TServerDiskLogConfig` must be initialized using `StructPrepare` before invoking `ServerDiskLogConfigSet` or the function will fail.

The `pszFileName` field of `TServerDiskLogConfig` must be initialized to point to the user allocated and initialized buffer. The `dwFileNameSize` field is not used.

It is recommended that the user invokes `ServerDiskLogConfigGet` to get the server disk log configuration defaults, updates the pertinent fields, and then invokes `ServerDiskLogConfigSet` to set the new desired values.

The DAPcell Local or DAPcell server will start a disk logging session when `StartAcquiring` is invoked only if the following has been done:

1) Under Windows Control Panel | Data Acquisition Processor | Disk I/O tab | Disk Logging:
   - Set the Default Path to a valid path (or valid paths) on the server PC
   - Set the Permission to Restricted or Normal
   - Select the Save button before closing the dialog, or changes will be lost
For more information on Default Path and Permission, refer to the DAP Service documentation by selecting the Help button, or going to the main documentation reference on the DAPtools CD.

2) Set a valid filename for pszFileName of TServerDiskLogConfig.

3) Enable ServerDiskLogEnabled.

The disk logging session will end when StopAcquiring is invoked.

The Accel32 server does not support server disk logging sessions.

See Also
ServerDiskLogConfigGet method, ServerDiskLogEnabled property, TServerDiskLogConfig type
**ServerDiskLogEnabled property**

The `ServerDiskLogEnabled` property enables or disables the state of the server disk log option.

**Applies To**

`TDsc`

**Declaration**

```vbnet
property ServerDiskLogEnabled: boolean;
```

**Default**

False

**Access Restrictions**

None

**Remarks**

`ServerDiskLogEnabled` allows enabling or disabling the state of the server disk log option. If `ServerDiskLogEnabled` is true, the server will log data to disk when `StartAcquiring` is invoked. If `ServerDiskLogEnabled` is false, the server will not log any data to disk when `StartAcquiring` is invoked.

**See Also**

`ServerDiskLog` property, `ServerDiskLogBytes` property
Sharpness property

The Sharpness property is an array property that specifies the sharpness of the filter.

Applies To
TFilterDesign

Declaration
property Sharpness[iFilterIndex: integer]: integer;

Parameters
iFilterIndex
Filter index of interest. Valid filter indices are 0 through 7.

Default
Dependent on SampleRate
153600 s/s: 37
102400 s/s: 119
76800 s/s: 95
51200 s/s, 10240 s/s, 2048 s/s, 1024 s/s: 195
All others: 137

Access Restrictions
Run time only

Remarks
Sharpness specifies the sharpness of the corner frequency response. It is specified as an odd number. Valid numbers are in the range 37 to 255, depending on the sample rate.

Valid filter indices are 0 through 7. If the filter index is not valid, the exception ‘Filter index is out of range’ is raised.

Sharpness is a TFilterDesign property.
See Also

`ConfigDialogShow` method, `UnitStepLengthGet` method, `UnitStepGet` method
**SlaveCount property**

The `SlaveCount` property returns the number of slaves attached to a Master iDSC board. It is only useful in a Master/Slave Configuration.

**Applies To**

`TDsc`

**Declaration**

```delphi
property SlaveCount: integer;
```

**Default**

N/A

**Access Restrictions**

Read only

**Remarks**

`SlaveCount` only applies to a Master iDSC board. It is zero for a Normal iDSC board or Slave iDSC board.

**See Also**

`Master` property
**SlaveName method**

The `SlaveName` method returns the name of a Slave iDSC board that is attached to a Master iDSC board, given a slave index. It is only useful in a Master/Slave Configuration.

**Applies To**

`TDsc`

**Declaration**

```pascal
function SlaveName(
  iSlaveIndex: integer // Slave index.
): string
```

**Parameters**

`iSlaveIndex`

Slave index of interest. Valid slave indices are from 0 through `SlaveCount` - 1.

**Return Values**

If the function succeeds, the return value is the slave name for the given slave index. If the function fails, the return value is an error message string.

**Remarks**

`SlaveName` should only be called on a Master iDSC board. When passed in a valid slave index, it returns the corresponding slave name connected to the Master iDSC board.

Valid slave indices are from 0 through `SlaveCount` - 1. To get the names of all slaves attached to a Master iDSC board, the user could use a for loop that goes from 0 to `SlaveCount` - 1.

The method will return an error message string if `SlaveName` is invoked on a Slave or Normal iDSC board or if the user passes an invalid slave index to `SlaveName`.

**See Also**

`Master` property

---

362 DSC Component Programmer's Interface
StartAcquiring method

The StartAcquiring method starts the data acquiring process. Data will start showing up at the PC.

Applies To
TDsc

Declaration
procedure StartAcquiring;

Remarks
StartAcquiring starts the sampling process, which causes data to start showing up at the PC. StartAcquiring will force a Calibrate if it cannot find the saved calibration values, and a CommandsLoad if the iDSC board configuration or filter designs have changed. If the iDSC board configuration or filter designs have changed, the user may see a delay of two times GroupDelay seconds.

Once StartAcquiring is invoked, the user can use BufferGetEx to read blocks of data into buffers.

See Also
BufferGetEx method, Calibrate method, CommandsLoad method
**StopAcquiring method**

The **StopAcquiring** method stops the data acquiring process. Data will stop showing up at the PC.

**Applies To**

**TDsc**

**Declaration**

```plaintext
procedure StopAcquiring;
```

**Remarks**

**StopAcquiring** stops the sampling process. It is called after **StartAcquiring** to stop data from showing up at the PC.

**See Also**

**StartAcquiring** method
**StructPrepare method**

The **StructPrepare** method prepares structures for use with methods.

**Applies To**
- **TDsc**

**Declaration**

```pascal
procedure StructPrepare(
    var Struct; // Structure to initialize.
    ulSize: Cardinal // Size of structure.
);
```

**Parameters**
- **Struct**
  - Structure to initialize.
- **ulSize**
  - Size of structure.

**Remarks**

**StructPrepare** initializes the \( iInfoSize \) field and zeroes out all other fields of structures. It should be used with structures that have an \( iInfoSize \) field before the other fields of the structures are initialized. Since **StructPrepare** initializes the \( iInfoSize \) field, there is no need to initialize it separately.

**See Also**
- **TBufferGetEx** type, **TFilterParam** type, **TServerDiskLogConfig** type, **TXbPinConfig** type
**SystemErrorProcess method**

The **SystemErrorProcess** method processes error messages from the iDSC.

**Applies To**
**TDsc**

**Declaration**

```pascal
procedure SystemErrorProcess;
```

**Remarks**

**SystemErrorProcess** processes error messages from the iDSC if they exist, and then invokes the **OnSystemError** event. **SystemErrorProcess** is automatically invoked during the **BufferGetEx** method. The user can also call the **SystemErrorProcess** method and invoke the **OnSystemError** event if errors from the iDSC are suspected.

**See Also**

**OnSystemError** event
**TcEnabled property**

The **TcEnabled** property specifies the set of enabled timing channels on the iDSC board.

**Applies To**

TDsc

**Declaration**

```property TcEnabled: TTimingChannels;
TTimingChannels = set of TTimingChannel;
TTimingChannel = (
  Tc0,       // Timing Channel 0.
  Tc1,       // Timing Channel 1.
  TcWidth32  // 32-bit width.
);
```

**Default**

[]

**Access Restrictions**

Run time only

**Remarks**

**TcEnabled** specifies the enabled timing channels. It is of type **TTimingChannels**, which is a set of **TTimingChannel**. **TcEnabled** can be manipulated using the **set** operators.

**TcEnabled** is configured from a **set** of three timing channel options **[Tc0, Tc1, TcWidth32]**. To find out the number of enabled timing channels, use the **TcEnabledCount** property.

**Tc0** enables Timing Channel 0 and **Tc1** enables Timing Channel 1. **TcWidth32** forces the timing channel width to be 4 bytes (32-bits) regardless of the sample rate. If **TcWidth32** is not enabled, the timing channel width can be either 2 bytes or 4 bytes depending on the sample rate. To find out the actual timing channel width, use the **TcWidth** property.
See Also

TcEnabledCount property, TcMaximum property, TcWidth property
**TcEnabledCount property**

The `TcEnabledCount` property returns the number of enabled timing channels on the iDSC board.

**Applies To**

`TDsc`

**Declaration**

```property
tcEnabledCount: integer;
```

**Default**

Zero

**Access Restrictions**

Read only; Run time only

**Remarks**

`TcEnabledCount` returns the number of enabled timing channels on the iDSC board. It is related to the `TcEnabled` property. Valid values for the enabled timing channel count are in the range zero to two.

**See Also**

`TcEnabled` property
**TcMaximum property**

The `TcMaximum` property specifies the maximum timing channel value based on the sample rate.

**Applies To**

`TDsc`

**Declaration**

```plaintext
property TcMaximum: integer;
```

**Default**

Dependent on sample rate

**Access Restrictions**

Read only; Run time only

**Remarks**

`TcMaximum` specifies the maximum timing channel value which is dependent on the sample rate. `TcMaximum` can range from 128 to 2457600. The wide range of `TcMaximum` forces `TcWidth` to be either 2 bytes or 4 bytes.

**See Also**

`TcEnabled` property, `TcWidth` property
**TcWidth property**

The TcWidth property specifies the width, in bytes, of timing channel values.

**Applies To**
TDsc

**Declaration**

```pascal
property TcWidth: integer;
```

**Default**

Dependent on sample rate

**Access Restrictions**

Read only; Run time only

**Remarks**

TcWidth specifies the width, in bytes, of timing channel values. It returns either 2 for 16-bit values or 4 for 32-bit values.

TcWidth is dependent on the sample rate. The timing channel width for sample rates 8 s/s to 600 s/s are always 4 bytes. The timing channel width for sample rates 640 s/s to 153600 s/s are 2 bytes by default. The timing channel width for these higher sample rates can be forced to 4 bytes by setting TcWidth32 in the TcEnabled property.

**See Also**

TcEnabled property, TcMaximum property
**TransferFunctionGet method**

The **TransferFunctionGet** method plots the filter responses of the filter designs.

**Applies To**

**TFilterDesign**

**Declaration**

```pascal
procedure TransferFunctionGet(
    iFilterIndex: integer; // Filter index.
    iLength: integer; // Number of data points.
    var Buffer: array of double // Buffer to receive data.
);
```

**Parameters**

- **iFilterIndex**
  - Filter index of interest. Valid filter indices are 0 through 7.

- **iLength**
  - Number of data points to return.

- **Buffer**
  - Buffer for storing the transfer function data points. The buffer must be large enough to accommodate the requested *iLength* data points.

**Remarks**

**TransferFunctionGet** plots the filter responses of the filter designs. There can be up to eight filter response plots since there are up to eight filter designs.

**TransferFunctionGet** is a **TFilterDesign** method.

**See Also**

- **UnitStepLengthGet** method, **UnitStepGet** method
### UnitStepGet method

The `UnitStepGet` method plots the unit step responses of the filter designs.

**Applies To**
- `TFilterDesign`

**Declaration**

```pascal
procedure UnitStepGet(
  iFilterIndex: integer; // Filter index.
  iLength: integer; // Number of data points.
  var Buffer: array of double // Buffer to receive data.
);
```

**Parameters**

- **iFilterIndex**
  - Filter index of interest. Valid filter indices are 0 through 7.

- **iLength**
  - Number of data points to return.

- **Buffer**
  - Buffer for storing the unit step data points. The buffer must be large enough to accommodate the requested `iLength` data points.

**Remarks**

`UnitStepGet` plots the unit step responses of the filter designs. The `UnitStepLengthGet` method should be used to determine the `iLength` parameter. There can be up to eight unit step response plots since there are up to eight filter designs.

`UnitStepGet` is a `TFilterDesign` method.

**See Also**

- `UnitStepLengthGet` method, `TransferFunctionGet` method
**UnitStepLengthGet method**

The **UnitStepLengthGet** method returns the number of data points in the unit step response.

**Applies To**

*TFilterDesign*

**Declaration**

```pascal
function UnitStepLengthGet(
    iFilterIndex: integer // Filter index.
):integer;
```

**Parameters**

*iFilterIndex*

Filter index of interest. Valid filter indices are 0 through 7.

**Remarks**

**UnitStepLengthGet** returns the number of data points in the unit step response. The method should be used to determine the *iLength* parameter of **UnitStepGet**.

**UnitStepLengthGet** is a *TFilterDesign* method.

**See Also**

**UnitStepGet** method, **TransferFunctionGet** method
**XbCalibrate method**

The `XbCalibrate` method performs calibration on the external board.

**Applies To**

`TDsc`

**Declaration**

```pascal
procedure XbCalibrate(
    iRate: integer // Sample rate to use for calibration.
)
```

**Parameters**

`iRate`

Sample rate to use for external board calibration. A sample rate of 100 s/s is recommended.

**Remarks**

`XbCalibrate` is used to calibrate the external board. It is advisable to not exceed a sample rate of 1024 s/s or the calibration values may be unstable. `XbCalibrate` works only if `XbEnabled` is true, otherwise an exception is raised.

A higher sample rate allows `XbCalibrate` to execute faster but the calibration values are less accurate. A lower sample rate executes slower but the calibration values are more accurate. The recommended value of 100 s/s works well.

`XbCalibrate` is not automatically invoked if the external board calibration values are not found. The user has to explicitly call `XbCalibrate` to calibrate the external board.

**See Also**

`XbEnabled` property, `ExternalBoard` property
**XbEnabled property**

The **XbEnabled** property enables or disables the external board.

**Applies To**

TDsc

**Declaration**

```property XbEnabled: boolean;```

**Default**

False

**Access Restrictions**

None

**Remarks**

**XbEnabled** allows enabling or disabling the external board. If **XbEnabled** is true, the external board is enabled and visible in `ConfigDialogShow`. If **XbEnabled** is false, the external board is disabled and not visible in `ConfigDialogShow`.

To access the external board, **XbEnabled** must be true so that the configuration information is sent to the external board. Also **XbCalibrate** will work only if **XbEnabled** is true.

**See Also**

**XbCalibrate** method
**XbPinConfigGet method**

The `XbPinConfigGet` method gets the external board pin configuration associated with a pin index. The pin configuration includes the input type, input range, input offset, input offset range, and output excitation.

**Applies To**

`TExternalBoard`

**Declaration**

```pascal
procedure XbPinConfigGet(
  iPinIndex: integer; // Pin index.
  var pXbPinConfig: TXbPinConfig // Information record.
);
```

**Parameters**

- **iPinIndex**
  - Pin index of interest. Valid pin indices are 0 through 7.

- **pXbPinConfig**
  - A `TXbPinConfig` record that receives the pin configuration. `TXbPinConfig` must be initialized using `StructPrepare`.

**Remarks**

`XbPinConfigGet` returns the pin configuration for an associated pin index in the `TXbPinConfig` record. `TXbPinConfig` must be initialized using `StructPrepare` before invoking `XbPinConfigGet` or the method will fail.

Valid pin indices are 0 through 7. If the pin index is invalid, the exception ‘Pin index is out of range’ is raised.

`XbPinConfigGet` is a `TXbPinConfig` method.

**See Also**

`ConfigDialogShow` method, `XbPinConfigSet` method, `TXbPinConfig` type
XbPinConfigSet method

The XbPinConfigSet method sets the pin configuration associated with a pin index. The pin configuration includes the input type, input range, input offset, input offset range, and output excitation.

Applies To
TExternalBoard

Declaration
procedure XbPinConfigSet(
    iPinIndex: integer;  // Pin index.
    const pXbPinConfig: TXbPinConfig  // Information record.
);

Parameters
iPinIndex
Pin index of interest. Valid pin indices are 0 through 7.

pXbPinConfig
A TXbPinConfig record that passes the pin configuration. TXbPinConfig must be initialized using StructPrepare.

Remarks
XbPinConfigSet modifies the pin configuration for an associated pin index. All members of TXbPinConfig, except for fInputOffsetRange, must be set before invoking XbPinConfigSet.

TXbPinConfig must be initialized using StructPrepare before invoking XbPinConfigSet or the method will fail. The iInputType, fInputRange, fInputOffset, and fOutputExcitation members must be set to the new desired values. fInputOffsetRange cannot be set since it is a read only property.

Note that the pin configuration can be graphically set in ConfigDialogShow.

Valid pin indices are 0 through 7. If the pin index is invalid, the exception ‘Pin index is out of range’ is raised.

XbPinConfigSet is a TXbPinConfig method.
See Also
ConfigDialogShow method, TXbPinConfig type, XbPinConfigGet method
## Obsolete Interface

The objects, properties, and methods listed below are obsolete. They are discussed in greater detail in the following pages.

<table>
<thead>
<tr>
<th>Category</th>
<th>Dsc Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom command services</td>
<td>TDaplCCList object</td>
</tr>
<tr>
<td></td>
<td>DaplCCList property</td>
</tr>
</tbody>
</table>
DAPL Custom Command Support

Using the DAPL custom command interface

To define DAPL custom commands, the user should use DaplCCList. Each line of the custom command list is a custom command filename. Once a single or a list of custom commands have been defined for the iDSC using DaplCCList, the custom commands will be downloaded to the iDSC when the user invokes CommandsLoad or StartAcquiring if the download code in Download is set to enabled.

If the default stack size of 1024 bytes is not sufficient for the custom commands, the user can change the stack size using StackSize. The stack size should only be changed after the custom commands have been defined using DaplCCList but before CommandsLoad or StartAcquiring is invoked.

If the user wants to retrieve the defined custom commands, invoke the DaplCCList property. Download is used to get the state of the download code. StackSize is used to get the stack size of a custom command.

Below is a Delphi example, where Dsc1 of type TDsc is already on the form:

```delphi
Dsc1.DaplCCList.Text := // Define two custom
    'c:\t1.bin'#13#10 + // commands.
    'c:\t2.bin';

Dsc1.DaplCCList.StackSize[0] := 2048; // Set t1.bin stack size to 2048 bytes.
Dsc1.DaplCCList.Download := true; // Enable custom command download.
Dsc1.StartAcquiring; // Send custom command list to iDSC.
//... (other function calls) ...
```
**TDapICCList object**

The `TDapICCList` object defines the behavior of the `DapICCList` property.

**Remarks**

`TDapICCList` supports the properties below:

**Properties**

- `Download` property
- `StackSize` property
**DaplCCList property**

The *DaplCCList* property defines the DAPL custom command list for the iDSC through the *TDaplCCList* object.

**Applies To**

*TDsc*

**Declaration**

```delphi
property DaplCCList: TDaplCCList;
```

**Access Restrictions**

Read only; Run time only

**Remarks**

*DaplCCList* allows the user to define a DAPL custom command list for the iDSC. Each custom command in the list must be delimited by a carriage-return, line-feed, or both.

The DAPL custom command list is sent to the iDSC when the user invokes *CommandsLoad* or *StartAcquiring* if the download code in *Download* is enabled.

**See Also**

*TDaplCCList* object, *Download* property, *StackSize* property
Download property

The Download property indicates whether downloading the DAPL custom command list is enabled or disabled.

Applies To

TDaplCCList

Declaration

property Download: boolean;

Default

False

Access Restrictions

None

Remarks

Download allows enabling or disabling the download code to indicate if downloading the DAPL custom command list is enabled or disabled.

If the download code is enabled, the DAPL custom command list is downloaded to the iDSC when the user invokes CommandsLoad or StartAcquiring. If the download code is disabled, the DAPL custom command list is not downloaded to the iDSC.

The user should only have to download the DAPL custom command list to the iDSC once at startup.

See Also

DaplCCList property, StackSize property
**StackSize property**

The *StackSize* property is an array property that configures the stack size of the custom command identified by an index.

**Applies To**

TDaplCCList

**Declaration**

```property StackSize[INDEX: integer]: integer;```

**Default**

1024

**Access Restrictions**

None

**Remarks**

*StackSize* configures the stack size of the custom command identified by an index from the list of custom commands. If there is only one custom command in the list, the index is 0.

If the required stack size of the custom command is larger than 1024 bytes, the stack size must be set after the DAPL custom command list has been defined using **DaplCCList**. The DAPL custom command list is then sent to the iDSC with the new stack size when the user invokes **CommandsLoad** or **StartAcquiring** if **Download** is enabled.

**See Also**

**DaplCCList** property, **Download** property
11. iDSC Board Hardware Architecture

The iDSC board hardware architecture is partially defined by a set of features common to all models. Additionally, the architecture supports a number of options which differentiate various models. Among the common features that characterize the iDSC board architecture are the structure and division of labor between the analog and digital filters, the ability to self-calibrate using an on-board reference, isolation of the analog front-end from the PC ground and the ability to synchronize multiple boards for simultaneous sampling. Some of the characteristics that vary from model-to-model include the system bus (ISA and PCI), input voltage range, and maximum sample rate.

While configuration of the iDSC board hardware is accessible only through the programmer's interfaces described in an earlier section, an understanding of the architecture will help you achieve maximum performance out of the iDSC board.

Hardware Overview

The iDSC board hardware combines the best of analog and digital design to provide powerful capabilities at a very reasonable cost. Each of the eight analog front-end sections has an input buffer amplifier, an analog anti-aliasing filter, and an oversampling analog-to-digital converter. Two 100 MHz digital signal processors provide programmable lowpass filtering with extremely sharp cut-off characteristics and near-perfect phase linearity. An Intel processor handles data buffering and PC communications.

The iDSC board's exceptional versatility results from combining fixed-cutoff analog anti-alias filters with 64X oversampling sigma-delta analog-to-digital converters and configurable digital filters. The very high constant acquisition rate necessary for oversampling greatly simplifies the design of the analog anti-alias filters. Once the data is digitized, the DSPs filter and decimate the data to obtain the desired sampling rate, while preserving freedom from aliases. The DSPs also implement the user's lowpass and bandpass filter specifications. The hallmark of the iDSC board's flexibility lies in the ability to independently specify the sample rate and, on a channel-by-channel basis, the filter cutoff frequencies.

The components of the iDSC board architecture provide the necessary bandwidth for true 16-bit programmable filtering with attenuation of at least 96 dB for all frequencies above the effective Nyquist frequency. The analog-to-digital converters of the iDSC board operate internally at a fixed sampling rate of 9830400 samples/second. After decimation by 64 within the analog-to-digital converters, the
digital signal processors receive output data at a fixed sampling rate of 153600 samples/second. The digital signal processors then filter and decimate from the raw data rate of 153600 samples/second down to the effective sampling rate, if lower. The effective Nyquist frequency, the highest frequency which can be resolved by sampling at the effective sampling rate, is half of the effective sampling rate.

Features of the iDSC board include:

- Guaranteed Anti-aliasing
- Variable Sampling Rates from 8 to 153,600 Samples/Second
- Independently variable cut-off frequencies from 2% to 80% of Nyquist
- Very steep transition bands: -96dB/quarter-octave typical
- Eight simultaneous, dedicated channels
- 16-bit Resolution
- Linear Phase Response
- 300 Volts input signal isolation from PC ground
- Simple User Interface—consistent across all applications
- No programming required to configure low pass and bandpass filters

To achieve this performance, the iDSC board uses an impressive array of hardware including:

- 8 fourth order analog anti-alias filters
- 64-times oversampling 16 bit Sigma-Delta analog-to-digital converters
- A 96 MHz Intel 486 DX4 processor
- 16MB RAM on the iDSC 1816
- 4MB RAM on the iDSC 816
- Two 100 MHz Motorola DSP56303 digital signal processors
iDSC Board Block Diagram
Warm-up and Self-Calibration

The iDSC board architecture provides for manual calibration of a single on-board voltage reference. The recommended calibration interval for this reference is one year. Microstar Laboratories provides this calibration service at a nominal fee.

Using this reference, an iDSC board performs a self-calibration of offset and gain on each channel the first time it is initialized in a PC. Self-calibration can also be initiated under software control. This procedure takes well under one minute.

Best performance is obtained when self-calibration is initiated after the board reaches a stable operating temperature. The length of time required to stabilize depends greatly on the PC and the environment in which it operates. It is recommended that sampling be initiated and other operating conditions be duplicated as closely as possible for 30 minutes to 1 hour prior to initiating self calibration. If operating conditions do not vary significantly from day to day, self calibration will need to be run only infrequently thereafter. Some experimentation may be required to determine the self-calibration interval appropriate for your acquisition system.

Isolation

The iDSC board's entire analog section, including the analog-to-digital converters, is isolated from the digital section and, therefore, from the PC's ground. The analog section "floats" with respect to this ground. By providing a low-impedance ground reference with your input signals, you completely determine the ground potential of the analog section. Although all eight channels are provided with a separate ground connection, all eight grounds are joined together at the iDSC board's external J1 connector. Please contact Microstar Laboratories for alternative solutions if you have balanced differential signals for which this arrangement is unsuitable.

Components which provide optical and galvanic isolation, and the physical layout tolerances are designed to provide a minimum of 300 Volts isolation between the field (sensor) system and any power or ground rail on the PC. You should consider carefully whether it will be necessary to provide a weak path for current to flow between your sensor system and the PC ground to prevent the potential difference from exceeding this limit.

An important consideration in maintaining the 300 Volt Isolation is the spacing between the iDSC board and any board present in an adjacent slot. Below 10,000 feet mean sea-level (pressure) altitude, the minimum clearance required to maintain this level of isolation is .064 inches of free space, slightly more than the thickness of the
printed circuit boards. Failure to observe this clearance may result in extensive damage to the circuit boards and the PC.

Warning: Improper installation of equipment may result in fire, electrical shock, or other hazards.

When multiple iDSC boards are present in a single PC, their respective analog sections are floating with respect to each other except for two cases. The first, an external connection, occurs when the input signal grounds are related through the system of field sensors. The second case arises when the boards are synchronized for simultaneous sampling by means of the synchronization cable. This cable joins the analog ground planes of the boards and thus joins the grounds of the input signals. Again, please contact Microstar Laboratories for alternative solutions if this arrangement is not suitable for your application. Note that in both cases, the joined analog sections are still isolated from the digital section and PC ground.

**Simultaneous Sampling and Synchronization**

By design, the iDSC board architecture samples all eight channels simultaneously. It is possible to build larger simultaneous acquisition systems made up of iDSC boards and even mixed systems of iDSC boards and DAP boards by means of synchronization cables. In an iDSC board-only system, boards are synchronized with the MSCBL 078 cable. One software-designated board acts as the timing controller and all other boards connected by the cable are configured as subordinates to the controller. Details of `DscMasterSet` and related functions may be found in the “Programming Interfaces” section of this document.

For information on synchronizing a mixed system of iDSC boards and DAP boards, please contact Microstar Laboratories.
12. Installation

This chapter contains detailed installation instructions for the iDSC 1816 and iDSC 816 boards, and the iDSC board software. The Advanced Installation Options chapter provides additional information for the iDSC 816 in nonstandard PC configurations.

Installing an iDSC board involves the following steps:

1. Installing the iDSC 1816 / Installing the iDSC 816
2. Installing the iDSC Board Software
3. Testing the Installation

iDSC Board Handling Precautions

Static control is required for handling all electronic equipment. The iDSC board is especially sensitive to static discharge because it contains many high-speed analog and digital components. To protect the iDSC board, observe the following precautions:

- Wear a grounding strap when handling the iDSC board. If it is not possible to use a grounding strap, continuously touching a metal screw on a grounded PC offers protection.
- If it is necessary to transport the iDSC board outside of the PC, be sure to shield the iDSC board in a conductive plastic bag. If a conductive bag is not available, shield the iDSC board by wrapping it completely in aluminum foil. Do not ship or store an iDSC board in plastic peanuts without suitable shielding.

Static damage to analog components can cause subtle problems, including oscillation, increased settling time, and reduced slew rate. If you suspect that an iDSC board has been affected by static discharge, return it to Microstar Laboratories for testing, repair, and quality control.
Installing the iDSC 1816

System Hardware Requirements

The iDSC 1816 is compatible with 5V 32 bit PCI bus slots that support bus-mastering in 486/Pentium/Pentium Pro/Pentium II/Pentium III/Pentium 4/Xeon/Athlon and other PC AT compatible computers.

Installation Steps

Caution: Do not install the iDSC 1816 while the PC is on.

1. Make sure that connections with a high voltage potential are disconnected or turned off.
2. Turn off the PC and remove the PC's cover.
3. Insert the iDSC 1816 into any free PCI slot.
4. Screw down the back panel of the iDSC 1816 to the back chassis of the PC.
5. Make sure that the iDSC 1816 is properly installed before turning on the PC or connecting signals or cables to the iDSC 1816. Refer to the Isolation section in the iDSC Board Hardware Architecture chapter.

Warning: Improper installation of equipment may result in fire, electrical shock, or other hazards.

The iDSC 1816 requires approximately 17.5 Watts from the PC's power supply. If the system behaves erratically with the iDSC 1816 installed, the PC may need a larger power supply.
Installing the iDSC 816

System Hardware Requirements

The iDSC 816 is compatible with 16 bit ISA or 32 bit EISA bus slots in 486/Pentium/Pentium Pro/Pentium II/Pentium III/Pentium 4/Xeon/Athlon and other PC AT compatible computers.

Standard Configurations

The iDSC 816 is factory-configured to use interrupt 2 and I/O addresses in the range 220-22F (hexadecimal). This configuration does not conflict with most standard PC hardware. If you have nonstandard PC hardware or any installed cards that use the same interrupt vector or I/O address as the iDSC 816, please read the Advanced Installation Options chapter before preceding with installation.

Note: Some sound cards use I/O addresses in the 220-22F range. If your system has a sound card, check the configurations.

Interrupt and I/O address conflicts may cause subtle or obvious problems in your PC. If your PC does not operate properly after installing the iDSC 816, check that there are no configuration conflicts.

Installation Steps

Caution: Do not install the iDSC 816 while the PC is on.

1. Make any necessary changes to the hardware configurations. Interrupt vector and I/O address information are provided in the Advanced Installation Options chapter.
2. Make sure that connections with a high voltage potential are disconnected or turned off.
3. Turn off the PC and remove the PC's cover.
4. Insert the iDSC 816 into any free ISA slot.
5. Screw down the back panel of the iDSC 816 to the back chassis of the PC.
6. Make sure that iDSC 816 is properly installed before turning on the PC or connecting signals or cables to the iDSC 816. Refer to the Isolation section in the iDSC Board Hardware Architecture chapter.

Warning: Improper installation of equipment may result in fire, electrical shock, or other hazards.
The iDSC 816 requires approximately 17.5 Watts from the PC's power supply. If the system behaves erratically with the iDSC 816 installed, the PC may need a larger power supply.

**Several iDSC Boards**

Many iDSC boards can operate simultaneously in one PC. Running several boards in parallel increases the maximum sampling rate and the real-time processing power of a system. The Accel32/DAPcell server supports up to fourteen iDSC boards in one PC. However, the number of iDSC boards is limited by the number of available PCI/ISA slots in the PC.

**Installing the iDSC Board Software**

When you insert the DAPtools CD into your CD-ROM drive, the Microstar Laboratories Setup Launcher will automatically run. When you select the 'Getting Started' link, you will obtain instructions for installing iDSC board software.

If the Windows hardware installer asks for the INF file, you can find the MSLACOM.INF file at the root of the DAPtools CD.

You can obtain more detailed instructions for installing iDSC board software under various operating systems by selecting the 'Documentation' link, followed by 'DAPtools Basic/Accel32/Installation'.

**Testing the Installation**

To test the installation, run the DSCview.exe program. Begin acquiring data by selecting Start! from the main menu and verify that the data are showing up in the graph or table window. An error message will be displayed if there was a problem with the installation.

**Troubleshooting the iDSC 1816**

If the Accel32 Service will not start, check the host PC hardware manual to make sure that the particular PCI slot that the iDSC 1816 uses supports bus-mastering. If necessary, switch to a different slot.
Installation for the iDSC 1816 and iDSC 816 in standard hardware configurations is described in the Installation chapter of this document. This section covers iDSC 816 installation with nonstandard configurations.

Nonstandard Configurations

The iDSC board uses two resources from the host PC:

- an interrupt vector
- a range of I/O addresses

The iDSC board allows several interrupt vector and I/O address selections. The interrupt vector may be 2, 3, 4, 5, 11, or 15. When selecting an interrupt vector, note the following interrupt vectors used by standard cards:

- EGA/VGA 2
- serial port COM2 3
- serial port COM1 4
- parallel port #2 5
- hard disk controller on IBM XT 5
- parallel port #1 7
- secondary IDE controller 15

The iDSC board comes configured to use interrupt 2 since this interrupt does not conflict with most standard hardware. If you have any other cards installed, determine the interrupts that the cards use and select an iDSC board interrupt number distinct from these. The host computer may lose access to one of the serial COM ports or one of the parallel ports, depending on the selection.

To change interrupt vectors, locate the twenty-pin HOST CONFIGURE (J10) connector directly above the gold fingers on the iDSC board printed circuit board.
The six possible interrupt selections are:

<table>
<thead>
<tr>
<th>Interrupt Vector</th>
<th>Jumper</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>pin pair 9</td>
</tr>
<tr>
<td>3</td>
<td>pin pair 6</td>
</tr>
<tr>
<td>4</td>
<td>pin pair 7</td>
</tr>
<tr>
<td>5</td>
<td>pin pair 8</td>
</tr>
<tr>
<td>11</td>
<td>pin pair 5</td>
</tr>
<tr>
<td>15</td>
<td>pin pair 4</td>
</tr>
</tbody>
</table>

To change the interrupt, remove the jumper and replace it according to this table. Note that exactly one of the pin pairs 4, 5, 6, 7, 8, and 9 should be connected.

The iDSC board comes configured to use I/O addresses in the range 220-22F (hexadecimal). To change this range, change the jumpers on the HOST CONFIGURE connector. Pin pairs 1, 2, and 3 select the I/O address of the iDSC board according to the following table:

<table>
<thead>
<tr>
<th>I/O Address Range</th>
<th>Jumpers</th>
</tr>
</thead>
<tbody>
<tr>
<td>220 - 22F</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>230 - 23F</td>
<td>2, 3</td>
</tr>
<tr>
<td>240 - 24F</td>
<td>1, 3</td>
</tr>
<tr>
<td>250 - 25F</td>
<td>3</td>
</tr>
<tr>
<td>320 - 32F</td>
<td>1, 2</td>
</tr>
<tr>
<td>330 - 33F</td>
<td>2</td>
</tr>
<tr>
<td>340 - 34F</td>
<td>1</td>
</tr>
</tbody>
</table>
**Multiple Board Installation**

Up to fourteen iDSC boards can operate simultaneously in one PC. Running several boards in parallel increases the maximum sampling rate and the real-time processing power of a system.

Microstar Laboratories iDSC boards are also compatible with Microstar Laboratories Data Acquisition Processors. Any combination of up to fourteen iDSC boards and DAP boards can run as one system in a PC.

Each iDSC 816 board requires one ISA-compatible slot. All of the iDSC 816 boards and ISA-compatible DAP boards in a PC share just one interrupt line; no DMA lines are required. The iDSC 816 boards and ISA-compatible DAP boards are distinguished from one another by their I/O addresses. Before installing iDSC 816 boards in the PC, select a distinct I/O address for each board. Set the I/O addresses with the jumpers on the **HOST CONFIGURE** connector. Information on the **HOST CONFIGURE** is provided at the beginning of this chapter.

---

**Note:** Pin pair 10 of the **HOST CONFIGURE** connector sets the level of the PC’s interrupt line. Pin pair 10 must be connected for one ISA-compatible board in a PC, and must not be connected for all other iDSC 816 boards and DAP boards.

---

Once the iDSC boards have been properly installed, run the Accel32 / DAPcell server setup as described in the **Installation** chapter. It will auto-detect the iDSC board system configuration on the host machine and also auto-detect any DAP boards on the system.
This chapter describes the physical interface of the iDSC board.
**Input/Output Connector**

Analog and digital voltages connect to the iDSC board through a 50-pin connector on the back panel of the PC. This connector is located on the edge of the iDSC board and is labeled J1 IN/OUT. J1 has a double row of pins on 0.050 inch centers. The connector is AMP part number 787394-5. This connector mates with discrete wire connector T&B part number HFM050A or insulation displacement connector AMP part number 786090-5. J1 mates with Microstar Laboratories cable numbers MSCBL 047-01, MSCBL 048-01, MSCBL 049-01K, MSCBL 050-01, and MSCBL 051-01K.

Looking at the input/output connector from the back of a PC, the pin numbering is

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RESERVED</td>
</tr>
<tr>
<td>2</td>
<td>RESERVED</td>
</tr>
<tr>
<td>3</td>
<td>RESERVED</td>
</tr>
<tr>
<td>4</td>
<td>RESERVED</td>
</tr>
<tr>
<td>5</td>
<td>RESERVED</td>
</tr>
<tr>
<td>6</td>
<td>RESERVED</td>
</tr>
<tr>
<td>7</td>
<td>RESERVED</td>
</tr>
<tr>
<td>8</td>
<td>RESERVED</td>
</tr>
<tr>
<td>9</td>
<td>RESERVED</td>
</tr>
<tr>
<td>10</td>
<td>RESERVED</td>
</tr>
<tr>
<td>11</td>
<td>+5 VOLTS</td>
</tr>
<tr>
<td>12</td>
<td>RESERVED</td>
</tr>
<tr>
<td>13</td>
<td>RESERVED</td>
</tr>
<tr>
<td>14</td>
<td>RESERVED</td>
</tr>
<tr>
<td>15</td>
<td>RESERVED</td>
</tr>
<tr>
<td>16</td>
<td>RESERVED</td>
</tr>
<tr>
<td>17</td>
<td>RESERVED</td>
</tr>
<tr>
<td>18</td>
<td>RESERVED</td>
</tr>
<tr>
<td>19</td>
<td>RESERVED</td>
</tr>
<tr>
<td>20</td>
<td>RESERVED</td>
</tr>
<tr>
<td>21</td>
<td>RESERVED</td>
</tr>
<tr>
<td>22</td>
<td>RESERVED</td>
</tr>
<tr>
<td>23</td>
<td>RESERVED</td>
</tr>
<tr>
<td>24</td>
<td>RESERVED</td>
</tr>
<tr>
<td>25</td>
<td>iDSC +18V</td>
</tr>
<tr>
<td>26</td>
<td>iDSC -18V</td>
</tr>
<tr>
<td>27</td>
<td>RESERVED</td>
</tr>
<tr>
<td>28</td>
<td>RESERVED</td>
</tr>
<tr>
<td>29</td>
<td>A7+</td>
</tr>
<tr>
<td>30</td>
<td>A6+</td>
</tr>
<tr>
<td>31</td>
<td>A5+</td>
</tr>
<tr>
<td>32</td>
<td>A4+</td>
</tr>
<tr>
<td>33</td>
<td>A3+</td>
</tr>
<tr>
<td>34</td>
<td>A2+</td>
</tr>
<tr>
<td>35</td>
<td>A1+</td>
</tr>
<tr>
<td>36</td>
<td>A0+</td>
</tr>
<tr>
<td>37</td>
<td>RESERVED</td>
</tr>
<tr>
<td>38</td>
<td>RESERVED</td>
</tr>
<tr>
<td>39</td>
<td>RESERVED</td>
</tr>
<tr>
<td>40</td>
<td>RESERVED</td>
</tr>
<tr>
<td>41</td>
<td>RESERVED</td>
</tr>
<tr>
<td>42</td>
<td>RESERVED</td>
</tr>
<tr>
<td>43</td>
<td>RESERVED</td>
</tr>
<tr>
<td>44</td>
<td>RESERVED</td>
</tr>
<tr>
<td>45</td>
<td>RESERVED</td>
</tr>
<tr>
<td>46</td>
<td>RESERVED</td>
</tr>
<tr>
<td>47</td>
<td>RESERVED</td>
</tr>
<tr>
<td>48</td>
<td>RESERVED</td>
</tr>
<tr>
<td>49</td>
<td>RESERVED</td>
</tr>
<tr>
<td>50</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

Note: Use the pin numbering on this chart, rather than numbers which may be found on your connector. Connectors from different manufacturers are not numbered consistently.
Analog inputs are indicated by A0+ through A7+; their corresponding ground inputs are A0- through A7-.

External timing channel inputs are indicated by EXTERNAL TIMING CHANNEL 0 - INPUT and EXTERNAL TIMING CHANNEL 1 - INPUT. These correspond to Tc0 and Tc1 in software.

Pins 11 and 40 connect to the isolated 5 volt digital power supply. Pins 25 and 26 are +18 volt and -18 volt supplies respectively. This information is advisory only. No current is available from these pins. iDSC board performance is unspecified if current is drawn for any purpose.

Pins 1, 2, 3, 4, 5, 6, 7, 8, 9, 23, 24, 27, 28, 37, 38, 39, 42, 43, 44, 45, 46, 47, 48, 49, and 50 are indicated by RESERVED. These pins are reserved for future expansion and should not be used.
Analog Inputs

Warning: Measuring signals with a high voltage potential from the PC supply is always hazardous. The iDSC board and all cables and boards connected to the iDSC board should be considered hazardous whenever any connections are made to a high voltage potential.

Each analog input on the iDSC board has dedicated anti-alias filter and analog-to-digital conversion circuits. All inputs are optically and galvanically isolated from the PC power supplies and ground. Each iDSC 1816 channel provides:

- High input impedance
- Single-ended inputs
- Input range of ±5 Volts or ±10 Volts, software selectable
- Absolute maximum ±40 Volts relative to signal ground
- 300 Volts isolation from the PC power supply
- Simultaneous Sampling

Each iDSC 816 channel provides the same features except:
- Input range is fixed at either ±5 Volts or ±10 Volts (model dependent)
The iDSC 1816 analog input signals pass through analog multiplexers used for self-calibration and then on to the high-impedance inputs of the FET op amps. Since the multiplexers of the iDSC 1816 have channel protection built in, it does not have separate channel protectors. The DC input impedance is very high; the AC input impedance is dominated by the capacitance of the channel protector and switch or multiplexer. The iDSC 816 analog input signals pass through channel protectors and analog switches used for self-calibration. Following this they pass to the high-impedance inputs of FET op amps.

The following table shows approximate typical resistance and capacitance values for the equivalent circuit of the figure above:

<table>
<thead>
<tr>
<th>Component</th>
<th>iDSC 1816</th>
<th>iDSC 816</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>N/A</td>
<td>80Ω</td>
</tr>
<tr>
<td>R2</td>
<td>400Ω</td>
<td>25Ω</td>
</tr>
<tr>
<td>C1</td>
<td>N/A</td>
<td>5 pF</td>
</tr>
<tr>
<td>C2</td>
<td>5 pF</td>
<td>35 pF</td>
</tr>
<tr>
<td>C3</td>
<td>30 pF</td>
<td>55 pF</td>
</tr>
</tbody>
</table>
**iDSC Board Synchronization Connector**

A synchronization connector (J3 on the iDSC 1816 and J100 on the iDSC 816), allows the synchronization of several iDSC boards to deliver simultaneous sampling with respect to one other. By connecting the synchronization connector of each iDSC board using a MSCBL 078 cable available from Microstar Laboratories, the iDSC boards can be synchronized together. Up to fourteen iDSC boards can be synchronized this way. To enable master/slave synchronization in software please refer to the Programming Interfaces section of this document. This connector should only be used for synchronizing iDSC boards together. The iDSC 1816 also has an External Synchronization Connector, J2, described below.

The full part number of the cable is MSCBL 078-xx-L0.8 where xx specifies the number of slave processors. The MSCBL 078 is compatible with both the iDSC 1816 and the iDSC 816. It replaces the MSCBL 014-xx-L0.7 which is incompatible with the iDSC 1816.

**iDSC 1816 External Synchronization Connector**

Note: When using an iDSC 1816 as a remote master with an MSXB 045 synchronization board, connector J2 is used to control the MSXB 045 synchronization board. Therefore it is not available for other applications. See the documentation for the MSXB 045 synchronization board for details on using this product.

An external synchronization connector, J2, provides a means to synchronize DAPs or other measurement circuits with an iDSC 1816. This connector is not available on the iDSC 816. The synchronization signal is a 153.6 kHz word clock with a 50% duty cycle. The samples aggregated to form a single output word are collected at the 64X oversampling rate between the rising edges of this clock. The clock output has CMOS levels and can source 1mA, sinking up to 12mA.

The J2 connector is located between the PCI carrier board and the daughter card. Viewed from above with J1-end of the board closest to you, the pin numbering is

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Word Clock</td>
</tr>
<tr>
<td>2</td>
<td>Ground</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
For additional information on using this connector and/or synchronizing a mixed system of iDSC boards and DAP boards, please contact Microstar Laboratories.
Index

About This Document ................................................................................................................. 4
AddOne .................................................................................................................................... 305
AddOne method .............................................................................................................255, 273, 276, 296, 333, 334
Address property .............................................................................................................247, 253, 272, 277
Advanced Installation Options ............................................................................................ 399
Analog Inputs .................................................................................................................... 406
App01 - BASIC ................................................................................................................. 57
App02 - GRAPH ............................................................................................................... 57
App03 - LOG .................................................................................................................... 57
App04 - LOGVW ............................................................................................................. 57
App05 - DapiFFT ............................................................................................................. 57
App06 - DapiCC ............................................................................................................. 57
App07 - Disk Logging (1 iDSC) .......................................................................................... 58
App08 - Disk Logging (2 iDSC with synchronization) ...................................................... 58
App09 – A Group of iDSC ............................................................................................... 59
Attenuation ....................................................................................................................... 14, 23
Attenuation property ......................................................................................................257, 274, 278
Block Diagram .................................................................................................................. 391
BlockSize property .........................................................................................................258, 274, 280, 353
BufferAvail method .........................................................................................................253, 272, 281
BufferGet method ..........................................................................................................253, 272, 282, 284
BufferGetEnabled property .............................................................................................253, 272, 284
BufferGetEx method ......................................................................................................252, 253, 259, 272, 281, 284, 285
Calibrate method .............................................................................................................248, 253, 272, 287, 363
Channels ........................................................................................................................... 37, 42, 43, 46, 47, 48, 49
CommandsLoad ...............................................................................................................250, 303, 381, 383, 384, 385
CommandsLoad method .................................................................................................248, 253, 272, 285, 288, 321, 363
Component library .......................................................................................................... 243
ConfigDialogOptions .......................................................................................................253, 272, 289, 293
ConfigDialogOptions property ......................................................................................... 289
ConfigDialogShow .............................................................................................................289, 290, 305
ConfigDialogShow method ..............................................................................................253, 255, 272, 273, 291, 294, 315, 378
Configuration Window ..................................................................................................... 35, 37, 39
Copyrights and Trademarks ............................................................................................. i
Count property ..................................................................................................................255, 273, 276, 296, 304, 305
Creating a DSCC Application .......................................................................................... 246
Creating a DSCIO Application ......................................................................................... 106
Creating an iDSC Board Application in HP VEE ............................................................. 79
Creating an iDSC Board Application in LabVIEW ......................................................... 56
Creating an iDSC Board Application in LabWindows/CVI .............................................. 75
custom command ............................................................................................................ 232, 381
Custom Command Support ............................................................................................ 232, 381

Index
Index

CutoffFreqHigh property ................................................................. 257, 274, 297
CutoffFreqLow property ............................................................... 257, 274, 298
CutoffSlopeHigh property ............................................................ 257, 274, 299
CutoffSlopeLow property .............................................................. 257, 274, 301
DAPL custom commands .......................................................... 106, 246
DAPL Support .............................................................................. 110, 250
DAPL text .................................................................................. 106, 110, 246, 250
DapiICCList................................................................................ 381, 382, 383, 385
DapiICCList property ................................................................. 383
DapiText ................................................................................... 250, 251, 253, 273, 303
DapiText property ..................................................................... 303
DASYLab Examples ................................................................. 52
Data Format .............................................................................. 62
data track ................................................................................ 45
data tracking ........................................................................... 42
DeleteOne ................................................................................ 305
DeleteOne method ................................................................. 255, 273, 296, 304, 333, 334
Disk Log Options ................................................................. 37, 48, 49
Disk Log Options|Close File .................................................. 49
Disk Log Options|Format .............................................................. 49
Disk Log Options|Open File ..................................................... 49
Disk Log Window ................................................................... 35, 37, 48, 49
Channels.................................................................................. 49
DiskLog .................................................................................. 70
DLL MEX Reference ............................................................. 88
DLL Reference ........................................................................ 60
Download ................................................................................ 381, 382, 383, 384, 385
Download property ................................................................. 384
DSC Component ................................................................... 241
DSC Component Obsolete Interface .................................... 380
DSC Graphical Design ........................................................... 11, 39
data property ......................................................................... 255, 273, 305
DscAddressGet ..................................................................... 130, 161
DscAddressSet ...................................................................... 131, 161
DscBufferAvail ....................................................................... 132
DscBufferGet .......................................................................... 133, 137, 285
DscBufferGetEnabledGet .................................................. 135
DscBufferGetEnabledSet ..................................................... 136
DscBufferGetEx ................................................................. 113, 120, 132, 137, 199
DSCC interface .................................................................. 241
DSCC Interface Examples ................................................ 242
DSCC Introduction ............................................................... 241
DSCC UNC .............................................................................. 247
DscCalibrate .......................................................................... 108, 139, 199
DscCommandsLoad ........................................................... 108, 140, 152, 154, 159, 199, 232, 234
DscConfigDialogOptionsGet ............................................. 141, 143
DscMasterSet ................................................................. 108, 170
DscMemoryUsed ............................................................ 171
DscOnSystemErrorSet ...................................................... 120, 172
DscOperateModeGet ....................................................... 173
DscOperateModeSet ......................................................... 174
DscPinEnabledCount ....................................................... 175
DscPinEnabledGet .......................................................... 176
DscPinEnabledSet ........................................................... 177
DscPinToFilterMapGet ................................................... 179
DscPinToFilterMapSet .................................................... 180
DscRemoteMasterGet ...................................................... 181
DscRemoteMasterSet ....................................................... 182
DscRunning ................................................................. 183
DscSampleRateGet ........................................................ 184
DscSampleRateSet ........................................................ 108, 185
DscScansDiscarded ......................................................... 187
DscServerDiskLogBytes .................................................. 115, 188
DscServerDiskLogConfigGet .......................................... 121, 122, 123, 189, 191
DscServerDiskLogConfigSet .......................................... 121, 122, 123, 191
DscServerDiskLogEnabledGet ....................................... 188, 193
DscServerDiskLogEnabledSet ....................................... 192, 194, 357
DscServerDiskLogFileFileNameGet ................................. 195
DscServerDiskLogFileFileNameSet ................................ 196
DscSlaveCount ............................................................ 197, 198
DscSlaveHandle ............................................................. 197, 198
DscStartAcquiring ........................................................ 234
DscStartAcquiring ....................................................... 200
DscStopAcquiring ......................................................... 108, 110, 123, 183, 192, 200
DscStructPrepare ........................................................ 113, 123, 126, 137, 189, 191, 201
DscSystemErrorProcess ............................................... 120, 202
DscSystemErrorProcess method ..................................... 172
DscTcEnabledCount ....................................................... 203
DscTcEnabledGet .......................................................... 204
DscTcEnabledSet ........................................................... 205
DscTcMaximum .............................................................. 207
DscTcWidth ................................................................. 208
DscTransferFunctionGet ............................................... 209
DscUnitStepGet ........................................................... 210
DscUnitStepLengthGet ................................................. 211
DSCview Interface ........................................................ 35
DscXbCalibrate ............................................................. 108, 212
DscXbEnabledGet ........................................................ 213
DscXbEnabledSet ........................................................ 214
DscXbPinConfigGet ....................................................... 125, 126, 215
DscXbPinConfigSet ....................................................... 125, 126, 216
Enabled .................................................................................................................... 12, 21
Events ............................................................................................................................ 272
External Board Parameters .......................................................................................... 31
  Input Offset ............................................................................................................... 31
  Input Offset Range .................................................................................................... 31
  Input Range .............................................................................................................. 31
  Input Type .............................................................................................................. 31
  Output Excitation .................................................................................................... 32
External Board Screen .................................................................................................. 11, 16, 17
ExternalBoard property ................................................................................................. 252, 253, 273, 306
File ................................................................................................................................ 36, 37
FileFlagsAttributes property ..................................................................................... 258, 274, 307, 353
FileName property ..................................................................................................... 258, 274, 309, 353
FileShareMode property ............................................................................................. 258, 274, 310, 353
Filter Characteristics .................................................................................................. 7
Filter Design Parameters ............................................................................................. 11, 22
  Attenuation .......................................................................................................... 23
  Filter Name ............................................................................................................ 22
  Filter Type ............................................................................................................ 22
  High Cutoff Frequency ......................................................................................... 23
  High Cutoff Slope ................................................................................................. 23
  Low Cutoff Frequency ......................................................................................... 22
  Low Cutoff Slope ................................................................................................. 23
  Sharpness ........................................................................................................... 22
Filter Design Screen ................................................................................................. 11, 14, 17, 18, 21, 22, 24, 40, 145, 219, 292, 294
Filter Name ............................................................................................................. 12, 14, 21, 22
Filter Response .......................................................................................................... 24
  Linear Display ....................................................................................................... 26
  Linear Zoom Display .............................................................................................. 27
  Log Display .......................................................................................................... 28
  Log Zoom Display ................................................................................................. 29
  Unit Step Display ................................................................................................. 30
Filter Type ................................................................................................................ 14, 22
FilterDesign property ................................................................................................. 252, 253, 272, 311
FilterIndex method ................................................................................................. 257, 274, 312
FilterName property ................................................................................................. 257, 274, 313
FilterParametersGet method ..................................................................................... 252, 257, 263, 264, 274, 293, 314
FilterParametersSet method ..................................................................................... 252, 257, 263, 264, 274, 293, 315
FilterType property ................................................................................................. 257, 274, 317
Flags property ......................................................................................................... 258, 274, 309, 319, 353
Functions ............................................................................................................... 127
Get Error .................................................................................................................... 72
Graph Options ............................................................................................................ 42
Graph Options|Display .............................................................................................. 37, 43
Graph Options|Enabled ............................................................................................. 43
Graph Options|History .............................................................................................. 37, 43
Graph Window ........................................................................................................ 35, 37, 42, 43, 45
<table>
<thead>
<tr>
<th>Term</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Excitation</td>
<td>16, 32</td>
</tr>
<tr>
<td>Output File</td>
<td>49</td>
</tr>
<tr>
<td>OutputExcitation property</td>
<td>256, 273, 344</td>
</tr>
<tr>
<td>Package library</td>
<td>243</td>
</tr>
<tr>
<td>Physical Interface</td>
<td>403</td>
</tr>
<tr>
<td>PinEnabled property</td>
<td>253, 272, 338, 345</td>
</tr>
<tr>
<td>PinEnabledCount property</td>
<td>253, 272, 346</td>
</tr>
<tr>
<td>PinToFilterMap property</td>
<td>257, 274, 347</td>
</tr>
<tr>
<td>Property, Method, and Event Summary</td>
<td>241, 272</td>
</tr>
<tr>
<td>RemoteMaster property</td>
<td>248, 253, 272, 348</td>
</tr>
<tr>
<td>Running property</td>
<td>253, 272, 349</td>
</tr>
<tr>
<td>sample rate</td>
<td>118</td>
</tr>
<tr>
<td>Sample Rate</td>
<td>12, 20, 22, 23, 27, 28, 29</td>
</tr>
<tr>
<td>SampleRate property</td>
<td>248, 253, 272, 350</td>
</tr>
<tr>
<td>Save and load workspaces</td>
<td>36</td>
</tr>
<tr>
<td>ScansDiscarded property</td>
<td>253, 272, 352</td>
</tr>
<tr>
<td>Section 1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Section 2. Application Software</td>
<td>33</td>
</tr>
<tr>
<td>Section 3. Programming Interfaces</td>
<td>101</td>
</tr>
<tr>
<td>Section 4. Installation and Setup</td>
<td>387</td>
</tr>
<tr>
<td>Server Disk Log Window</td>
<td>35, 50</td>
</tr>
<tr>
<td>ServerDiskLog property</td>
<td>252, 253, 273, 353</td>
</tr>
<tr>
<td>ServerDiskLogBytes property</td>
<td>252, 261, 354</td>
</tr>
<tr>
<td>ServerDiskLogBytes property</td>
<td>253, 273, 354</td>
</tr>
<tr>
<td>ServerDiskLogConfigGet</td>
<td>267</td>
</tr>
<tr>
<td>ServerDiskLogConfigGet method</td>
<td>252, 258, 266, 268, 274, 355, 356</td>
</tr>
<tr>
<td>ServerDiskLogConfigSet</td>
<td>267</td>
</tr>
<tr>
<td>ServerDiskLogConfigSet method</td>
<td>252, 258, 266, 268, 274, 353, 356</td>
</tr>
<tr>
<td>ServerDiskLogEnabled property</td>
<td>253, 273, 354, 358</td>
</tr>
<tr>
<td>Several iDSC Boards</td>
<td>398</td>
</tr>
<tr>
<td>Sharpness</td>
<td>14, 22</td>
</tr>
<tr>
<td>Sharpness property</td>
<td>257, 274, 359</td>
</tr>
<tr>
<td>Simultaneous Sampling and Synchronization</td>
<td>393</td>
</tr>
<tr>
<td>slave</td>
<td>108, 248</td>
</tr>
<tr>
<td>SlaveCount property</td>
<td>248, 253, 272, 361, 362</td>
</tr>
<tr>
<td>SlaveName method</td>
<td>248, 253, 272, 362</td>
</tr>
<tr>
<td>Special Addressing</td>
<td>53</td>
</tr>
<tr>
<td>StackSize</td>
<td>381, 382, 383, 385</td>
</tr>
<tr>
<td>StackSize property</td>
<td>385</td>
</tr>
<tr>
<td>Start</td>
<td>36</td>
</tr>
<tr>
<td>Start!</td>
<td>36, 37, 38</td>
</tr>
<tr>
<td>StartAcquiring</td>
<td>250, 303, 381, 383, 384, 385</td>
</tr>
<tr>
<td>StartAcquiring method</td>
<td>288</td>
</tr>
<tr>
<td>Static sensitivity</td>
<td>395</td>
</tr>
<tr>
<td>Stop</td>
<td>36, 37</td>
</tr>
</tbody>
</table>
Stop!.......................................................... 36, 37
StopAcquiring.......................................................... 250
StopAcquiring method ............................................ 248, 253, 268, 272, 331, 349, 357, 364
StructPrepare.......................................................... 259
StructPrepare method .................................................. 356
StructPrepare method ............................................. 253, 264, 268, 272, 285, 355, 365
Structures ................................................................. 112
SubVIs Reference ......................................................... 65
Synchronizing Several iDSC Boards ........................................... 53
System Options .......................................................... 37
System|Address ............................................................ 37
System|Board Setup Display ......................................... 37, 41
System|Calibrate .......................................................... 37, 38
System|Commands Load ............................................... 37, 38
System|Configuration Lock ........................................... 37, 38
System|Memory Display ................................................ 37, 38
System|Memory Display ............................................... 38
System|ErrorProcess method ........................................ 253, 273, 339, 366
Table Options .............................................................. 46
Table Options|Display ..................................................... 37, 47
Table Options|Enabled ..................................................... 47
Table Options|History ...................................................... 37, 47
Table Window ............................................................. 35, 37, 46, 47
Channels ................................................................. 47
TBufferGetEx .............................................................. 112, 113, 137
TBufferGetEx method ................................................ 252, 253, 268, 272, 285, 305
TEnabled property ...................................................... 253, 257, 367
TEnabledCount property ................................................. 253, 257, 369
TMaximum property .................................................... 253, 273, 370
TWidth property ........................................................ 253, 273, 371
TDaplCCList ................................................................. 382, 383
TDaplCCList method .................................................. 382
TDaplCCList object ..................................................... 382
TDc .......................................................... 251, 261, 281
TDc method ................................................................. 252, 253, 255, 305
TDcGroup ................................................................. 252
TDcGroup method ...................................................... 252
TDcGroup object .......................................................... 255
TDcLoInt64 ................................................................. 112, 115, 123, 188, 252, 261, 268, 354
TDcLoInt64 method .................................................... 261
Testing the Installation .................................................. 395, 398
TExternalBoard ........................................................... 395, 398
TFilterDesign .............................................................. 252, 256
TFilterDesign method .................................................. 252
TFilterDesign object ................................................... 252, 257
TFilterDesign property .............................................. 297, 298, 299, 301, 312, 313, 317
Index

TFilterDesign type .......................................................... 278
TFilterParam ................................................................. 112, 117, 156, 157, 186
TFilterParam type .......................................................... 252, 263, 314, 315
TProcSystemErrorStdcall ............................................. 112, 120
TransferFunctionGet method ........................................ 257, 274, 372
Troubleshooting ............................................................. 398
TServerDiskLog object ................................................... 252, 258, 280, 353
TServerDiskLogConfig ................................................. 112, 115, 121, 189, 191, 261, 355, 356
TServerDiskLogConfig type ........................................... 252, 266, 355, 356
TXbPinConfig ............................................................... 112, 125
TXbPinConfig type ........................................................ 252, 270, 377, 378
Unit Step Display .......................................................... 15, 24, 30
UnitStepGet method ....................................................... 257, 274, 373
UnitStepLengthGet method .......................................... 257, 274, 374
Universal Naming Convention ..................................... 107, 247
Using an iDSC Board Module with DASYLab ............... 52
Using an iDSC Board with MATLAB ............................. 82
Using More Than One iDSC Board with DASYLab ........... 52
Using the iDSC Board with DASYLab ............................ 51
Using the iDSC Board with HP VEE ............................... 77
Using the iDSC Board with LabVIEW ............................ 55
Using the iDSC Board with LabWindows/CVI .................. 74
Using the iDSC Board with MATLAB ............................. 81
Warm-up and Self-Calibration ...................................... 392
workspaces .................................................................. 35
Workspaces .................................................................. 36
Xbcalibrate method ....................................................... 248, 253, 273, 375
XbEnabled property ....................................................... 253, 273, 376
XbPinConfigGet method .............................................. 252, 256, 270, 271, 274, 377
XbPinConfigSet method .............................................. 252, 256, 270, 271, 274, 378
Zoom In ........................................................................ 45
zooming ........................................................................ 42