1. 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

Modifying one of the examples described below may create a new application in LabVIEW. The following section provides an outline about how LabVIEW interfaces to the iDSC board.

An iDSC board application should performs 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, step5 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 `MslDscServerDiskLogConfigSet` . 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.

App09 – A Group of iDSC

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.

`Call Library Function` node in `DSC.LLB`	**DSCIO DLL Functions**
DscHandleOpenA	`DscHandleOpen`
DscHandleClose	`DscHandleClose`
DscCalibrate	`DscCalibrate`
DscCommandsLoad	`DscCommandsLoad`
DscStartAcquiring	`DscStartAcquiring`
DscStopAcquiring	`DscStopAcquiring`
DscBufferAvail	`DscBufferAvail`
DscBufferGet	`DscBufferGet`
DscBufferGetEx	`DscBufferGetEx`
DscConfigRead	`DscConfigRead`
DscConfigWrite	`DscConfigWrite`
DscConfigWriteSize	`DscConfigWriteSize`
DscConfigDialogShow	`DscConfigDialogShow`
DscFilterNameGet	Obsolete
DscFilterNameSet	Obsolete
DscPinToFilterMapGet	`DscPinToFilterMapGet`
DscPinToFilterMapSet	`DscPinToFilterMapSet`
DscFilterIndexA	`DscFilterIndex`
DscGroupDelay	`DscGroupDelay`
DscAddressGetA	`DscAddressGet`
DscAddressSetA	`DscAddressSet`
DscIdGetA	`DscIdGet`

DscIdSetA	DscIdSet
DscOperateModeGet	DscOperateModeGet
DscOperateModeSet	DscOperateModeSet
DscSampleRateGet	DscSampleRateGet
DscSampleRateSet	DscSampleRateSet
DscPinEnabledGet	DscPinEnabledGet
DscPinEnabledSet	DscPinEnabledSet
DscPinEnabledCount	DscPinEnabledCount
DscMasterGet	DscMasterGet
DscMasterSet	DscMasterSet
DscSlaveCount	DscSlaveCount
DscMemoryUsed (new)	DscMemoryUsed
DscLastErrorTextGetA	DscLastErrorTextGet
DscLastErrorTextSetA	DscLastErrorTextSet
DscDaplTextSetA	DscDaplTextSet
DscDaplTextLengthGet	DscDaplTextLengthGet
DscDaplTextGetA	DscDaplTextGet
DscDaplCCDownloadGet	DscDaplCCDownloadGet*
DscDaplCCDownloadSet	DscDaplCCDownloadSet*
DscDaplCCListGetA	DscDaplCCListGet*
DscDaplCCListLengthGet	DscDaplCCListLengthGet*
DscDaplCCListSetA	DscDaplCCListSet*
DscDaplCCStackSizeGet	DscDaplCCStackSizeGet*
DscDaplCCStackSizeSet	DscDaplCCStackSizeSet*
DscServerDiskLogEnabledSet (new)	DscServerDiskLogEnabledSet
DscServerDiskLogEnabledGet (new)	DscServerDiskLogEnabledGet
DscServerDiskLogBytes (new)	DscServerDiskLogBytes
DscXbCalibrate (new)	DscXbCalibrate
DscXbEnabledGet (new)	DscXbEnabledGet
DscXbEnabledSet (new)	DscXbEnabledSet
DscXbPinConfigGet (new)	DscXbPinConfigGet
DscXbPinConfigSet (new)	DscXbPinConfigSet
DscGroupConfigDialogShow (new)	DscGroupConfigDialogShow
DscGroupHandleOpen (new)	DscGroupHandleOpen
DscGroupHandleClose (new)	DscGroupHandleClose
DscGroupAddOne (new)	DscGroupAddOne
DscGroupDeleteOne (new)	DscGroupDeleteOne
DscGroupCount (new)	DscGroupCount
DscGroupDsc (new)	DscGroupDsc
DscGroupConfigRead (new)d	DscGroupConfigRead
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	`Call Library Function` **node in** `DSC.LLB`
DscServerDiskLogConfigSet	`MslDscServerDiskLogConfigSet`
DscFilterParametersSet	`MslDscFilterParametesSet`

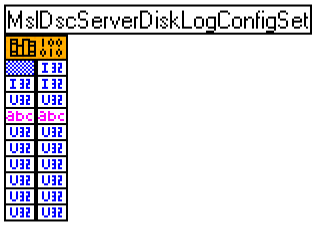
Data Format

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:

C Data Type	**LabVIEW Data Type**
Char*	C String Pointer
Int	Signed 32-bit Integer
Short *	Signed 16-bit Integer
Long	Unsigned 32-bit Integer
Double	Double
HDSC	Signed 32-bit Integer
TBufferGetEx	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.
TDscIoInt64	Array Data Pointer of Signed 32-bit Integer. For example usage, please see Num Data subVI. 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.

MslDscServerDiskLogConfigSet



Input	Name (Type)	Corresponding parameters in `TServerDiskLogConfig` and brief descriptions	Output	Pass-throug h
1:	*hDsc (Integer)*	---	1:	True
2:	*Flag (Long)*	`DwFlags` Specifies various logging options.	2:	True
3	*LogFile (String)*	`pszFileName` Points to a null-terminated string that specifies the name of the disk logfile.	3:	True
4:	*FileShareMode (Long)*	`dwFileShareMode` Specifies the file share properties of the disk logfile.	4:	True
5	*OpenFlags (Long)*	`dwOpenFlags` Specifies how file opening is to be handled.	5:	True
6:	*FileFlagsAttr ibutes (Long)*	`dwFileFlagsAttributes` Specifies additional file attributes.	6:	True
7:	*BlockSize (Long)*	`dwBlockSize` Specifies the minimum amount of data, in bytes, to write to the logfile at one time.	7:	True
8:	*Lowi64 (Long)*	`I64MaxCount` Specifies the low 32-bit maximum number of bytes to	8:	True

		log.		
9:	*Highi64 (Long)*	`I64MaxCount` Specifies the high 32-bit maximum number of bytes to log.	9:	True

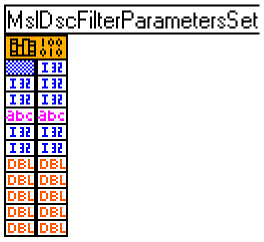
This `Call Library Function` node builds the structure `TServerDiskLogConfig` and passes it to `DscServerDiskLogConfigSet`, which initiates a disk logging session between an iDSC board specified by *hDsc* and a disk file specifies by `LogFile`. The parameter `BlockSize` is provided for disk transfer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

If a full path is not given for the parameter `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 sessions starts when `DscStartAcquiring` is invoked. Once it starts, it continues until the number of bytes specified in `Lowi64` and `Highi64` has been logged or until `DscStopAcquiring` or `DscHandleClose` is invoked on `hDsc`.

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

MslDscFilterParametersSet



Input	Name (Type)	Corresponding parameters in `TFilterParam` and brief descriptions	Output	Pass-through
1:	*hDsc (Integer)*	---	1:	True
2:	*FilterIndex (Integer)*	---	2:	True
3	*Name*	`achName`	3:	True

	(String)	Points to a null-terminated string that specifies the name of the filter.		
4:	*FilterType (Integer)*	`iFilterType` Specifies the type of the filter: `0` for lowpass, `1` for bandpass. The default is lowpass..	4:	True
5	*Sharpness (Integer)*	`iSharpness` Specifies the sharpness of the filter.	5:	True
6:	*CutoffFreqLow (Double)*	`fCutoffFreqLow` Specifies the low cutoff frequency of the filter.	6:	True
7:	*CutoffSlopeLow (Double)*	`fCutoffSlopeLow` Specifies the low cutoff slope of the filter.	7:	True
8:	*CutoffFreqHigh (Double)*	`fCutoffFreqHigh` Specifies the high cutoff frequency of the filter.	8:	True
9:	*CutoffSlopeHigh (Double)*	`fCutoffSlopeHigh` Specifies the high cutoff slope of the filter.	9:	True
10:	*Attenuation (Double)*	`fAttenuation` Specifies the attenuation of the filter.	10:	True

This `Call Library Function` node builds the structure `TFilterParam` and passes it to `DscFilterParametersSet`, which sets the filter parameters associated with a filter at index *`FilterIndex`* on an iDSC board specified by *`hDsc`*. Valid filter indices are 0 through 7.

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

SubVIs Reference

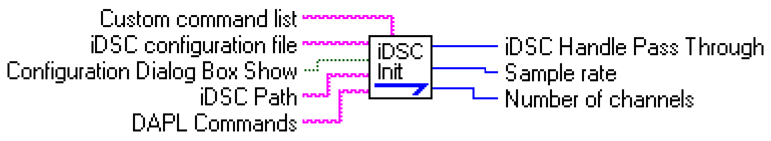
This package provides several subVIs to make it easy to configure and cleanup iDSC board communication. In LabVIEW if you select Show | Help from the Help menu, and place the mouse cursor over the subVI, diagrams as shown below will appear. To place a subVI onto a LabVIEW diagram:

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 `\\.\Dap0`.

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.

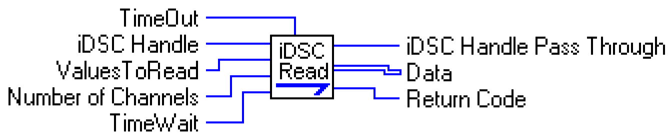
iDSC Init performs the following operations:

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.

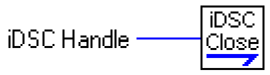
iDSC Read performs the following operations:

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 specifies 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 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`.

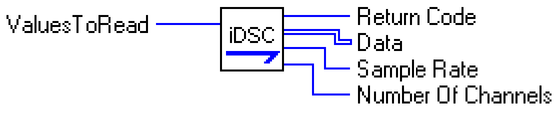
iDSC Close has no output:

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 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:

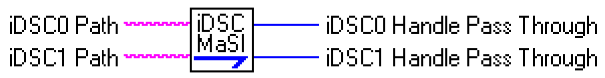
1. `Return Code` reports the result of the get data operation. If the get data operation is succeeds, this variable contains the number of data bytes read. If the get data operation fails, this variable contains -1.
   Integer.
2. `Data` is a pointer to an array, which contains data from the iDSC board.
   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 MaSl

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



iDSC MaSl 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 MaSl 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.

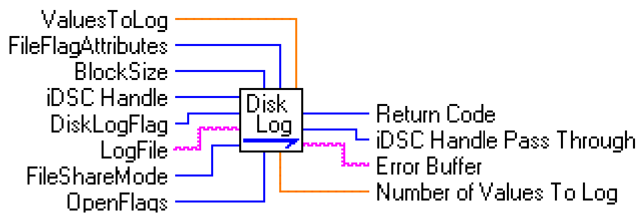
iDSC MaSl performs the following operations:

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. i`DSC 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 sever 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.`

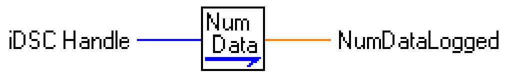
The iDSC 1816 board has to be configured for server disk logging before start acquiring data. So, a Disk Log subVI has to be called before `DscStartAcquiring`.

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.



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 outputs:

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`.

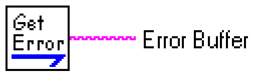
This subVI queries for the number of bytes being logged to a disk file by a server. Since the function `DscServerDiskLogBytes` returns a 64-bit integer, which is not supported by LabVIEW, this subVI is created to interface between DSCIO and LabVIEW. The results are stored in two 32-bit integers. The 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:

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

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.



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.