DAPtools for LabVIEW Manual

Installation Guide &
Function Reference

Version 2.30

Microstar Laboratories, Inc.

Microstar Laboratories, Inc.
2265 116th Avenue N.E.
Bellevue, WA 98004
Tel: (425) 453-2345
Fax: (425) 453-3199
http: // www.mstarlabs.com

Contents

1. Introduction

A Data Acquisition Processor (DAP) can simplify and greatly improve the speed of LabVIEW applications. A few simple DAPL commands configure the data acquisition and processing capabilities of the DAP. The net result is that a DAP in combination with LabVIEW gives the ability to quickly create sophisticated and reliable applications. The DAPtools for LabVIEW contains this reference manual and the DAP32.LLB library, which includes all the necessary library function calls to use a DAP in the LabVIEW environment. Several examples are included showing the use of many of the functions. DAP32.LLB is intended to be an aid to reduce development time for LabVIEW users. The included examples are simple, but show the power of a Data Acquisition Processor when used with the sophisticated GUI capabilities of LabVIEW.

2. Getting Started

DAPtools for LabVIEW requires basic knowledge in several areas:

- LabVIEW basics – If you are a new LabVIEW user, the LabVIEW Tutorial provides a good place to get started. The tutorial is recommended before starting to use a Data Acquisition Processor with LabVIEW.

- The LabVIEW "Call Library Function." – It is important to understand how the Call Library Function maps onto Windows DLL functions and their data types. The DAP32.LLB library file includes Call Library Function definitions to the DAPIO32.DLL functions for a DAP. So this is done already. However, it is good to understand the basics of this capability.

- DAPL (Data Acquisition Programming Language) – See the DAPL Manual and Applications Manual for usage and further examples.

Requirements

Windows 95 or later with Accel32 or DAPcell and LabVIEW 8.0 or later. The DAPtools for LabVIEW has been verified to work with up to LabVIEW 8.6. If you have a newer version and encounter problems, please report to us.

To work with an earlier version of LabVIEW, contact us for a previous version of the DAPtools.

Changes in version 2.30

The examples and files in DAP32.LLB have been updated to work with LabVIEW 8.0 and later. Two examples, App15 and App16, have been added.

Installation

1. LabVIEW must be installed correctly. Please refer to the LabVIEW installation instructions.
2. Install Accel32 or DAPcell Server for the DAP board(s) from the DAPtools CD.
3. Install DAPtools for LabVIEW from the CD. The default destination is C:\Program Files\Microstar Laboratories\DAPtools\LabVIEW. The file `DAP32.LLB` contains the application examples and all the supported functions in "List of DAP Functions."

3. Basic Programming Steps

LabVIEW accesses a DAP in the same way as other programming languages. The following description applies to LabVIEW as well as other languages. A LabVIEW subVI is provided to automatically perform these steps for you (**DAP Init** subVI). However, it is useful to know what it does.

1. Open communication pipes to the ACCEL32 driver which provides communication with the DAP.
2. Send a RESET command to the DAP to stop any present activity.
3. Flush any old data that may be left in the communication pipes.
4. Send DAPL commands to configure the DAP for your specific application. In LabVIEW a string object is useful for storing DAPL commands which is wired to the input of the **DAP Init** subVI. A useful alternative is the DAPConfig function which reads DAPL commands from a file and optionally replaces key parameters.
5. Read data from the Data Acquisition Processor. The App01 - DVM example in DAP32.LLB uses a while loop to continuously read data until the STOP button is pressed.
6. Before quitting, send a DAPL STOP command to stop the DAPL program. Note: The STOP button on the button bar does not stop the DAPL program, meaning THE DAP 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 **DAP Close** subVI which closes the ACCEL32 Handles assigned to the communication pipes. All communication pipes opened should be closed after each run.

Using Several Data Acquisition Processors

The Accel32 or DAPcell Server needs to be installed for all of the DAP boards in the same PC. The boards can be synchronized using a synchronization cable provided by Microstar. Refer to the DAP manual or contact Microstar Laboratories for more information on synchronizing multiple DAP boards in a system.

The **DAP Init** sub VI included in DAPtools for LabVIEW uses the standard DAP0 text and binary communication pipes as outlined in the DAPIO32 documentation. To access additional DAP boards make a copy of the **DAP Init** subVI for each board. Edit the subVI and change the *ACCEL32* Names from 'dap0' to 'dap1', 'dap2', etc. All the handles need to be closed at the end of the program using **DapHandleClose**. Application 14 is an example on using multiple DAP boards in LabVIEW.

Programming suggestions

It is helpful to use the function **DapInputAvail** to determine if there are data values ready to be read. This is helpful to avoid hang situations. A hang can occur if the DAP does not send data. When data stops arriving, a call to read DAP data will wait forever in effect causing the program to hang. Using **DapInputAvail** to verify that data values are available allows a well behaved recovery. After a program is well tested, it is possible to remove the **DapInputAvail** test, but many programs continue to use it to guard against unforeseen problems.

Data transfer can stop when there is an overflow that stops input sampling. Note that all the data that was collected before the overflow is transferred. In cases where the DAP has an overflow condition it generally means that the data rate is too fast for the processing or transfer speed of LabVIEW. Here are some ideas that may help to resolve this condition:

- Reduce the amount of processing in LabVIEW and put more processing on the DAP
- Read and process data in blocks with **DapBufferGet** or **MslDapBufferGetEx** instead of **DapInt16Get**.
- Look for ways to increase the efficiency of the LabVIEW code
- Benchmark the system. In other words test sample rates to find the actual maximum speed of the system.
- Look for bottleneck areas and optimize them if possible
- Look for logical problems that may be causing memory overflows that are not related to speed
- Reduce the sample rate if possible

4. Overview of Example Applications

Several examples are provided in the file DAP32.LLB. These examples illustrate many of the configuration and data transfer methods available.

App01 – DVM

This example shows single channel data acquisition with a digital voltmeter display. The code uses the **DapInt16Get** function to get a single integer sample at a time from the DAP. **DapInt16Get** is put into a For Loop to repeat the process until the user clicks on the STOP button. A more efficient way to get data is to use the **DapBufferGet** or **DapBufferGetEx** function, which is shown in several other examples including the App02 and App15.

App02 – One Graph

This example is essentially the same as the DVM App, but instead uses the **DapBufferGet** function to receive and graph data in blocks.

App03 – DapConfig

Uses the **DapConfig** function to send a DAPL file to the Data Acquisition Processor. Although no parameters were set, the **DapConfigParamSet** command could have been used to set a parameter such as the TIME. This is done App07.

App04 – Disk Log to Binary File

This example shows how data can be saved efficiently to a binary data file using blocks of data. It is important to note that LabVIEW saves binary data with bytes reversed. If another PC application reads the binary data from the file it may need to swap the bytes again.

App05 – Disk Log to Spreadsheet

This example converts binary data from the DAP into floating point spreadsheet format. Due to the formatting, the application has a low transfer rate (< 1ksample/sec on a 486 66Mhz PC). For more efficiency, we recommend using a binary disk logging format.

App06 – Text Messages

This example shows how to read text messages from a DAP. It allows DAPL commands to be sent to the DAP and displays any responses. This example also includes the **DapLastErrorTextGet** function to output the message text of the calling thread's last error, including custom driver-specific errors. This type of error checking can be useful for debugging programs, but reduces the efficiency and throughput of the program.

App07 – Frequency Response

This example uses **DapConfigParamSet** to set the sample rate (DAPL TIME). The application has a control input for the block size, but the user needs to insure that this matches 1/2 of the FFT size set in the Fftapp.dap file. Also see the DAPIO32 Manual for more information on replaceable parameters. The application does not include any error checking or checking for amount of data available on the DAP.

App08 – Multi Display

This example sends merged streams of data to LabVIEW where the values are separated and displayed on several different display controls.

App09 – Transfer Data to DAP

This example shows how to send data to a DAP. In this case the DAP performs an FFT calculation and sends the results back to the PC for display.

App10 – High-Speed Transfer to and from the DAP

This example shows how to read data from and send data to a DAP using **DapBufferGetEx** and **DapBufferPutEx**, respectively. The LabVIEW program sends sinewaves to the DAP and the DAP transfer the data back to the user interface for display.

App11 – High-Throughput Disk Logging

This example shows how to stream data directly to disk using the DAPcell Server disk logging service, **DapPipeDiskLog**. It queries for the DAP model and the amount of data logged to disk as the program continues. The DAPcell Server needs to be installed and the disk logging option needs to be enabled in the Control Panel to run this example.

App12 – Install a Custom Command Module

This example installs a 32-bit custom command module to a DAP. Once a module is installed, it is downloaded every time the Accel32 or DAPcell service starts when the PC boots up. It does not need to be installed or downloaded again unless it is uninstalled.

App13 – Download a Custom Command Module

This example loads a 32-bit custom command module to a DAP. A module needs to be downloaded every time after PC reboots. If it is installed, as in App02, the download is automatic when the PC boots up.

App14 – Two-DAP configuration

This example shows how to use two DAP boards in the same VI. The boards are not synchronized as master and slave, but this example is a good starting point for such a configuration.

App15 – Multi Display – High Speed

This example is similar to App08 where it reads a merged data stream from the DAP board, but it uses **MslDapBufferGetEx** to read a block of data instead of a single value to enhance data throughput. **MslDapBufferGetEx** is recommended for new applications.

App16 – Control Analog and Digital Outputs

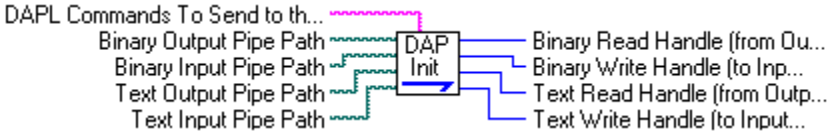
This example shows how to set the analog and digital outputs onboard of a DAP board from LabVIEW.

5. DAP Init and DAP Close SubVIs

DAPtools for LabVIEW provides several subVIs to make it easy to initialize and terminate communication with a DAP.

DAP Init

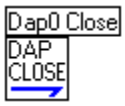
The **DAP Init** subVI delivers an easy way to configure a DAP. In LabVIEW if you select Show help from the Help menu and place the mouse cursor over the **DAP Init** subVI, the following diagram will appear:



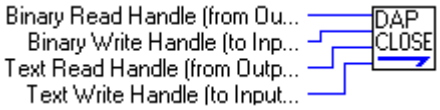
The DAPL Commands input accepts a LabVIEW string that contains DAPL commands for configuring a DAP. The remaining inputs normally are not used. They can be used if you would like to specify different communication pipes to open. If you are using more than one DAP it generally is best to make a copy of **DAP Init** and change the com pipe names from 'dap0' to 'dap1', 'dap2', etc. The output pins of **DAP Init** supply the handles that allow other functions to access the DAP communication pipes.

DAP Init opens handles to the DAP, sends a RESET command, and flushes the input communication pipes. See the diagram for **DAP Init** for additional details of its functionality.

DAP Close



The **DAP Close** subVI takes the handles that were opened by **DAP Init** and closes them. Before closing the handles it sends a STOP command to the DAP through the Text Write Handle. LabVIEW help displays the following diagram. The wires from the **DAP Init** subVI need to match to the inputs of the **DAP Close** subVI.

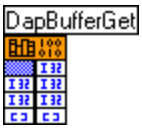


6. Descriptions for DAPIO32 function calls in the DAP32.LLB

The DAP32.LLB includes a VI called ‘List of Call Function’ for all the functions available to LabVIEW. The following descriptions explain the usage for each function. The graphical symbols below show the data types of the input and output pins. For additional information see the DAPIO32 Reference Manual.

DapBufferGet

The **DapBufferGet** function reads a block of data from the target pipe.



		Return:	Bytes read
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*Number of Bytes*	Output 2:	Pass through
Input 3:	*Buffer Pointer*	Output 3:	Buffer output

Parameters

Handle Number

Specifies the open handle to the ACCEL32 device.

Number of Bytes

Specifies the number of data bytes to read

Buffer Pointer

Points to the buffer to receive data.

Return Values

If the function succeeds, the return value is the number of bytes read.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

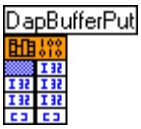
DapBufferGet reads Data Acquisition Processor data into the location pointed to by *Buffer Pointer*. Usually this is initialized as an I16 array. **DapBufferGet** reads data until it has read the given *Number of Bytes*. This function can be used with either text or binary Data Acquisition Processor data. Return Code is the number of bytes read (always identical to *Number of Bytes*).

See Also

MslDapBufferGetEx

DapBufferPut

The **DapBufferPut** function writes a block of data to the DAP.



		Return:	Bytes written
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*Number of Bytes*	Output 2:	Pass through
Input 3:	*Buffer Pointer*	Output 3:	Pass through

Parameters

Handle Number

Specifies the open handle to an ACCEL32 device.

Number of Bytes

Specifies the number of data bytes to write.

Buffer Pointer

Points to the block of data to write.

Return Values

If the function succeeds, the return value is the number of data bytes actually written.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

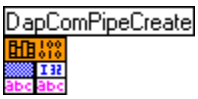
DapBufferPut sends a specified *Number of Bytes* from the buffer pointed to by *Buffer Pointer* to the Data Acquisition Processor. The parameter *Handle Number* contains a handle to an ACCEL32 device (text or binary input communication pipe). This function can be used to send either text or binary data to the Data Acquisition Processor. Return Code is the number of bytes written (always identical to *Number of Bytes*).

See Also

MslDapBufferPutEx

DapComPipeCreate

The **DapComPipeCreate** creates a communication pipe channel between the PC and a DAP. It physically creates pipes both on the PC and on the DAP. This function is destructive on all DAPs except PCI DAPs.



		Return:	True or False
Input 1:	*String Pointer*	Output 1:	Pass through

Parameters

String Pointer

Pointers to a null-terminated string that specifies the name and attributes of the pipe channel to create.

Return Values

If the function succeeds, the return value is TRUE.

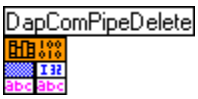
If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

The pipe information string consists of the UNC pipe name, which takes the form of \\<PcName>\<DapName>\<PipeName>. <PcName> is replaced by the computer name of the host PC or "." If the host PC is local. <DapName> is replaced by Dap0, Dap1, …, or DapX, based on the number of DAPs installed on the system and for which DAP the operation is intended. The pipe information string also supports an optional list of attributes. Refer to the DAPIO32 Reference Manual for more information.

DapComPipeDelete

The **DapComPipeDelete** removes the communication channel with the specified UNC name. This function is destructive on all DAPs except PCI DAPs.



		Return:	True or False
Input 1:	*String Pointer*	Output 1:	Pass through

Parameters

String Pointer

Pointers to a null-terminated string that specifies the name of the pipe channel to delete.

Return Values

If the function succeeds, the return value is TRUE,

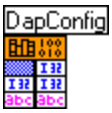
If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function physically removes the paired pipes from the PC and from the DAP. This operation is DESTRUCTIVE. All data about the target pipe in both the PC and the DAP are lost. If the target communication channel does not exist, this function returns success without doing anything.

DapConfig

The **DapConfig** function sends a DAPL command file to a DAP.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*DAPL File Name*	Output 2:	Pass through

Parameters

Handle Number

Handle to an ACCEL32 device.

DAPL File Name

Name of file to send to DAP.

Return Values

If the function succeeds, the return value is TRUE; the complete file was successfully sent to DAP.

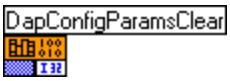
If the function fails, the return value is FALSE; something went wrong sending file. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

Reads the DAPL command file specified by *DAPL File Name* from disk and sends the contents to the Data Acquisition Processor. The parameter *Handle Number* contains a handle to an ACCEL32 device. The specified ACCEL32 device must be opened for text Data Acquisition Processor communication. Also see the DAPIO32 Manual for more information on replaceable parameters.

DapConfigParamsClear

The **DapConfigParamsClear** function clears all program-defined parameters and default parameters.



Input: *none* Return: True or False

Parameters

None.

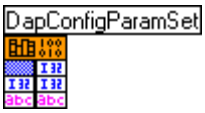
Return Values

If the function succeeds, the return value is TRUE; the parameters were cleared.

If the function fails, the return value is FALSE. Something went wrong clearing parameters. Call **DapLastErrorTextGet** to retrieve additional information about the error.

DapConfigParamSet

The **DapConfigParamSet** function initializes a program-defined parameter.



		Return:	Return code
Input 1:	*Parameter Number*	Output 1:	Pass through
Input 2:	*Parameter String*	Output 2:	Pass through

Parameters

Parameter Number

Parameter number in the range 1 to 100.

Parameter String

String for parameter.

Return Values

If the function succeeds, the return value is TRUE; the parameter was set.

If the function fails, the return value is FALSE; something went wrong setting the parameter. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

DapConfigParamSet sets a parameter string that is used by **DapConfig** to replace corresponding %x parameters in the DAPL command file. The *Parameter Number* can be from 1 to 100. *Parameter String* is the string that replaces the corresponding %x parameter number in the DAPL command file.

DapHandleClose

The **DapHandleClose** function releases an assigned ACCEL32 handle.

		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Specifies the handle to close.

Return Values

If the function succeeds, the return value is TRUE,

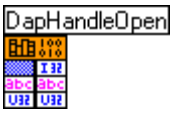
If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

The **DapHandleClose** function closes the assigned ACCEL32 handle number. Handles should be closed when the program ends, or Windows will generate an error. The examples show the use of the **DapHandleClose** function in conjunction with a STOP button.

DapHandleOpen

The **DapHandleOpen** function returns a handle to the target with the specified name.



		Return:	Handle Number
Input 1:	*ACCEL32 Name*	Output 1:	Pass through
Input 2:	*Open Flags*	Output 2:	Pass through

Parameters

ACCEL32 *Name*

Points to a target name string that specifies the target to open.

Open Flags

Specifies the desired access to acquire.

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 zero). Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

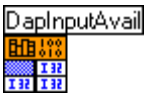
The **DapHandleOpen** function opens a handle number for the given *ACCEL32 Name* string. Handles are assigned as a text string such as described in the DAPIO32 Manual. An example is:

```
\\.\dap0\$Binout
```

Open Flags is a hexadecimal number: 80000000 for reading, 40000000 for writing, 20000000 for querying, and 10000000 for disk I/O.

DapInputAvail

The **DapInputAvail** function gets the number of data bytes available to be read from an input buffer.



		Return:	Bytes Available
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Contains a handle to text or binary output communication pipe previously opened by **DapHandleOpen**.

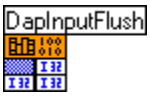
Return Values

If the function succeeds, the return value is the number of data bytes available in the input buffer.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

DapInputFlush

The **DapInputFlush** function flushes data from a DAP com-pipe and returns the number of bytes flushed.



		Return:	Bytes Flushed
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Specifies a handle to the target DAP com-pipe.

Return Values

If the function succeeds, the return value is the number of bytes flushed.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

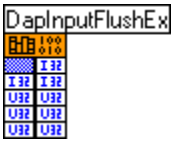
There is a built-in 20 second time-out; if it is unable to flush all data from *Handle Number* after the time-out, it returns failure (-1). If no data are found after a delay of 100 ms, the function returns successfully with the number of bytes flushed so far.

See Also

DapInputFlushEx

DapInputFlushEx

The **DapInputFlushEx** function flushes input data from the target pipe. It always returns even when it is unable to flush all data from the target pipe. **DapInputFlushEx** is an extended version of **DapInputFlush**.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*Time Out*	Output 2:	Pass through
Input 3:	*Time Wait*	Output 3:	Pass through
Input 4:	*Count Pointer*	Output 4	Returned Count

Parameters

Handle Number

Specifies a handle to the target DAP com-pipe.

Time Out

Specifies the maximum amount of time in milliseconds within which the flushing operation should complete.

Time Wait

Specifies the minimum amount of time in milliseconds for which the function should guarantee that the target pipe remains empty to claim success.

Count Pointer

Points to a 32-bit variable that receives the number of data bytes actually flushed. If the caller does not need the number of bytes flushed, set this parameter to NULL.

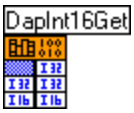
Return Values

If the function succeeds, the return value is TRUE. The variable *Count Pointer* points to the variable containing the total number of data bytes flushed.

If the function fails, the return value is FALSE. The variable *Count Pointer* contains the actual number of bytes flushed. Call **DapLastErrorTextGet** to retrieve additional information about the error.

DapInt16Get

The **DapInt16Get** function reads a single 16-bit integer from a DAP.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*I16 Integer Pointer*	Output 2:	Integer output

Parameters

Handle Number

Handle to DAP com-pipe.

I16 Integer Pointer

Pointer to location to receive integer.

Return Values

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

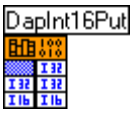
Remarks

Reads a binary integer from the Data Acquisition Processor into the location pointed to by *Integer Pointer*. This function can be used only with binary data. The parameter *Handle Number* contains a handle to a binary output communication pipe. The *Integer Data* parameter must be initialized with a signed 16-bit integer such as zero. **DapInt16Get** is used in the Multi Display example.

Use **DapBufferGet** or **MslDapBufferGetEx** to retrieve a data block instead of one single integer from the DAP.

DapInt16Put

The **DapInt16Put** function writes a single 16-bit integer to a DAP.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*I16 Integer Value*	Output 2:	Pass through

Parameters

Handle Number

Handle to DAP input binary com-pipe.

I16 Integer Value

Integer to write to DAP.

Return Values

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

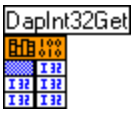
Remarks

This function sends the integer specified by the parameter *Integer Value* to the DAP via the binary input com pipe.

Use **DapBufferPut** or **MslDapBufferPutEx** to send to the DAP a data block instead of one single value.

DapInt32Get

The **DapInt32Get** function reads a single 32-bit integer from a DAP.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*I32 Integer Pointer*	Output 2:	Integer output

Parameters

Handle Number

Handle to DAP com-pipe.

I32 Integer Pointer

Pointer to location to receive integer.

Return Values

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

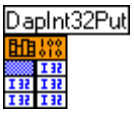
Remarks

This function reads a 32-bit integer from the DAP via the binary output com pipe into the location pointed to by *I32 Integer Pointer*. The *Integer Data* parameter must be initialized with a signed 32-bit integer such as zero.

Use **DapBufferGet** or **MslDapBufferGetEx** to retrieve a data block instead of one single integer from the DAP.

DapInt32Put

The **DapInt32Put** function writes a single 32-bit integer to a DAP com-pipe.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*I32 Integer Value*	Output 2:	Pass through

Parameters

Handle Number

Handle to DAP com-pipe.

I32 Integer Value

Integer to write to DAP.

Return Values

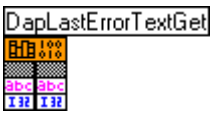
If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Use **DapBufferPut** or **MslDapBufferPutEx** to send to the DAP a data block instead of one single value.

DapLastErrorTextGet

The **DapLastErrorTextGet** function retrieves the message text of the calling thread's last error, including custom DAPIO32-specific errors.



Input 1:	*String Pointer*	Output 1:	Error message string
Input 2:	*String Length*	Output 2:	Pass through

Parameters

String Pointer

Points to an application-supplied buffer that receives the error message.

String Length

Specifies the size of the application supplied buffer.

Return Values

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

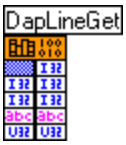
If the function fails, the return value is the pointer to a null string.

Remarks

Sends a text error message to the location defined by *String Pointer*. The message is generated by either the last DAPIO32.DLL function called or a driver-specific message. *String Length* should be initialized with a byte array large enough to hold the largest error message. 100 bytes is usually sufficient. **DapLastErrorTextGet** is used in the Text Messages application.

DapLineGet

The **DapLineGet** function gets a string of characters from the DAP com-pipe with user specified time-out.



		Return:	Number of characters read or -1
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*String Length*	Output 2:	Pass through
Input 3	*String Pointer*	Output 3	Buffer Output
Input 4	*Time Wait*	Output 4	Pass through

Parameters

Handle Number

Handle to DAP text com-pipe.

String Length

Length of *String Pointer* including space for the null character..

String Pointer

Pointer to buffer to store null-terminated string.

Time Wait

Time, in milliseconds, which the com pipe must remain empty to return without a complete string. May be zero.

Return Values

If the function succeeds, it returns the number of characters read from the DAP excluding line-feed characters. Zero indicates that *TimeWait* milliseconds elapsed without reading any characters.

If the function fails, the return value is -1. Either the handle is invalid of the *String Pointer* is NULL. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function reads characters from the com-pipe *Handle number* and places them in *String Pointer* up to the first carriage-return character.

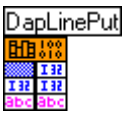
If no characters are available from the com-pipe for more than *Time Wait* milliseconds, returns with whatever characters have been read so far in *String Pointer* and a count of the number of characters read.

See also

DapStringGet

DapLinePut

The **DapLinePut** function writes a string of characters to a DAP com-pipe and terminates the string with a carriage-return.



		Return:	Number of characters written or 0
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*String Pointer*	Output 2:	Pass through

Parameters

Handle Number

Handle to DAP com-pipe.

String Pointer

Null-terminated string to write to DAP. It may be NULL. In this case just a carriage-return is sent to the DAP com-pipe.

Return Values

If the function succeeds, the return value is the number of characters actually written to the DAP com-pipe, including the terminating carriage return.

If the function fails, the return value is 0 (zero). Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

Before appending the carriage-return, **DapLinePut** strips all line-feed and carriage return characters from the right side of the string.

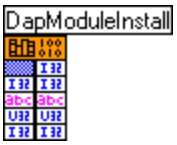
It only checks the end of the string for carriage-return or line-feed characters. To send a string to the DAP com-pipe without modification, use **DapStringPut**.

See also

DapStringPut

DapModuleInstall

The **DapModuleInstall** installs a module to the DAP board(s) associated with the handle.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2	*Module Path*	Output 2	Pass through
Input 3	*Service Flag*	Output 3	Pass through
Input 4	*Reserved*	Output 4	Pass through

Parameters

Handle Number

Identifies the DAP board(s) on which to install the module. If this is a DAP handle, installs the module to the target DAP board. If this is a server handle, installs the module to all DAP board(s) on the server.

Module Path

Path and file name of module to install

Service Flag

Bit Flags telling the service how to proceed with the installation. The following flags are available:

1 – Do not copy the module file

2 – Do not load the module

4 – Force loading

8 – Do not replace the existing module file

10 – Force installation

Reserved

Must be NULL (zero).

Return Values

If the installation is successful, the return value is TRUE. If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

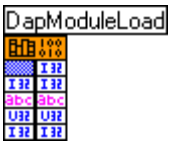
Remarks

This operation is persistent. Once a module is installed, it stays until the module is uninstalled using **DapModuleUninstall**.

The service loads the module to the DAP board(s) unless the flag for no loading (i.e. numeric value of 2) is specified. If installation fails, the loading will not proceed. See the DAPIO32 Reference Manual for details on the bit flags.

DapModuleLoad

The **DapModuleLoad** loads a module to the DAP board(s) associated with the handle.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2	*Module Path*	Output 2	Pass through
Input 3	*Service Flag*	Output 3	Pass through
Input 4	*Reserved*	Output 4	Pass through

Parameters

Handle Number

Identifies the DAP board(s) on which to load the module. If this is a DAP handle, loads the module to the target DAP board. If this is a server handle, loads the module to all DAP board(s) on the server.

Module Path

Path and file name of module to load.

Service Flag

Bit Flags telling the service how to proceed with the installation. The following flags are available:

4 – Force loading

8 – Do not replace the existing module file

Reserved

Must be NULL (zero).

Return Values

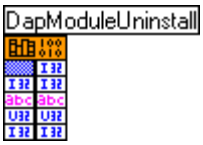
If the loading is successful, the return value is TRUE. If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function loads the specified module without installing it into the system. Therefore, the effect is not persistent. Once the target DAP board(s) are reinitialized (e.g. PC reboot), it is necessary to load the module again if it is not installed using **DapModuleInstall**.

DapModuleUninstall

The **DapModuleUninstall** uninstalls a module from the DAP board(s) associated with the handle.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2	*Module Name*	Output 2	Pass through
Input 3	*Service Flag*	Output 3	Pass through
Input 4	*Reserved*	Output 4	Pass through

Parameters

Handle Number

Identifies the DAP board(s) from which to uninstall the module. If this is a DAP handle, uninstalls the module from the target DAP board only. If this is a server handle, uninstalls the module from all DAP board(s) on the server.

Module Name

Name of module to uninstall

Service Flag

Bit Flags telling the service how to proceed with the uninstall operation. The following flags are available:

2 – Do not unload the module

4 – Force unloading

10 – Force uninstall

20 – Remove dependents of the module

Reserved

Must be NULL (zero).

Return Values

If the uninstall is successful, the return value is TRUE. If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

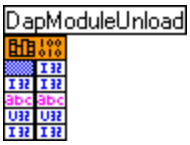
Remarks

This operation is persistent. Once a module is uninstalled, it is removed until the module is re-installed using **DapModuleInstall**.

The service unloads the module from the DAP board(s) unless the flag for no unloading (i.e. numeric value of 2) is specified. If uninstall fails, the unloading will not proceed. See the DAPIO32 Reference Manual for details on the bit flags.

DapModuleUnload

The **DapModuleUnload** unloads a module from the DAP board(s) associated with the handle.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2	*Module Name*	Output 2	Pass through
Input 3	*Service Flag*	Output 3	Pass through
Input 4	*Reserved*	Output 4	Pass through

Parameters

Handle Number

Identifies the DAP board(s) from which to unload the module. If this is a DAP handle, unloads the module from the target DAP board. If this is a server handle, unloads the module from all DAP board(s) on the server.

Module Path

Name of module to unload.

Service Flag

Bit Flags telling the service how to proceed with the unloading. The following flags are available:

4 – Force unloading

20 – Remove dependents of the module

Reserved

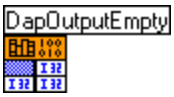
Must be NULL (zero).

Return Values

If the unloading is successful, the return value is TRUE. If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

DapOutputEmpty

The **DapOutputEmpty** function completely empties the target pipe for writing.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Specifies the open handle to an input pipe to the DAP.

Return Values

If the function succeeds, the return value is TRUE. The target pipe is completely empty.

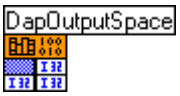
If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

DapOutputEmpty empties an input pipe to the DAP, such as $SYSIN and $BININ. When it returns, the pipe is completely empty on both sides, on the PC and on the DAP.

DapOutputSpace

The **DapOutputSpace** function gets the number of byte spaces available in the target pipe for writing to the DAP.



		Return:	Bytes of space available to write
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Specifies the open handle to an input pipe, such as $SYSIN and $BININ, to the DAP.

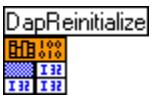
Return Values

If the function succeeds, the return value is the number of byte spaces available in the output buffer.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

DapReinitialize

The **DapReinitialize** function reloads a fresh copy of DAPL to the target DAP board(s) along with all modules installed for the target DAP board(s).



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Handle to DAP board(s) to reinitialize. If this is a DAP handle, reinitialize the target DAP board only. If this is a server handle, reinitialize all the DAP board(s) on the server.

Return Values

Returns TRUE if the DAP board(s) were successfully reinitialized, returns FALSE if an error occurred. Call **DapLastErrorTextGet** to retrieve additional information about the error.

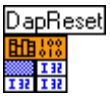
Remarks

This is a destructive operation. All data on the target DAP board(s) will be lost.

This function performs a hardware reset of the specified DAP board(s) and reloads the DAPL operating system along with all installed modules. This is the same as the initialization performed at system boot.

DapReset

The **DapReset** function performs an interlocked DAPL "RESET" on the target DAP board(s).



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through

Parameters

Handle Number

Handle to DAP board(s) to RESET. If this is a DAP handle, perform the DAPL RESET operation on the target DAP board only. If this is a server handle, perform the DAPL RESET operation on all the DAP board(s) on the server.

Return Values

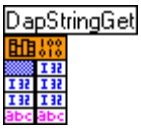
Returns TRUE if the operation is successful, returns FALSE if an error occurred. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

If the operation is successful, the RESET is guaranteed complete before this function returns.

DapStringGet

The **DapStringGet** function gets a string of characters from the DAP.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*String Length*	Output 2:	Pass through
Input 3:	*String Pointer*	Output 3:	String output

Parameters

Handle Number

Handle to DAP com-pipe from which to read a null-terminated string.

String Length

Maximum length of string.

String Pointer

Pointer to buffer to store null-terminated string.

Return Values

If the function succeeds, the return value is TRUE.

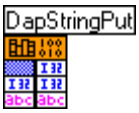
If the function fails, the return value is FALSE; there is something wrong with the handle. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

Reads Data Acquisition Processor text data into the string buffer pointed to by *String Pointer*. The string buffer must be large enough to contain the Data Acquisition Processor data string. **DapStringGet** reads data until it reads a carriage-return from the Data Acquisition Processor. The parameter *String Pointer* must be initialized with an unsigned 8 bit integer array with initial values such as zero.

DapStringPut

The **DapStringPut** function writes a string of characters to a DAP com-pipe.



		Return:	True or False
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*String Pointer*	Output 2:	Pass through

Parameters

Handle Number

Handle to DAP com-pipe.

String Pointer

Null-terminated string to write to DAP.

Return Values

If the function succeeds, the return value is TRUE; the string is written.

If the function fails, the return value is FALSE; there is something is wrong with the handle. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function writes a string of characters to a DAP com-pipe. No terminating character is appended to the string so to send a line of text to the DAPL interpreter the string must include an explicit carriage-return (“\r”).

7. Wrapper functions for DAPIO32 functions in DAP32.LLB

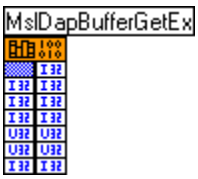
A number of the DAPIO32.DLL functions are initialized by structures and they are not be accessed directly in LabVIEW using the Call Library Function. A wrapper DLL, MSLAPP.DLL, contains interface functions that accept the individual parameters, build the structures necessary, and invoke the DAPIO32 functions directly. To distinguish from the original DAPIO32 functions, these wrapper functions have the prefix Msl followed by the function name for which they interface. Listed below are the wrapper functions and their corresponding functions in the DAPIO32.DLL.

Wrappers	*DAPIO32 functions*
MslDapBufferGetEx	DapBufferGetEx
MslDapBufferPutEx	DapBufferPutEx
MslDapCommandDownload	DapCommandDownload
MslDapHandleQueryInt32	DapHandleQueryInt32
MslDapHandleQueryInt64	DapHandleQueryInt64
MslDapHandleQueryString	DapHandleQueryString
MslDapPipeDiskFeed	DapPipeDiskFeed
MslDapPipeDiskLog	DapPipeDiskLog

The following descriptions explain the usage for each function. The graphical symbols below show the data types of the input and output pins. For additional information see the DAPIO32 Reference Manual.

MslDapBufferGetEx

The **MslDapBufferGetEx** function is a wrapper for the DapBufferGetEx function. It reads a block of data from the target pipe.



		Return:	Number of bytes read or -1
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*Min Bytes*	Output 2:	Pass through
Input 3:	*Max Bytes*	Output 3:	Pass through
Input 4:	*Time Wait*	Output 4:	Pass through
Input 5:	*Time Out*	Output 5:	Pass through
Input 6:	*Bytes Multiple*	Output 6:	Pass through
Input 7:	*Buffer Pointer*	Output 7:	Buffer Output

Parameters

Handle Number
Handle to the target pipe. Requires a handle opened with read access. The target pipe must be an output pipe from the DAP board.

Min Bytes
Minimum number of bytes to get. It can be zero or an positive integer.

Max Bytes
Maximum number of bytes to get. It must be greater than or equal to *Min Bytes*.

Time Wait
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.

Time Out
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.

Bytes Multiple
Specifies that the number of bytes read must be restricted to multiple of this value.

Buffer Pointer
Points to the buffer that receives data.

Return Values

If the function succeeds, the return value is the number of data bytes read.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

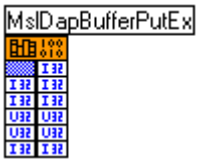
This function builds the structure TDapBufferGetEx with the specified input parameters and passes it to DapBufferGetEx. The DapBufferGetEx function is an extended version of the DapBufferGet. It allows a minimum request count, *Min Bytes*, and a maximum request count, *Max Bytes*, to be specified.

If *Min Bytes* is zero **MslDapBufferGetEx** will never block even when there are no data available. A zero value of *Bytes Multiple* is treated the same as one.

See the DAPIO32 Reference Manual for more details on the structure and the function.

MslDapBufferPutEx

The **MslDapBufferPutEx** function is a wrapper for the DapBufferPutEx function. It reads a block of data from the target pipe.



		Return:	Number of bytes written or -1
Input 1:	*Handle Number*	Output 1:	Pass through
Input 2:	*Bytes Put*	Output 2:	Pass through
Input 3:	*Time Wait*	Output 4:	Pass through
Input 4:	*Time Out*	Output 5:	Pass through
Input 5:	*Bytes Multiple*	Output 6:	Pass through
Input 6:	*Buffer Pointer*	Output 6:	Pass through

Parameters

Handle Number

Handle to the target pipe. Requires a handle opened with write access. The target pipe must be an input pipe from the DAP board.

Bytes Put

Number of bytes of data to put.

Time Wait

Maximum amount of time in milliseconds that the put operation can be blocked waiting for available space. If no space becomes available in that amount of time, the service aborts the operation.

Time Out

Maximum amount of time in milliseconds that the put operation should complete. If it fails to complete in this amount of time, the service aborts the operation.

Bytes Multiple

Specifies that the number of bytes written must be restricted to multiple of this value.

Buffer Pointer

Points to the buffer that contains the data.

Return Values

If the function succeeds, the return value is the number of data bytes written.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

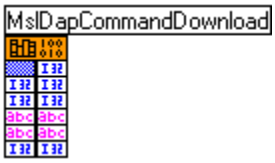
Remarks

This function builds the structure TDapBufferPutEx with the specified input parameters and passes it to DapBufferPutEx. The DapBufferPutEx function is an extended version of the DapBufferPut. It will block waiting for space if the target pipe becomes full before it completes writing all data. In this case, *Time Wait* and *Time Out* determine the behavior of the function. If the put operation fails to complete in *Time Out* milliseconds, or it the pipe remains full for *Time Wait* milliseconds, the functions returns immediately. A zero value of *Bytes Multiple* is treated the same as one.

See the DAPIO32 Reference Manual for more details on the structure and the function.

MslDapCommandDownload

The **MslDapCommandDownload** function is a wrapper for the DapCommandDownload function. It downloads a 16-bit custom command (i.e. .BIN) to DAPL.



		Return:	True or False
Input 1:	*$SYSIN Handle*	Output 1:	Pass through
Input 2:	*$SYSOUT Handle*	Output 2:	Pass through
Input 3:	*CC Filename*	Output 4:	Pass through
Input 4:	*CC Name*	Output 5:	Pass through
Input 5:	*Stack size*	Output 6:	Pass through

Parameters

$SysIn Handle
Handle open to $SYSIN on a DAP board.

$SysOut Handle
Handle open to $SYSOUT on a DAP board or zero.

CC Filename
Path to the compiled binary of the custom command.

CC Name
Name for the custom command or NULL.

Stack size
Stack size for custom command .

Return Values

If the function succeeds, the return value is TRUE; the service downloaded the custom command without error.

If the function fails, the return value is FALSE. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

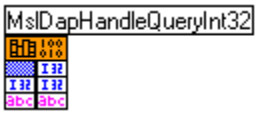
This function builds the structure TDapCommandDownload with the specified input parameters and passes it to DapCommandDownload.

CC Name is the name of the custom command. It may be up to 11 characters and must consist of valid letters for a DAPL command name (A-Z_ followed by A-Z0-9_). It may be NULL. In this case, the first 11 characters, excluding the extension, of *CC Filename* are used for the custom command name.

See the DAPIO32 Reference Manual for more details on the structure and the function.

MslDapHandleQueryInt32

The **MslDapHandleQueryInt32** function is a wrapper for the DapHandleQueryInt32 function. It queries for information about a target that yields 32-bit integer results.



		Return:	32-bit integer result or -1
Input 1:	*Query Handle*	Output 1:	Pass through
Input 2:	*String pointer*	Output 2:	Pass through

Parameters

Query Handle

Identifies the handle to query about.

String pointer

Points to a null-terminated query key string .

Return Values

If the function succeeds, the return value is the 32-bit integer result.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

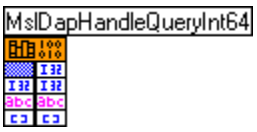
The DapHandleQueryInt32 provides a more convenient interface than DapHandleQuery for queries that yield 32-bit integer results.

There are four types of handles, a NULL handle, a handle to a server PC, a handle to a DAP and a handle to a communication pipe on a DAP. The handle to be queried can be a handle opened with any one of the following attributes, DAPOPEN_READ, DAPOPEN_WRITE, or DAPOPEN_QUERY.

See the DAPIO32 Reference Manual for more details on the query keys that yield 32-bit integer results and the handle that is required for each key.

MslDapHandleQueryInt64

The **MslDapHandleQueryInt64** function is a wrapper for the DapHandleQueryInt64 function. It queries for information about a target that yields 64-bit integer results.



		Return:	True or False
Input 1:	*Query Handle*	Output 1:	Pass through
Input 2:	*String pointer*	Output 2:	Pass through
Input 3:	*Array pointer*	Output 3:	Numeric result

Parameters

Query Handle

Identifies the handle to query about.

String pointer

Points to a null-terminated query key string .

Array pointer

Points to a 2-element array for storing the result.

Return Values

If the function succeeds, the return value is the 64-bit integer result.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function is similar to the **MslDapHandleQueryInt32**. The DapHandleQueryInt64 provides a more convenient interface than DapHandleQuery for queries that yield 64-bit integer results.

Since LabVIEW does not support 64-bit integers, the results are stored in two 32-bit integers. The first element of the Array pointer represents the low 32-bit of the result and the second element the high 32-bit. 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 can be stored in a DOUBLE in LabVIEW.

See the DAPIO32 Reference Manual for more details on the query keys that yield 64-bit integer results and the handle that is required for each key.

MslDapHandleQueryString

The **MslDapHandleQueryString** function is a wrapper for the DapHandleQuery function to provide a more convenient interface for queries that yield string results.

		Return:	True or False
Input 1:	*Query Handle*	Output 1:	Pass through
Input 2:	*Pointer to Query key string*	Output 2:	Pass through
Input 3:	*Pointer to result string*	Output 3:	Numeric result

Parameters

Query Handle

Identifies the handle to query about.

Pointer to Query key string

Points to a null-terminated query key string .

Pointer to result string

Points to a string for storing the result.

Return Values

If the function succeeds, the return value is TRUE.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function applies to all queries that yield strings as results. See DapHandleQuery in the DAPIO32 Reference Manual for more information.

MslDapPipeDiskFeed

The **MslDapPipeDiskFeed** function is a wrapper for the DapPipeDiskFeed function.

Note: This service is only available with DAPcell Server and DAPcell Local Server version 4.00 or later. It is not available with Accel32. The disk logging option needs to be enabled in the Control Panel before this function can be used.

		Return:	True or False
Input 1:	*Feed Handle*	Output 1:	Pass through
Input 2:	*Bytes Put*	Output 2	Pass through
Input 3:	*Time Wait*	Output 3	Pass through
Input 4	*Time Out*	Output 4	Pass through
Input 5:	*Bytes Multiple*	Output 5:	Pass through
Input 6:	*Filename Pointer*	Output 6:	Pass through
Input 7:	*File Share Mode*	Output 7:	Pass through
Input 8:	*Disk Reading Flag*	Output 8:	Pass through
Input 9:	*File Attributes*	Output 9:	Pass through
Input 10:	*Low Part of Max Count*	Output 10	Pass through
Input 11:	*High Part of Max Count*	Output 11	Pass through
Input 12:	*Block Size*	Output 12:	Pass through

Parameters

Disk I/O Handle
Handle previously obtained using **DapHandleOpen** with the DAPOPEN_DISKIO attribute.

Bytes Put
Specifies the number of bytes of data to put.

Time Wait
Specifies the longest time in milliseconds that the put operation can be blocked waiting for available space.

Time Out
Specifies the longest time in milliseconds that the put operation should complete.

Bytes Multiple
Specifies that the number of bytes written must be restricted to multiple of this value.

Filename Pointer
Points to a null-terminated string that specifies the name of the data file to read.

File Share Mode
Specifies the file share properties of the disk data file.

Disk Reading Flag
Specifies various disk-reading options

File Attributes
Specifies additional file attributes.

Low Part of Max Count
Lower 32-bit of the maximum number of bytes to feed. The default value is zero.

High Part of Max Count
Higher 32-bit of the maximum number of bytes to feed.

Block Size
Specifies the minimum amount of data, in bytes, to read from the data file at one time when enough data are available. This field is provided for disk transfer optimization. The default value is 8192.

Return Values

If the function succeeds, the return value is True.

If the function fails, the return value is –1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function builds the structures TdapBufferPuttEx and TdapPipeDiskFeed and pass them to DapPipeDiskFeed to initiate a disk logging session between a DAP and a disk file. The TdapBufferPutEx is the same as the one used in **DapBufferPutEx**.

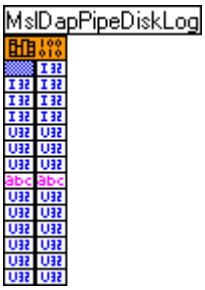
The *I64MaxCount* in TdapPipeDiskFeed is a 64-bit integer. Since LabVIEW does not support 64-bit integers, the maximum count needs to be split into its low and high 32-bit. The maximum count can be represent using DOUBLE in LabVIEW. The low and high 32-bit can be calculated using the Quotient & Remainder function. The low 32-bit is the remainder when the maximum count is divided by 2^{32}; the high 32-bit is the quotient. The same computation is used for calculating the maximum number of bytes to log to disk for **MslDapPipeDiskLog**. Application 11 shows how the computation is done.

Once the disk feed session is initiated, it continues until the number of bytes specified has been read, or the end of the file is reached, or until the handled pass to the command is closed using **DapHandleClose**. See the DAPIO32 Manual for detail information on the flags and attributes.

MslDapPipeDiskLog

The **MslDapPipeDiskLog** function is a wrapper for the DapPipeDiskLog function.

Note: This service is only available with DAPcell Server and DAPcell Local Server version 4.00 or later. It is not available with Accel32. The disk logging option needs to be enabled in the Control Panel before this function can be used.



		Return:	True or False
Input 1:	*Disk I/O Handle*	Output 1:	Pass through
Input 2:	*Min Bytes*	Output 2	Pass through
Input 3	*Max Bytes*	Output 3	Pass through
Input 4:	*Time Wait*	Output 4	Pass through
Input 5	*Time Out*	Output 5	Pass through
Input 6:	*Bytes Multiple*	Output 6	Pass through
Input 7:	*Disk Logging Flag*	Output 7:	Pass through
Input 8:	*Filename Pointer*	Output 8:	Pass through
Input 9:	*File Share Mode*	Output 9:	Pass through
Input 10:	*File Open Mode*	Output 10:	Pass through
Input 11:	*File Attributes*	Output 11:	Pass through
Input 12:	*Max Count*	Output 12	Pass through
Input 13:	*Block Size*	Output 13:	Pass through

Parameters

Disk I/O Handle
Handle previously obtained using **DapHandleOpen** with the DAPOPEN_DISKIO attribute.

Min Bytes
Specifies the minimum number of bytes to get. It can be zero or any positive integer.

Max Bytes
Specifies the maximum number of bytes to get. It must be greater than or equal to *Min Bytes*.

Time Wait
Specifies the longest time in milliseconds that the get operation can be blocked waiting for data.

Time Out
Specifies the longest time in milliseconds that the get operation should complete.

Bytes Multiple
Specifies that the number of bytes read must be restricted to multiple of this value.

Disk Logging Flag
Specifies various logging options.

Filename Pointer
Points to a null-terminated string that specifies the name of the primary disk logfile and the name of a possible mirror disk logfile.

File Share Mode
Specifies the file share properties of the disk logfile.

File Open Mode
Specifies how file opening is to be handled.

File Attributes
Specifies additional file attributes.

Max Count
Specifies the maximum number of bytes to log. The default value is zero.

Block Size
Specifies the minimum amount of data, in bytes, to write to the logfile at one time. This field is provided for disk transfer optimization. The default value is 8192.

Return Values

If the function succeeds, the return value is TRUE. The logging session was started successfully.

If the function fails, the return value is -1. Call **DapLastErrorTextGet** to retrieve additional information about the error.

Remarks

This function builds the structures TDapBufferGetEx and TDapPipeDiskLog and pass them to DapPipeDiskLog to initiate a disk logging session between a DAP and a disk file. The TDapBufferGetEx is the same as the one used in **DapBufferGetEx**.

Once the logging session starts, it continues until the number of bytes specified in *Max Count* has been logged or until the handled pass to the command is closed using **DapHandleClose**. See the DAPIO32 Manual for detail information on the flags and attributes.

Index