

DAPtools for Matlab Manual

DAPIO32 Interface for Matlab

Version 5.00

Microstar Laboratories, Inc.

This manual contains proprietary information which is protected by copyright. All rights are reserved. No part of this manual may be photocopied, reproduced, or translated to another language without prior written consent of Microstar Laboratories, Inc.

Copyright © 2000 – 2009

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

Microstar Laboratories, DAPcell, Accel, Accel32, DAPL, DAPL 2000, DAP Measurement Studio, DAPstudio, DAPcal, DAPlog, DAPview, Data Acquisition Processor, DAP, DAP840, DAP4000a, DAP4200a, DAP4400a, DAP5000a, DAP5016a, DAP5200a, DAP5216a, DAP5380a, DAP5400a, and Channel List Clocking are trademarks of Microstar Laboratories, Inc.

Microstar Laboratories requires express written approval from its President if any Microstar Laboratories products are to be used in or with systems, devices, or applications in which failure can be expected to endanger human life.

Microsoft, MS, and MS-DOS are registered trademarks of Microsoft Corporation. Windows is a trademark of Microsoft Corporation. IBM is a registered trademark of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation. Novell and NetWare are registered trademarks of Novell, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Part Number DAPMATLABM500

Contents

1. Introduction.....	4
2. Getting Started.....	5
Requirements	5
Installation.....	5
New Features.....	5
3. MEX Functions.....	6
Some General Notes	7
dapccldd	8
dapclose	9
dapclrpa	10
dapcnfig.....	11
dapcpc.....	12
dapcpd	14
dapfedpd.....	15
dapflshi	17
dapflsho	18
dapgavl	19
dapgetm	20
dapgstr	22
dapmdin.....	24
dapmdd	25
dapmduin.....	26
dapmduld.....	27
daplogpd.....	28
dapopen	31
dappavl	33
dappstr	34
dapputm.....	35
dapquery	37
daprecnf.....	39
dapreset.....	40
daprint.....	41
dapsetpa	42
4. Examples.....	43
Set 1	43
File INIT.M	43
File DATA.M	43
File STOP.M	44
DAPL Command File: DATA.DAP	44
Running the Examples	44
Set 2	45
Reading Binary File.....	46
5. Appendix A – Obsolete Functions.....	47
dapflush	48
dapgbuf	49
dappbuf	51

1. Introduction

DAPtools for MATLAB™ allows a user to communicate with a Data Acquisition Processor™ (DAP) using the MATLAB programming environment. It integrates the processing power of a Data Acquisition Processor with the powerful technical computing environment of MATLAB.

MATLAB performs high-level numeric computation and visualization of data sets. DAPtools for MATLAB allows acquired and processed data from a Data Acquisition Processor to be further manipulated under MATLAB to suit the specific requirements of an application.

MATLAB is a registered trademark of *The MathWorks, Inc.*

2. Getting Started

DAPtools for Matlab requires basic knowledge in Matlab and DAPL (Data Acquisition Programming Language) for the DAP. See the DAPL Manual and Application Manual for usage and examples.

Requirements

Windows 95 or later with Accel32 or DAPcell and Matlab R2007a or later. The DAPtools for Matlab has been verified to work with up to Matlab R2009b. If you have a newer version and encounter problems, please report to us.

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

Installation

Follow the steps below to install the DAPtools for Matlab.

1. Insert a DAPtools Standard or Professional CD. If the CD does not start automatically, go to Start | Run... and type [CD drive letter]:\Setup.exe. For example, if the CD drive is E, type "E:\Setup.exe". Click on OK.
2. Select DAPtools for Matlab and follow the instructions onscreen to complete the installation.
3. Go to the installed directory. The default destination is C:\Program Files\Microstar Laboratories\DAPtools\Matlab. Unzip the file MEXW32.zip to the root of the installed directory.

In order to use the DAPtools, Matlab and a DAP board should be installed with Accel32 or DAPcell Server. Refer to the DAP Manual for instructions on installing a DAP.

You must add the installed directory of DAPtools for Matlab to the MATLAB search path. The MATLAB's search path is established in the MATLABRC.M file, which resides in the directory MATLAB is installed. The best way to do this is to add an addpath command to STARTUP.M file. If STARTUP.M does not exist, you can create one with a text editor. You should add one line to the installed directory. Below is a description of adding the search path, the DAPtools for MATLAB installed directory, to the MATLAB search path.

1. Create STARTUP.M, if not already exists, in \TOOLBOX\LOCAL\ under MATLAB installed directory.
2. Add the following lines to it.

```
disp('Adding DAPtools for MATLAB to MATLAB search path');  
p='C:\ProgramFiles\MicrostarLaboratories\DAPtools\Matlab'  
addpath(p, path);
```

3. Save the file. MATLAB will run STARTUP.M every time it starts up.

New Features

Version 5.00 of the DAPtools for MATLAB has been recompiled with a 32-bit compiler. The new MEX files are of extension MEXW32. Despite of the warning messages from MATLAB, the previous MEX functions of extension DLL are still compatible with up to MATLAB R2009b.

The function dapflush is no longer supported. It is replaced by dapflshi.

3. MEX Functions

The DAPIO32 DLL contains functions that make it easy to create a PC application for the DAP board. MATLAB cannot directly call the DAPIO32 DLL. However, MATLAB can call and execute MEX functions, which provide access to the functions in DAPIO32 DLL.

DAPtools for MATLAB contains a collection of MEX functions, which provides an interface between DAPIO32 DLL and MATLAB. For this reason, the DAPIO32 DLL must be installed. It is copied to the Windows directory when one of the following drivers is installed with a DAP board: Accel32, DAPcell Local, Client, or Server.

Not all functions provided in the DAPIO32 DLL are accessible through MEX functions because of the nature of the application environment. The name of each MEX function differs from the function name in DAPIO32 DLL. Also, the usage of some of the functions may vary slightly to be compatible with the MATLAB environment. The following table shows the comparison between the MEX functions and DAPIO32 DLL routines. You can obtain help on any function by typing `help function`, e.g. `help dapclose`, at the MATLAB command window.

MEX Functions for MATLAB

dapccld
dapclose
dapclrpa
dapcnfig
dapcpc
dapcpd
dapfedpd
dapflshi
dapflsho
dapgavl
dapgetm
dapgstr
dapmdin
dapmld
dapmduin
dapmduld
daplogpd
dapopen
dappavl
dappstr
dapputm
dapquery
daprecnf
dapreset
daprinit
dapsetpa

DAPIO32 DLL Routines

DapCommandDownload
DapHandleClose
DapConfigParamsClear
DapConfig
DapComPipeCreate
DapComPipeDelete
DapPipeDiskFeed
DapInputFlushEx
DapOutputEmpty
DapInputAvail
DapBufferGetEx
DapLineGet
DapModuleInstall
DapModuleLoad
DapModuleUninstall
DapModuleUnload
DapPipeDiskLog
DapHandleOpen
DapOutputSpace
DapLinePut
DapBufferPutEx
DapHandleQuery
DapConfigRedirect
DapReset
DapReinitialize
DapConfigParamSet

Some General Notes

Return values report whether the functions have executed successfully. Most of the returned values are messages (string) or error codes (number). The function descriptions and the DAPIO32 Reference Manual explain these codes.

For the parameter names, you may choose any name your prefer. It is important to use all of the string input arguments with single quotes.

Some of the MEX functions take filename as input parameter. If filename does not exist in the current directory, the MEX function will look for filename in MATLAB search path.

Some of the MEX functions have optional return values. All names for the returned arguments like `handle`, `code`, `numBytes`, `dataM`, and `string` are chosen by the user. Some of the return arguments are optional. If a name is not given for these, the optional returned values are inaccessible.

Some functions have optional *TimeWait* and *TimeOut* parameters, in unit of milliseconds. The parameter *TimeWait* specifies the longest time in milliseconds that the operation can be blocked waiting for data. If no data show up in that amount of time, the service aborts the operation. The parameter *TimeOut* specifies the longest time in milliseconds that the operation should complete. If it fails to complete in this amount of time, the service aborts the operation. When *TimeOut* is non-zero, it takes precedence over *TimeWait*. That means the service aborts the operation when *TimeOut* expires even though *TimeWait* is not expired yet. The service ignores *TimeOut* if its value is zero. The default for *TimeWait* and *TimeOut* are 100 and 20,000 milliseconds respectively. Please see the corresponding function in the DAPIO32 Reference Manual for more information.

Some parameters are lists of bitmap options. Bitmap options are represented by a string containing zero or more reserved option names. List the names between single quote characters (' '). Use a blank space to separate option names. The option names are case sensitive and must have correct spelling.

dapccldd

Download a custom command to a DAP board.

```
<boolean> = dapccldd(hDapTextIn, hDapTextOut, 'filename')
```

Parameters

hDapTextIn

Handle to a DAP com pipe.

Double.

Must be a handle to a DAP text input com pipe opened with write access.

hDapTextOut

Handle to a DAP com pipe.

Double.

Must be a handle to a DAP text output com pipe opened with read access.

'filename'

UNC path and name of the binary file of the custom command to be downloaded.

String.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function loads 16-bit custom command binaries to a DAP board. The command binary must be developed by using the Developer's Toolkit for DAPL and must be compiled by the operating system that runs on the DAP board (DAPL 4.x or DAPL 2000). Do not use this command to load 32-bit command modules to a DAP board. Install 32-bit command modules using the Data Acquisition Processor control panel application. For advanced application use `dapmdin` and related services.

Example

```
dapccldd(hSysin, hSysout, 'c:\DEXPAND.bin');
```

Download the custom command DEXPAND by sending its binary file DEXPAND.bin to the com pipe specified by `hSysin`.

See Also

`dapmdin`

dapclose

Closes a target handle to a DAP, communication pipe or server.

<boolean> = **dapclose**(*handle*)

Parameters

handle

Handle to a DAP, a com pipe or a server.

Double.

Must be previously returned by **dapopen**.

Return Values

boolean

Indicate whether *handle* is being close successfully.

Double.

Optional. TRUE if successful, FALSE if function failed.

Description

This function closes the handle specified by *handle*, which has previously opened with **dapopen**. The return argument *boolean* is optional. If no return argument is given, error checking is handled internally, and an error message will be prompted.

Example

```
dapclose (hDapSysOut)
```

Closes the handle specifies by hDapSysOut without retaining the return value.

```
ret = dapclose (hDapBinOut)
```

Closes the handle specifies by hDapBinOut, and retains the return value in the return argument ret.

See Also

dapopen

dapclrpa

Clears all program-defined parameters and default parameters.

`<boolean> = dapclrpa`

Parameters

None

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional

Description

This function clears all program-defined parameters and default replaceable parameters in memory. While this function returns a result, it currently never returns failure. Future versions may return failure, so it is worth checking the result.

Example

```
dapclrpa
```

Clear all program-defined and default parameters in memory without storing the returned function termination value.

See Also

dapclrpa

dapcnfig

Sends a DAPL file through a DAP communication pipe, with parameter substitution.

```
<boolean> = dapcnfig(handle, 'filename')
```

Parameters

handle

Handle to a DAP com pipe.

Double.

Must be a DAP com pipe opened with write access.

'filename'

Path and filename of the DAPL file being send to the DAP com pipe.

String.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function sends a DAPL command file to a the DAPL interpreter through the com pipe specified by handle. For more information, please see DapConfig in the DAPIO32 Reference Manual.

Example

```
dapcnfig(hDapSysin, 'c:\test.dap');
```

Sends the DAPL file test.dap in C: drive to the DAP com pipe specified by hDapSysin.

See Also

daprecnf, dapsetpa, dapclrpa

dapcpc

Physically creates a communication pipe channel between the PC and a DAP.

```
<boolean> = dapcpc('plInfo <[type=dType maxsize=v1 | maxsize=v2 blocking=bsz]>')
```

Parameters

plInfo

UNC path with pipe name to create.
String.

type = dType

PC side attributes that describe the data type of the elements transferred through the pipe.
Integer.
Optional. The value of *dType* must be *byte*, *word*, *long*, *float*, *double* or *text*.

maxsize = v1

PC side attributes that specify the maximum pipe buffer size in unit of elements.
Integer.

Optional. The value of *v1* should be at least 1024. When it is absent, the default value is 4096 (elements).

maxsize = v2

DAP side attributes that specify the maximum pipe buffer size in unit of elements.
Integer.

Optional. The value of *v2* should be at least 1024. When it is absent, it defaults to the *maxsize* on the PC side.

blocking = bsz

DAP side attributes that specify the number of elements available in the pipe buffer before data are transmitted to the PC side pipe buffer.

Integer.

Optional. Default is one.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function instructs DAPIO32 to create a communication pipe channel between the PC and a DAP. This function has only one parameter. It is a string of both UNC pipe name and optional attribute list. The UNC pipe name and optional attribute list are separated by a space. The optional attribute list contains two parts, PC side and DAP side, separated by a vertical bar. They can be absent if default values are assumed. For more information, please see `DapComPipeCreate` in the DAPIO32 Reference Manual.

Example

```
dapcpc('\\.\dap0\Cp4In [type = word maxsize=2048]')
```

Create a com pipe Cp4In with default DAP side attributes.

```
ret = dapcpc('\\.\dap0\Cp4In [type=long maxsize=2048 | maxsize = 4096]')
```

Create a com pipe Cp4In with new DAP side attributes.

See Also
dapcpd

dapcpd

Removes a communication connection from the PC and a DAP.

```
<boolean> = dapcpd('pInfo')
```

Parameters

'pInfo'

UNC path and name of the pipe to be deleted.
String.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function instructs DAPIO32 to delete a communication pipe channel between the PC and a Data Acquisition Processor. The pipe and name to be deleted are specified in a UNC notation in parameter *'pInfo'*. If the target communication pipe channel does not exist, this function returns success without doing anything. If any attribute information is present in the *pInfo* string, it will be ignored. For more information, please see `DapComPipeDelete` in the DAPIO32 Reference Manual.

Example

```
dapcpd('\\.\dap0\Cp4In [type = word, maxsize=2048]')
```

Deletes a com pipe Cp4In to Dap0 ignoring the pipe attributes list.

```
ret = dapcpd('\\.\dap0\Cp4In')
```

Deletes a com pipe named Cp4In to local Dap0, and stores the function termination status in the return argument `ret`.

See Also

dapcpc

dapfedpd

Initiates a background disk feed session that reads data from an existing data file and places it into a DAP com pipe.

```
<boolean> = dapfedpd(handle, 'flags', 'filename', 'fileShareMode', 'fileFlagAttr', maxCount, blockSize  
<,[bytesMult, timewait, timeout]> )
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be an input pipe to a DAP opened with diskio access.

'flags'

A list of bitmap options specifying the behavior of disk feed operation.
String.
Must be *'ServerSide'*, *'FlushBefore'*, *'ContinuousFeed'*, or *'BlockTransfer'*. See *bmFlags* under TDapPipeDiskFeed in DAPIO32 Reference Manual.

'filename'

UNC path name of the data file being sent to the DAP com pipe.
String.

'fileShareMode'

A list of bitmap options specifies the file share properties of the data file.
String.
Must be *'read'*, or *'write'*. See *dwFileShareMode* under TDapPipeDiskFeed in DAPIO32 Reference Manual.

'fileFlagAttr'

A list of bitmap option specifying additional file attributes.
String.
Must be *'normal'*, *'readonly'*, *'encrypted'*, or *'sequentialscan'*. See *dwFileFlagsAttributes* under TDapPipeDiskFeed in DAPIO32 Reference Manual.

maxCount

Maximum number of bytes to feed.
Double.
It must be a multiple of *bytesMult* value. Default is zero.

blocksize

Minimum amount of data (in bytes) to read from the data file when enough data are available.
Double.
Use this field to optimize disk transfer. The default value is 8192.

bytesMult

Number of bytes written must be restricted to multiples of this value.
Double.
Optional, can be zero. Default is one.

timewait

Longest time the feed operation can wait for available space.
Double.
Optional. Default is 1000 millisecond.

timeout

Longest time for the feed operation to complete.
Double.
Optional. Default is zero, which means no time out.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function initiates a disk feed session. During a disk feed session, data will be read from '*filename*' and transferred in blocks to the com pipe specified by *handle*. The parameter '*flags*' controls the behavior of disk-feed operation. The parameter '*fileShareMode*' defines how '*filename*' is read, and the parameter '*fileFlagAttr*' sets attributes of the '*filename*'. The parameter *blockSize* specifies how many bytes in a block of transfer. Since the speed of data transfer is machine dependent, increasing *blockSize* may improve the throughput of this operation.

Disk feeding operation occurs in the background. When this function returns, the feeding process is started, but this does not mean that the feeding is finished. The MATLAB application can continue doing other processing.

This function does not use a file handle. Instead, a `dapopen` function establishes a connection to a com pipe, then later this function completes the association with a disk file. Once the association with the file is created, the way to disassociate from the file is to close the handle passed to the command by using `dapclose`. Calling `dapquery` function for the current status can check feeding operation has finished. The optional parameters *bytesMult*, *timewait*, *timeout* are used as a group. If one of the optional parameter is used, all three must present. The default values are specified in the parameters section.

For more information, please see `DapPipeDiskFeed`, `TDapPipeDiskFeed` and `TDapBufferPutEx` in the DAPIO32 Reference Manual.

Example

```
dapfedpd(hDap, 'ContinuousFeed', 'c:\inData.dat', ", ", 0, 16384)
```

Feed the data from the file `inData.dat` in `c:` drive to the DAP specified by the handle `hDap`. The file is repeatedly fed to the DAP board to provide a continuous stream of data, transferred in blocks of 16384 bytes.

```
ret = dapfedpd(hDap, 'ServerSide', 'c:\inData.dat', ", ", 60000, 0, 10, 10, 90)
```

Feed 60000 bytes of data from the file `inData.dat` in `c:` drive to the DAP specified by the handle `hDap`. The data file to be read resides on the same side of the network connection as the DAP board. The data will be transfers in blocks of 8192 bytes as default. The number of bytes to feed must be a multiple of 10. This operation will wait at most 10 milliseconds for available space in the buffer. If the operation does not return in 90 milliseconds, the service aborts the operation. The function termination status is stored in the return argument `ret`.

See Also

`dapquery`

dapflshi

This function flushes a DAP output communication pipe for reading.

```
<boolean> = dapflshi(handle <, timeout, timewait>)
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be an output pipe from a DAP opened with read access.

timeout

Longest time for the flush operation to complete.
Double.
Optional. Default is 100 milliseconds.

timewait

A time interval that the pipe must remain empty after removing all data.
Double.
Optional. Default is 20,000 milliseconds.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function flushes all data in the target pipe specified by *handle*. The target pipe has to be an output pipe from a DAP and an input pipe to an application. It always returns, even when it is unable to flush all data from the target pipe. The *timewait* and *timeout* parameters control timing behavior. Because DAPL processing can produce streams of data independent of the PC application, it is common to have a backlog of data available to send to the PC. Commanding the DAP to stop producing data can guarantee that no new data are produced, but this cannot guarantee the com pipe is empty. Specifying *timewait* allows time for any transfers to complete. If this operation cannot complete before *timeout* interval, presumably the data generation process was not stopped successfully, so this function fails. The operation termination status will be returned in the optional return argument *boolean*.

Example

```
dapflshi(hDapSysout);
```

Flush the data in the com pipe specified by the handle hDapSysout, which is an output pipe from the DAP and opened with read access. The default values of *timewait* and *timeout* are used. The function termination status is not stored.

```
ret = dapflshi(hDapBinout, 10000, 1000);
```

Flush the data in the com pipe specified by the handle hDapBinout, which is an output pipe from the DAP and opened with read access. After removing all data, the pipe must remain empty for 1000 milliseconds. This must occur within 10,000 milliseconds. The function termination status is stored in the return argument ret.

See Also

dapflsho

MEX Functions

dapflsho

This function completely empties a DAP input communication pipe for writing.

`<boolean> = dapflsho(handle)`

Parameters

handle

Handle to a DAP com pipe.

Double.

Must be an input pipe to a PCI DAP opened with write access.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function completely empties com pipe specified by *handle* for writing. The com pipe has to be an input pipe to a DAP, and an output pipe from an application. This function is currently only supported by PCI DAP boards. When it returns, the pipe is completely empty on both sides, on the PC and on the DAP.

Unlike `dapflshi` the PC application has control of data generation, so this function does not need timing control parameters.

Example

```
dapflsho(hDapSysin);
```

Completely empties the input com pipe specified by the handle `hDapSysin`, and does not store the function termination status.

```
ret = dapflsho(hDapBinin);
```

Completely empties the input com pipe specified by the handle `hDapSysin`, and stores the function termination status in the return argument `ret`.

See Also

`dapflshi`

dapgavl

Get the number of bytes available for reading in the target pipe.

```
numBytes = dapgavl(handle)
```

Parameters

handle

Handle to a DAP com pipe.

Double.

Must be an output pipe from a DAP.

Return Values

numBytes

Number of bytes available in the target pipe for reading if successful; -1 if function failed.

Double.

Description

This function gets the number of bytes available in a com pipe specified by *handle*, for reading. It always returns a count in bytes regardless of the actual width of pipe elements. The returned number *numBytes* is the number of data bytes already buffered by the server. An application is always safe to read that number of data bytes from the pipe without being blocked.

The return value should be treated as a “data available/data not available” logic value. The numerical value does not reflect the total amount of data that might be present in PC memory. Unless the application has other processing that must not wait, it is better to use `dapgetm` with a *timeout* specification.

Example

```
ret = dapgavl(hDapBinout)
```

Get the number of bytes available for reading in the pipe specified by handle `hDapBinout`. The returned number of data bytes available is stored in the return value argument `ret`.

See Also

`dappavl`

dapgetm

Get a matrix of data from a communication pipe.

```
[dataM <, numData>] = dapgetm(handle, [mrows, ncols], 'dataType'  
                             <, timeWait, timeOut> )
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be opened with read access.

[mrows, ncols]

Dimension of the matrix to take from the com pipe.
A mrows-by-ncols matrix.

'dataType'

The type of data in the returned matrix dataM.
String.
Must be *'int16'*, *'int32'*, *'double'* or *'float'*.

timewait

Longest time the application can be blocked waiting for data.
Double.
Optional. Default is 100 milliseconds. If one of the optional parameters is used, both must present.

timeout

Longest time for the get operation to complete.
Double.
Optional. Default is 100 milliseconds. If one of the optional parameters is used, both must present.

Return Values

dataM

Data matrix taken from the com pipe, zero-matrix if function fails.

numData

Number of data items taken from the target pipe if successful; -1 if function failed.
Double.
Optional.

Description

When this function executes, a matrix of data will be read from the target DAP com pipe specified by *handle*. The dimension of the data matrix is specified by the parameters *mrows* and *ncols*. The matrix is filled by column, so each row corresponds to a new sampling cycle, and each column corresponds to a data source. The type of data being read is specified by the parameter *'dataType'*.

The *'dataType'* parameter must be one of the following values.

'int16' specifies 16-bit integer

'int32' specifies 32-bit integer

'double' specifies double precision floating point

'float' specifies single precision floating point

This function also allows an application to specify *timewait* and *timeout* parameters to control the behavior of the function. These two parameters are used in pairs. If one is used, the other must be present. This function will return immediately with the data read so far if a time-out occurs, and the remained space will be filled by zero.

The matrix of data is stored in the return argument *dataM*, and the number of data read is stored in the optional return argument *numData*.

Example

```
[dataM, ret] = dapgetm(hDapBinout, [6, 3], 'int16');
```

Get data from the com pipe specified by *hDapBinout*. It reads six data from each of the three data channels, and converting the 16-bit fixed point data to yield a 6-by-3 matrix. The default timing values are used. The returned matrix will be put in *dataM*, and the number of data is stored in *ret*, which should be 18 in this example.

```
dataM = dapgetm(hDapBinout, [4, 1], 'double', 500, 1000);
```

Get data from the com pipe specified by *hDapBinout*. It gets four data from one data channels. The returned matrix will be put in *dataM*, which is a 4-by-1 matrix with data type 'double'. Because the data type is already double format, there are no conversions. If there are no data available in 500 milliseconds, or if the operation does not return in 1000 milliseconds, the service will abort operation returning whatever data is available.

See Also

dapgbuf, dapputm

dapgstr

Get a string of characters from a Data Acquisition Processor communication pipe.

```
[outputStr <, numBytes>] = dapgstr(handle <, timewait>)
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be opened with read access.

timewait

Longest time to wait for a complete string.
Double.
Optional, can be zero. Default is 1000 milliseconds.

Return Values

outputStr

Characters read from the DAP com pipe.
String.

numBytes

Number of bytes read from the target pipe if successful; 0 if function failed.
Double.
Optional.

Description

This function reads a line of text from the com pipe specified by *handle* and returns it as a MATLAB string in *outputStr*. The parameter *timewait* specifies the longest allowed time to complete this operation. The default value is 1000 milliseconds. The number of characters received is indicated by return argument *numBytes*.

This function depends on algorithms in DAPIO32 to recognize the ends of strings. The text must come from the DAPL system or from a MS-DOS format text file. The DAPIO32 system uses carriage return characters (CR) rather than the modern convention of using endline (LF) characters to terminate text lines. Any text files written on Linux or Unix systems must be specifically written with MS-DOS CR-LF line terminations or this operation cannot read them.

The processing rules used by DAPIO32 are:

- 1) LF characters (ordinary endline) are ignored and not counted.
- 2) CR characters are counted but not returned in the string text, and they terminate a line.

Consequently,

- 1) if *numBytes* equals the string length, the function failed to find a valid end of line and probably returned multiple lines or a partial line.
- 2) if *numBytes* is larger than the actual string length by one, the function succeeded.
- 3) if the line terminations are wrong, this function continues reading until *timewait* because it cannot recognize the ends of lines.

Example

```
textFromDap = dapgstr(hDapSysout)
```

Get a string of characters from the DAP com pipe specified by the handle `hDapSysout`, and stores the string in the return argument `textFromDap`.

```
[textFromDap, ret] = dapgstr(hDapSysout, 100)
```

Get a string of characters from the DAP com pipe specified by the handle `hDapSysout` and store the string in the return argument `textFromDap`. The maximum time to wait for a complete string is 100 milliseconds, or 0.1 seconds. The number of characters in the string plus one is saved in the return argument `ret`.

See Also

`dappstr`

dapmdin

Install a module to DAP board(s).

```
<boolean> = dapmdin(handle, 'filename' <, 'flags'>)
```

Parameters

handle

Handle to a DAP or a server.
Double.
Must be opened with write access.

'filename'

UNC path name of the module to be installed.
String.

'flags'

A list of bitmap option specifying the installation mode.
String.
Optional. Must be *'nocopy'*, *'noreplace'*, *'noload'*, *'forceregister'* or *'forceload'*. See *bmFlags* as described under *DapModuleInstall* in DAPIO32 Reference Manual.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function installs a module from *'filename'* to the DAP(s) specified by *handle*. If *handle* is a DAP handle, it installs the module to the target DAP board. If *handle* is a server handle, it installs the module to all DAP boards on the server. This function allows the user to specify the installation mode by setting the *'flags'* parameter. This operation is persistent. Once a module is installed, it stays until the module is uninstalled. For more information, please see *DapModuleInstall* in DAPIO32 Reference Manual.

Example

```
ret = dapmdin(hdap, 'c:\dapliir.dlm');
```

Install *dapliir* module in the DAP specified by *hdap* with default installation mode: copying the file to the default module directory created by the *SETUP* program, replacing an existing installed module, loading the module to the target DAP board(s) after installation, and not to force the installation and loading of the module.

```
ret = dapmdin(hdap, 'c:\dapliir.dlm', '');
```

Similarly, install *dapliir* module in the DAP specified by *hdap* with default installation mode as described in the above example.

```
ret = dapmdin(hdap, 'c:\dapliir.dlm', 'noCopy noLoad');
```

To install *dapliir* module in to the DAP specified by *hdap* with options to use the module in its original location and defer loading.

See Also

daprint, *dapmduin*, *dapmdld*, *dapmduld*

dapmldd

Load a module to DAP board(s).

```
<boolean> = dapmldd(handle, 'filename' <,'flags'>)
```

Parameters

handle

Handle to a DAP or a server.
Double.
Must be opened with write access.

'filename'

UNC path name of the module to be loaded.
String.

'flags'

A list of bitmap option specifies the loading mode.
String.
Optional. Must be 'noreplace' or 'forceload'. See *bmFlags* as described under *DapModuleLoad* in DAPIO32 Reference Manual.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function loads a module from 'filename' to the DAP(s) specified by *handle* without installing it into the system. If *handle* is a DAP handle, it loads the module to the target DAP board. If *handle* is a server handle, it loads the module to all DAP boards on the server. This function allows the user to specify the loading mode by setting the 'flags' parameter. This operation is not persistent. Once the target DAP(s) are reinitialized, it is necessary to load the module again if it is not installed. For more information, please see *DapModuleLoad* in DAPIO32 Reference Manual.

Example

```
ret = dapmldd(hdap, 'c:\dapliir.dlm');
```

Load *dapliir* module to the DAP specified by *hdap* with default loading mode: replacing an existing loaded module, and not to force loading of the module to the target DAP board(s).

```
ret = dapmldd(hdap, 'c:\dapliir.dlm', 'noreplace');
```

Similarly, load *dapliir* module to the DAP specified by *hdap* with option to not replace any previously copy of the module.

See Also

daprint, dapmdin, dapmduin, dapmduld

dapmduin

Uninstall a module from DAP board(s).

```
<boolean> = dapmduin(handle, 'filename' <,>'flags'>)
```

Parameters

handle

Handle to a DAP or a server.
Double.
Must be opened with write access.

'filename'

UNC path name of the module to be uninstalled.
String.

'flags'

A list of bitmap option specifies the uninstallation mode.
String.
Optional. Must be 'noload', 'forceregister', 'forceload' or 'removedependents'. See *bmFlags* as described under *DapModuleUninstall* in DAPIO32 Reference Manual.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function uninstalls a module '*filename*' from the DAP(s) specified by *handle*. If *handle* is a DAP handle, it uninstalls the module from the target DAP board. If *handle* is a server handle, it uninstalls the module from all DAP boards on the server. This function allows the user to specify the uninstallation mode by setting the '*flags*' parameter. This operation is persistent. Once a module is uninstalled, it is gone until the module is reinstalled. For more information, please see *DapModuleUninstall* in DAPIO32 Reference Manual.

Example

```
ret = dapmduin(hdap, 'c:\dapliir.dlm');
```

To uninstall *dapliir* module from the DAP specified by *hdap* with default uninstallation mode: unloading the module from the target DAP boards(s) after uninstall, not to force uninstall and unloading the module, and not to remove dependents.

```
ret = dapmduin(hdap, 'c:\dapliir.dlm', 'noload');
```

Similarly, to uninstall *dapliir* module from the DAP specified by *hdap* with user-specified uninstalling mode of not to unload the module from the DAP.

See Also

daprint, dapmdin, dapmld, dapmduld

dapmduld

Unload a module from DAP board(s).

```
<boolean> = dapmduld(handle, 'filename' <,>'flags'>)
```

Parameters

handle

Handle to a DAP or a server.
Double.
Must be opened with write access.

'filename'

UNC path name of the module to be unloaded.
String.

'flags'

A list of bitmap option specifying the loading mode.
String.
Optional. Must be *'forceload'* or *'removedependents'*. See *bmFlags* as described under *DapModuleUnload* in DAPIO32 Reference Manual.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function unloads a module *'filename'* from the DAP(s) specified by *handle*. If *handle* is a DAP handle, it unloads the module from the target DAP board. If *handle* is a server handle, it unloads the module from all DAP boards on the server. This function allows the user to specify the unloading mode by setting the *'flags'* parameter. For more information, please see *DapModuleUnload* in DAPIO32 Reference Manual.

Example

```
ret = dapmduld(hdap, 'c:\dapliir.dlm');
```

Unload *dapliir* module from the DAP specified by *hdap* with default unloading mode: not to force unloading the module from the target DAP boards(s), and not to remove dependents of the module during unload.

```
ret = dapmduld(hdap, 'c:\dapliir.dlm', 'removedependents');
```

Similarly, unload *dapliir* module from the DAP specified by *hdap* with user-specified unloading mode of removing dependents of the module during unload.

See Also

daprnit, *dapmdin*, *dapmduin*, *dapmdd*

daplogpd

Initiates a disk logging session between a DAP com pipe and a disk file.

```
<boolean> = daplogpd(handle, 'flag', 'filename', 'fileShareMode', 'openFlags', 'fileFlagAttr', maxCount,  
    blockSize <, [length, timewait, timeout]> )
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be an output pipe from a DAP opened with diskio access.

'flag'

A list of bitmap options specifying the behavior of disklog operation.
String.
Must be 'ServerSide', 'FlushBefore', 'FlushAfter', 'MirrorLog', 'AppendData', or 'BlockTransfer'. See *bmFlags* under TDapPipeDiskLog in DAPIO32 Reference Manual.

'filename'

UNC path name of a primary disk log file and optionally a secondary mirror disk logfile.
String.

'fileShareMode'

A list of bitmap options specifying the file share properties of the disk log file.
String.
Must be 'read', or 'write'. See *dwFileShareMode* under TDapPipeDiskLog in DAPIO32 Reference Manual.

'openFlags'

A list of bitmap options specifying how to handle file opening.
String.
Must be 'CreateNew', 'CreateAlways', 'OpenAlways', or 'OpenExisting'. See *dwOpenFlags* under TDapPipeDiskLog in DAPIO32 Reference Manual.

'fileFlagAttr'

A list of bitmap options specifying additional file attributes.
String.
Must be 'normal', 'encrypted', 'writethrough', or 'sequentialscan'. See *dwFileFlagsAttributes* under TDapPipeDiskLog in DAPIO32 Reference Manual.

maxCount

Maximum number of bytes to write to the log file.
Double.
Default is zero.

blocksize

Minimum amount of data in bytes to write to the log file at one time.
Double.
Use this field to optimize disk transfer. The default value is 8192.

length

Amount of data in bytes to get from the com pipe at a time.
Double.
Optional, must be greater than or equal to zero. Default is 8192 bytes.

timewait

Longest time the log operation can be blocked waiting for data.

Double.

Optional. Default is 1000 millisecond.

timeout

Longest time for the log operation to complete.

Double

Optional. Default is zero, which means no time out.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function initiates a disk logging session between a DAP com pipe identified by *handle* and a disk file identified by '*filename*'. The parameter '*flags*' controls the behavior of disk-logging operation. The parameter '*fileShareMode*' defines how to open the file. The parameter '*openFlags*' defines how the file is shared with other application. The parameter *blocksize* specifies how many data to write to '*filename*' at a time. The operation will get *length* bytes data from the com pipe at a time. Disk logging continues until the number of bytes specified in the *maxCount* has been logged. If *maxCount* is zero, logging continues indefinitely until the *handle* is closed using *dapclose*. The optional parameters *length*, *timewait*, *timeout* are used as a group to control the timing of data transfer. If one of the optional parameter is used, all three must present.

This function does not use a handle for the log file. Instead, a *dapopen* function establishes a connection to a com pipe, then later this function completes the association with a disk file. Once the association with the file is created, the way to disassociate from the file is to close the handle passed to the command by using *dapclose*.

Disk logging operation occurs in the background. When this function returns, the logging process is started, but this does not mean the logging is finished. While logging is processing in the background, the MATLAB application can continue doing other processing. Calling *dapquery* function for the current status can check logging operation has finished.

If '*MirrorLog*' option is included in the '*flag*' bitmask option list, a secondary file name must be concatenated to the primary file name within the parameter '*filename*'. Separate the file names with a semicolon.

For more information, please see *DapPipeDiskLog*, *TDapPipeDiskLog* and *TDapBufferGetEx* in the DAPIO32 Reference Manual.

Example

```
daplogpd(hDap, 'serverSide flushBefore', 'c:\logfile.log', '', '', '', 0, 0)
```

Log the data from the DAP specified by handle *hDap* to a file called *logfile.log* in c: drive. The target pipe specified by *hDap* is first flushed. Default values for other parameters are used, so *blocksize* is 8192 bytes, no *timeout*, and data must arrive within one second.

```
ret = daplogpd(hDap, '', 'c:\logfile.log', 'read', 'CreateAlways', 'normal', 0, 0, 4096, 10, 0)
```

Log the data from the DAP specified by handle *hDap* to a file called *logfile.log* in c: drive. The application will create a file called *logfile.log*, or overwrite if the file already exists. Other application can have a read access to *logfile.log* only. This example get 4096 bytes from the DAP at a time. If there is no data available in 10 milliseconds, the service aborts the operation. The function termination status is stored in the return argument *ret*.

```
daplogpd(hDap, 'serverSide MirrorLog', 'c:\file.log;c:\mirror.log', '', '', '', 0, 0)
```

Log the data from the DAP specified by handle hDap to a file called file.log. Mirror logging creates a copy of the logged data in mirror.log. Default values for other parameters are used, so blocksize is 8192 bytes, no timeout, and data must arrive within one second.

See Also
dapquery

dapopen

Opens a handle to the target Server, DAP or communication pipe.

```
handle = dapopen('unc_path', 'code')
```

Parameters

'unc_path'

UNC path of the Server, DAP or com pipe.
String.

'code'

Operating mode of the target to be opened.
String.
Must be *'write'*, *'read'*, *'query'* or *'diskio'*.

Return Values

handle

Handle of the target DAP, com pipe or server.
Double.
Nonzero if successful, 0 if function failed.

Description

This function establishes a connection to a server, DAP or communication pipe. The path and name to the device are specified in a UNC (Universal Naming convention) notation in parameter *'unc_path.'* The *'code'* parameter specifies the access mode. The function returns a number *handle* that is used by other functions to access the device in a manner similar to a data file.

The *'code'* parameter has to be one of the following values.

'write' opens the pipe for writing

'read' opens the pipe for reading

'query' opens the pipe for query

'diskio' opens the pipe for disk I/O access

The handle of a com pipe opened with *'diskio'* can be used later to initiate and terminate a direct pipe disk I/O session using *dapfedpd* and *daplogpd* functions. It can also be used to query disk I/O status using the *dapquery* function. The drivers DAPCell Local or DAPCell Server|Client must be used for *'diskio'* option in the *'code'* parameter. For more information, please see *DapHandleOpen* in the DAPIO32 Reference Manual.

Example

```
hDapBinOut = dapopen('\\Server2\Dap0$binout', 'read')
```

Open the communication pipe *\$binout*, which is on *Dap0* located in *Server2*, for read access. An open handle to the target pipe is retained in *hDapBinOut*.

```
hDap0 = dapopen('\\.\Dap0', 'query')
```

Open the handle to the DAP at index 0 on the local machine for querying information. An open handle to the target DAP is retained in *hDap0*.

```
hPC16 = dapopen('\\PC16', 'query')
```

Open the handle to the remote server on *PC16* for querying information. An open handle is retained in *hPC16*.

See Also

dapclose, dapedpd, daplogpd, dapquery

dappavl

Get the number of bytes space available in a pipe for writing.

```
numBytes = dappavl(handle)
```

Parameters

handle

Handle to a DAP com pipe.

Double.

Must be an input pipe to a DAP.

Return Values

numBytes

Number of bytes space available in the target pipe for writing if successful; -1 if function failed.

Double.

Description

This function gets the number of bytes spaces available for writing in a com pipe specified by *handle*. It always returns a count in bytes to the return argument *numbytes* regardless of the actual width of pipe elements. An application is always safe to write that number of data bytes to the pipe without being blocked.

The result *numBytes* is not an indication of the full buffering capacity. In most applications, it is better to use the `dapputm` or `dappbuf` functions with a timeout parameter to make data transfers more efficiently.

Example

```
ret = dappavl(hDapBinin)
```

Get the number of byte spaces available in the pipe specified by handle `hDapBinin` for writing. The returned number of byte space is being stored in the return value argument `ret`.

See Also

`dapgavl`

dappstr

Writes a string of characters to a Data Acquisition Processor com pipe.

```
<numBytes> = dappstr(handle, 'inputStr')
```

Parameters

handle

Handle of the DAP, com pipe.

Double.

It has to be a handle to a DAP com pipe, opened with write access.

'inputStr'

Characters to write to the DAP com pipe.

String.

Return Values

numBytes

Number of bytes put to the target pipe if successful; 0 if function failed.

Double.

Optional.

Description

This function writes a string of characters represented by *'inputStr'* to the target com pipe specifies by *handle*. At the end of the string, this function attaches a carriage return as a line termination character. Each character written corresponds to one byte. The number of bytes written is stored in the optional return argument *numBytes*, which include one additional byte for line termination character. If no return argument is given, error checking is handled internally. An error message will be displayed when errors occur.

The most common use of this function is to send commands directly to the DAPL command line interpreter.

Note: The termination of text lines by a carriage return character is an old file format accepted by the DAPL system but not by most other operating systems.

Example

```
dappstr(hDapSysIn, 'START')
```

Send a START command to the DAP through the DAPL command text pipe using pipe handle hDapSysIn.

```
ret = dappstr(hDapSysIn, 'START')
```

The same as the previous example, but the number of byte written is stored in the return argument ret. The value of ret is six if the function is successful.

See Also

dapgstr

dapputm

Put a matrix of data to a communication pipe.

```
<numData> = dapputm(handle, dataM, 'dataType' <, timeWait, timeOut>)
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be opened with write access.

dataM

A data matrix to be put to a com pipe.
MATLAB matrix of double.

'dataType'

The type of data to put into the pipe.
String.
Must be 'int16', 'int32', 'double' or 'float'.

timewait

Longest time the put operation can wait for available space.
Double.
Optional. Default is 100 milliseconds. If one of the optional parameters is used, both must present.

timeout

Longest time for the put operation to complete.
Double.
Optional. Default is 20,000 milliseconds. If one of the optional parameters is used, both must present.

Return Values

numData

Number of data put to the target pipe if successful; -1 if function failed.
Double.
Optional.

Description

When this function executes, the data in matrix *dataM* will be written to the com pipe specified by *handle*. The matrix is sent by column. The parameter 'dataType' specifies the type of data to be written.

The 'dataType' parameter has to be one of the following values.

'int16' specifies 16-bit integer, represented by two bytes

'int32' specifies 32-bit integer, represented by four bytes

'double' specifies double, represented by eight bytes

'float' specifies floating point data, represented by four bytes

The values in the matrix are converted to the type specified by 'dataType'. Conversions can cause truncation or range errors, so be careful to round and scale appropriately, before writing 'int16' or 'int32' data types.

This function also allows an application to specify *timewait* and *timeout* parameters to control the behavior of the operation. If the put operation fails to complete in *timeout* milliseconds, or if the pipe remains full for *timewait* milliseconds, the function returns immediately. These two parameters are used in pairs. If one is used, the other must be present.

The number of data written to the DAP com pipe is stored in the optional return argument *numData*. If the function fails, the value of *numData* is -1.

Example

```
inputData = [129.9, 0.0263, 36.66, 0.038, 140.0, 8.88;  
            12992, 26333, -32666, 38000, 14000, 888;  
            12993, 26333, 32666, 38000, 14000, 888];  
numDataRead = dapputm(hDap, inputData, 'double');
```

Put a data matrix *inputData* as data type 'double' to the com pipe specified by *hDap*. The number of matrix elements put to the com pipe is stored in the return argument *numDataRead*.

```
numDataRead = dapputm(hDap, inputData, 'int32', 10000, 0);
```

The same as the previous example, except the data is converted to type 'int32', and the value of *timeWait* is set to 10000 milliseconds. If no data appear in 10,000 milliseconds, the service will abort the operation.

See Also

[dapbuf](#), [dapgetm](#)

dapquery

Queries for information associated with a handle.

```
['queryResult' <, boolean>] = dapquery(handle, 'queryKey')
```

Parameters

handle

Handle to a DAP, com pipe or server.
Double.
Must be an output pipe from a DAP opened with query access.

'queryKey'

Key associated with the information to be queried.
String.
For a complete set of keys available and descriptions, please see TDapHandleQuery and DapHandleQuery in the DAPIO32 Reference Manual.

Return Values

'queryResult'

Result from the query associate with *'queryKey'*.
String or Double.

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional.

Description

This function queries for information associated with the DAP, com pipe or server specified by handle. The parameter handle has to be previously opened with dapopen by *'query'* attribute.

There are four types of handles.

- NULL handle
- Handle to a server PC
- Handle to a Data Acquisition Processor
- Handle to a communication pipe on a Data Acquisition Processor.

A NULL handle has the value 0. Querying about a NULL handle gives information that is not specific to a particular server, a particular Data Acquisition Processor, or a particular pipe.

Depending on queryKey, the returned queryResult can be either a string or a double. If a return value of double is desired, the MATLAB function Str2Num can be used to convert the return value to a number. To find out the type of the returned queryResult, the MATLAB function Class can be used.

For more information and a detail example, please see TDapHandleQuery and DapHandleQuery in the DAPIO32 Reference Manual.

Example

```
hdap = dapopen('\\\\.\\dap0', 'query');  
[qResult, ret] = dapquery(hdap, 'DapName');
```

Query the name of dap0 in the local machine. The name of the target dap will be stored in the return argument 'qResult'. If successful, the return value ret is one.

```
[qResult, ret] = dapquery(0, 'ClientVersion');
```

Query the version of the DAPIO32 client software. The version will be stored in the return argument 'qResult'. If successful, the return value ret is one.

See Also

dapcpc, daplogpd, dapfedpd

daprecnf

Redirects the output of `dapcnfig` to a specified text file.

```
<boolean> = daprecnf('filename')
```

Parameters

'filename'

UNC path and filename of the destination file of redirection.

String.

Can be an empty string represented by a pair of single quote('').

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function is meant primarily for testing. It allows an application developer to redirect output of DAPL configuration file through the parameter substitution performed by `dapcnfig` without sending anything to the DAP board. Execute `daprecnf` function before `dapcnfig`. Follow up by executing `daprecnf` with an empty string to close existing redirection.

This function does not use a file handle. Instead, this function temporarily uses the named file in place of the DAPL system command pipe. The file is disassociated when redirection is closed.

Please see `DapConfigRedirect` in DAPIO32 Reference Manual for a more information.

Example

```
ret = daprecnf('c:\redirOutput.txt');  
dapcnfig(hdapSysIn, 'inOut.dap');  
daprecnf('');
```

Redirects the DAPL file being downloaded by `dapcnfig` to a text file called `redirOutput` in C: drive. Stores the function termination status in the return argument `ret`. The DAPL command file `inOut.dap` is sent to file `redirOutput.txt` on the c drive rather than to the com pipe `hdapSysIn`. Closes the redirection using `daprecnf` with an empty string.

See Also

`dapcnfig`

dapreset

Resets the target DAP board(s).

```
<boolean> = dapreset(handle)
```

Parameters

handle

Handle to DAP(s).

Double. Must be opened with write, read or query access.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

This function sends a DAPL command RESET to reset the DAP represented by *handle*. This is a simpler and safer way to reset a DAP than to send commands through the DAPL command pipe. This function does not return until the RESET operation is complete. The DAP handle has to be opened by `dapopen` with any access. If `$$Sysin` or `$$Sysout` pipes for a target DAP are already opened by any other running application with write, read or query access, and the MATLAB application opens the pipe using diskio access, this function cannot reset the DAP and will report failure. For more information, please see `DapReset` in DAPIO32 Reference Manual.

Example

```
ret = dapreset(hDap);
```

Reset the DAP specified by hDap.

See Also

`daprint`

daprint

Reloads the DAPL system and all installed modules.

`<boolean> = daprint(handle)`

Parameters

handle

Handle to DAP(s) or server.

Double.

Must be opened with read or write access.

Return Values

boolean

Function termination status.

Double; 1 for TRUE and 0 for FALSE.

Optional.

Description

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. If *handle* is a DAP handle, this reinitializes the target DAP board only. If it is a server handle, this reinitializes all the DAP boards on the server.

Note: This is a destructive operation. All data on the target DAP board(s) will be lost. Also, this invalidates any opened handles referring to this DAP and its com pipes.

Example

```
ret = daprint(hDap);
```

Cleans and reinitializes a DAP referred by handle *hDap*, and stores the function termination status in the return argument *ret*.

See Also

dapreset, dapmdin, dapmduin, dapmdld, dapmduld

dapsetpa

Initializes a program-defined parameter.

```
<boolean> = dapsetpa(paraNumber, 'paraValue')
```

Parameters

paraNumber

The parameter number to be set.
Integer.
Must be in the range of 1 to 100.

'paraValue'

The parameter value to be set.
String.

Return Values

boolean

Function termination status.
Double; 1 for TRUE and 0 for FALSE.
Optional

Description

This function initializes a program-defined parameter *paraNumber* to the value *paraValue*, which overwrites the default parameters in the DAPL file. The parameter list is stored in memory. If *'paraValue'* is an empty string with the form *' '*, the parameter value is cleared. After this function get calls, the MEX function *dapcnfig* should be used to download the DAPL command to the DAP.

Example

```
dapsetpa(2, '');
```

Sets the value of parameter 2 to an empty string in memory.

```
ret = dapsetpa(35, '500');  
dapcnfig(hDapSysin, 'c:\test.dap');
```

Sets the value of parameter 35 to *'500'* in memory. Stores the function termination status in the return argument *ret*. Sends the DAPL file *test.dap* in C: drive to the DAP com pipe specified by *hDapSysin*, substituting the text *'500'* for each occurrence of the text *'%35'*.

See Also

dapclrpa

4. Examples

The DAPtools for Matlab includes examples on using the MEX functions to interface with the DAP board.

Set 1

Example set 1 demonstrates the use of a Data Acquisition Processor with MATLAB. The set contains three MATLAB files, `init.m`, `data.m` and `stop.m`, and one DAPL file `data.dap`.

1. `init.m` – shows how to initialize communication with the DAP from MATLAB environment.
2. `data.m` – shows how to read data values from input channels into matrices.
3. `stop.m` – shows how to terminate communication with the DAP.
4. `data.dap` – shows how to configure the DAP.

File INIT.M

This file shows an example on how to initialize communication with the Data Acquisition Processor.

```
%% File INIT.M %%
% Open text and binary handles to ACCEL device
texthandle = dapopen('\\.\dap0\sysin', 'write')
if texthandle == -1
    error('Error opening DAP text handle')
end
binaryhandle = dapopen('\\.\dap0$binout', 'read')
if binaryhandle == -1
    error('Error opening DAP binary handle')
end

% Send a command to reset the DAP
dappstr(texthandle, 'RESET');
% Flush old DAP data in binary pipe
dapflshi(binaryhandle);
```

File DATA.M

This file shows how to configure the Data Acquisition Processor with the DAPL file `data.dap`, and reads data from the Data Acquisition Processor into matrices in MATLAB environment.

```
%% File DATA.M %%
% Configure DAP using DAPL command file DATA.DAP
cnfg = dapcnfig(texthandle, 'data.dap');
if cnfg < 1
    % Print error message
    error('Error configuring DAP')
end

% Get DAP data
% Read data from DAP
channel0 = dapgetm(binaryhandle, [200, 1], 'int16')
channel1 = dapgetm(binaryhandle, [200, 1], 'int16')
```

File STOP.M

This file shows how to terminate communication and closes all handles to the Data Acquisition Processor.

```
%% File STOP.M %%

% Stop DAP communication by sending STOP command to the DAP
dapstr(texthandle,'STOP');

% Close text and binary handles to ACCEL device
textclose = dapclose(texthandle)
if textclose==-1
    error('Error closing DAP text handle')
end
binclose = dapclose (binaryhandle)
if binclose==-1
    error('Error closing DAP binary handle')
end
```

DAPL Command File: DATA.DAP

This file shows the DAPL configuration for the Data Acquisition Processor.

```
;; DAPL file, DATA.DAP ;;

reset
ifdef a 2
    set ip0 s0    ; Sample analog input, s0
    set ip1 s1    ; Sample analog input, s1
    time 5000    ; Time of 5 msec
    count 1000   ; Take 1000 samples
end
pdef b
; Send data through binary communication pipe
merge(ip0, ip1, $binout)
end
start a, b
```

Running the Examples

To run the example at the MATLAB command window, type the following in the MATLAB command window in sequence:

```
init
data
stop
```

Set 2

These examples exercise most of the commands provided in DAPtools for MATLAB. These examples can be run by typing the name of the MATLAB file at the MATLAB command window without the extension, for example, LOGFILE.

Filename	Description
LOGFILE.M	Logs one channel of data from the DAP to the PC's hard disk.
DSTRING.M	Writes and reads strings and error messages from the DAP.
BIWAY.M	Writes matrix data to the DAP and reads processed data from the DAP.
WAVES.M	Uses the DAP as a function generator to generate a sine wave and triangular wave of different frequencies.
FOURIER.M	Gets FFT data from the DAP and performs 3-D plots of the data using MATLAB's graphics.

All the above examples open the DAP file handles, get DAP data, and close the DAP file handles. To force a running example to stop, you can use CTRL+C. However, the opened DAP file handles are not closed. You should run STOP.M to close both the text and binary file handles.

Reading Binary File

You can use MATLAB to read binary data previously recorded by MATLAB or another application such as DAPstudio or DAPview for Windows. The following commands show how to read the file DATABIN.LOG, which has logged binary data, in the MATLAB environment.

At the MATLAB command window, type the following:

```
fid = fopen('databin.log', 'rb');  
bindata = (fread (fid, [n,inf], 'short'))  
fclose(fid);
```

In the function MATLAB fread, the parameter n is the number of channels or pipes merged when writing the recorded data file. The parameter inf means read to the end of the file.

5. Appendix A – Obsolete Functions

The DAP MEX functions listed below are obsolete. They are not supported by the current release.

Obsolete	Replaced by	DAPIO32 DLL Routines
dapflush	dapflshi	DapInputFlushEx
dapgbuf	dapgetm	DapBufferGetEx
dappbuf	dapputm	DapBufferPutEx

dapflush

This function is renamed as `dapflshi`. Please see `dapflshi` for a function specification.

dapgbuf

Read a block of data from a Data Acquisition Processor com pipe into a data matrix in MATLAB.

```
[dataM <, numBytes>] = dapgbuf(handle, length <-, timewait, timeout>)
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be opened with read access.

length

Number of data to read.
Double.

timewait

Longest time the application can be blocked waiting for data.
Double.
Optional, can be zero.

timeout

Longest time for the get operation to complete.
Double.
Optional.

Return Values

dataM

Data sent from the DAP.
Double.
Data array with dimension length-by-1.

numBytes

Number of bytes read from the target pipe if successful; -1 if function failed.
Double.
Optional

Description

This function allows an application to specify a number of sample values as a data block to read. Only 16 bit word values are supported. It will read as many data bytes as possible within the specified range. This function also allows an application to specify *timewait* and *timeout* parameters to control the behavior of the function. These two parameters are used in pairs. If one is used, the other must be present. This function will return immediately with the data read so far if a time-out occurs. The returned data will be converted from word type to double and stored in a *length*-by-1 matrix. The number of bytes read is returned in the parameter *numBytes*.

Example

```
dataM = dapgbuf(hDapBinout, 64);
```

Read 64 short int data from the com pipe specified by hDapBinout. The returned data are placed in return parameter dataM, which is a 64-by-1 matrix.

```
[dataM, numBytes] = dapgbuf(binouthandle, 10, 200, 0);
```

Read 10 short int data from the com pipe specified by hDapBinout. The returned data are placed in the return parameter dataM, which is a 10-by-1 matrix. The number of bytes read is stored in the return parameter numBytes.

It should be 20, two bytes per short in data, in this case. The *timewait* is set to 200 milliseconds, which means if no data show up in 200 milliseconds, the service aborts this operation. Since *timeout* is zero, the service ignores it.

dappbuf

Write a block of data from MATLAB to the target pipe of a Data Acquisition Processor.

```
<numBytes> = dappbuf(handle <, length>, dataM <, timewait, timeout>)
```

Parameters

handle

Handle to a DAP com pipe.
Double.
Must be opened with write access.

length

Number of data to write.
Double.
Optional.

dataM

Data being write from the DAP.
Int, double, float.

timewait

Longest time the put operation can wait for available space.
Double.
Optional.

timeout

Longest time for the put operation to complete.
Double.
Optional.

Return Values

numBytes

Number of bytes written to the target pipe if successful; -1 if function failed.
Double.
Optional.

Description

This function writes an array of data to the target pipe. The data will be converted from double to short int. If *length* is not specified, this function sends all the values in *dataM*. This function also allows an application to specify *timewait* and *timeout* parameters to control the behavior of the operation. These two parameters are used in pairs. If one is used, the other must be present. The number of data written is returned in the optional parameter *numBytes*.

Example

```
dataM = [1 2 3 4 5 6 7 8 9 10];  
ret = dappbuf(hDapBinin, 20, dataM);
```

Writes the data matrix *dataM*, which has 10 data, to a com pipe specified by *hDapBinin*. The data will be converted from double to short int, two bytes per data. Therefore, the length of writing 10 data is 20 bytes. The number of bytes actually written to the target pipe is returned in the optional parameter *ret*, which should be 20 if the function succeeds.

```
dataM = [1 2 3 4 5 6 7 8 9 10];  
dappbuf(hDapBinin, 10, dataM, 0, 10);
```

Write the first five data of `dataM` to the com pipe specified by `hDapBinin`. The `timewait` and `timeout` are set to zero and ten milliseconds respectively. The service aborts the operation if the function does not return in ten milliseconds.