Old Forms of Commands

Supplement to
DAPL 2000 Manual, Version 6

Version 1.00

Microstar Laboratories, Inc.

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

Contents

Introduction

The DAPL 2000 version 2.5 release introduces some new versions of common processing commands with extended abilities to process data in 16-bit, 32-bit, 32-bit standard floating point, or 64-bit standard double-precision floating point. These commands are distributed in the form of a 32-bit downloadable command module, `DAPLCMDS.DLM`.

With the new abilities of the commands to process more data types, there are also new internal complexities that can change command behaviors in subtle ways. Most applications will find it impossible to observe any differences. However, some finely-tuned applications pushing the extreme limits might be affected by small differences in execution speed. For these applications, it is possible to revert to the previous versions of the processing commands, restoring the original behaviors but losing any new abilities to process the additional data types.

The following updated commands are provided in the `DAPLCMDS.DLM` module:

```
AVERAGE
BAVERAGE
COPY
MERGE
BMERGE
SEPARATE
SKIP
```

If an application requires the old behaviors of these commands, disable the new versions of these commands, by performing the following procedure.

1. Run the ACCEL32 control panel application.
2. Select the MODULES tab.
3. Highlight the `DAPLCMDS.DLM` module in the Installed Modules window.
4. Click the DELETE button.
5. Click OK in the options window.
6. Back in the ACCEL32 control panel application window, select the CONTROL tab.
7. Click the STOP button and wait for the DAPcell Local Server to stop.
8. Click the START button and wait for the DAPL system to reload, this time without loading the `DAPLCMDS` module.

Disabling loading of the DAPLCMDS module will have other side effects. Several commands previously distributed in 16-bit binary form are now available only through the DAPLCMDS module. Disabling the module makes these commands unavailable. Also, some related triggering commands become unavailable.

```
COMPRESS
DEXPAND
TAND
TOR
TCOLLATE
TOGGLE
TOGGWT
```

In case you find it necessary to try disabling the DAPLCMDS module, the command pages that follow describe the old forms of the commands, as they will operate after the new forms are disabled.

Warnings and Limitations

- DAPL 2000 is a 32-bit operating system and is compatible only with certain Data Acquisition Processors that have a 32-bit processor.
- The DAPLCMDS.DLM module can be used with the DAPL 2000 operating system version 2.5 or newer. There is no assurance of compatibility with any prior operating system versions.
- The DAPLCMDS.DLM module makes extensive use of floating point features when using floating point data types. These features will be very slow when executed on Data Acquisition Processor models with processors having no floating point hardware features.
- Future versions of the DAPL 2000 system will not be able to support the historical old forms of the processing commands. At that time, disabling the DAPLCMNDS module will be inadvisable, as this would make many processing commands unavailable.

Old Forms of Commands

AVERAGE

Define a task that computes the arithmetic mean of a group of samples.

AVERAGE *(<in_pipe>, <count>, <out_pipe>)*

Parameters

<in_pipe>
Input data pipe.
WORD PIPE

<count>
The number of samples from which the arithmetic mean is computed.
WORD CONSTANT

<out_pipe>
Output pipe for the averaged data.
WORD PIPE

Description

AVERAGE computes the arithmetic mean of *<count>* samples. Data values are received from *<in_pipe>* and results are sent to *<out_pipe>*. AVERAGE is useful for data compression and noise reduction.

Examples

```
AVERAGE (IPIPE0, 10, P1)
```

Average groups of 10 values from input channel pipe 0 and send the averages to pipe P1.

```
AVERAGE (P2, 4, P3)
```

Average groups of 4 values from pipe P2 and send the averages to pipe P3.

See Also

BAVERAGE, FIRFILTER, RAVERAGE

BAVERAGE

Define a task that computes multiple arithmetic means for multiplexed data.

BAVERAGE *(<in_pipe>, <n>, <m>, <out_pipe>)*

Parameters

<in_pipe>
Input data pipe.
`WORD PIPE | LONG PIPE`

<n>
The number of values in a block.
`WORD CONSTANT`

<m>
The number of blocks of data to be averaged.
`WORD CONSTANT`

<out_pipe>
Output pipe for averaged data blocks.
`WORD PIPE | LONG PIPE`

Description

<n> and *<m>* are positive nonzero integers. *<n>* indicates the number of values in the block. BAVERAGE reads *<m>* blocks, averages corresponding points in the blocks, and puts one block of averages into *<out_pipe>*. For every *<n>*<m>* values read, *<n>* values are put into *<out_pipe>*. The maximum size of *<n>* is 8192.

Some common applications of BAVERAGE:

- reducing noise in repetitive waveforms
- averaging data from multiple channels in parallel
- reducing noise in FFT power spectra

Example

```
BAVERAGE (P1, 100, 5, P2)
```

Read 5 blocks of 100 values, average corresponding values in the blocks, and send the 100 averages to pipe `P2`.

See Also

`RAVERAGE`, `WAIT`

BMERGE

Define a task that merges blocks of data.

BMERGE *(<in_pipe_0>, ... , <in_pipe_n-1>, <blocksize>, <out_pipe>)*

Parameters

<in_pipe_0>
First input data pipe.
`WORD PIPE | LONG PIPE`

<in_pipe_n-1>
Last input data pipe.
`WORD PIPE | LONG PIPE`

<blocksize>
A number that represents the length of the data blocks.
`WORD CONSTANT`

<out_pipe>
Output pipe for merged data blocks.
`WORD PIPE | LONG PIPE`

Description

BMERGE reads blocks of length *<blocksize>* sequentially from pipes *<in_pipe_0>*, *<in_pipe_1>*, … , *<in_pipe_n-1>*, and writes the blocks to *<out_pipe>*. For blocked data, such as the outputs of **WAIT** or **FFT** tasks, **BMERGE** is more efficient than **MERGE**.

The data types of pipes *<in_pipe_0>*, *<in_pipe_1>* etc., and pipe *<out_pipe>* must all be the same. The maximum size of *<blocksize>* is 8192 for word pipes and 4096 for long pipes.

BMERGE should be used only when *<in_pipe_0>*, *<in_pipe_1>*, … , *<in_pipe_n-1>*, are filled at the same rate. Otherwise, **BMERGEF** should be used.

Example

```
BMERGE (P1, P2, P3, P4, 1024, $BINOUT)
```

Read blocks of 1024 data values sequentially from pipes `P1`, `P2`, `P3`, and `P4`, and send the blocks to the PC through `$BINOUT`.

See Also

`BMERGEF`, `MERGE`, `MERGEF`, `NMERGE`, `SEPARATE`, `SEPARATEF`

COMPRESS

Define a task that encodes data in which changes occur infrequently.

COMPRESS *(<in_pipe>, <n> [, <threshold>]*, <out_pipe>)*

Parameters

<in_pipe>
Input data pipe.
`WORD PIPE`

<n>
A positive constant, less than or equal to 16, specifying the number of data streams read from *<in_pipe>*.
`WORD CONSTANT`

<threshold>
A number or sequence of numbers that represents the threshold value for which to report changes.
`WORD CONSTANT | WORD VARIABLE`

<out_pipe>
Output pipe for encoded data blocks.
`WORD PIPE`

Description

`COMPRESS` encodes data in which changes occur infrequently. `COMPRESS` can monitor a single data stream or a multiplexed data stream such as an input channel pipe list. `COMPRESS` takes data from *<in_pipe>* one value at a time, and compares that value to the corresponding value previously reported for the same channel. If the absolute difference is greater than or equal to the *<threshold>* parameter for that channel, a data block containing the new value is placed into *<out_pipe>*. Otherwise the data value is discarded. Output data blocks are always generated to report the first value from each input stream.

Note: `COMPRESS` is not predefined in DAPL. The code for `COMPRESS` must be downloaded to the Data Acquisition Processor from the 16-bit binary file COMPRESS.BIN. It is installed in the "Microstar Laboratories\Accel32\Dapl2000" directory. Command downloading can be performed from DAPview for Windows or from CDLOAD32. The CDLOAD32 utility is available with the source code in

the DAPDEV\EXAMPLES install directory on the DAPtools CD. When downloading **COMPRESS**, use a stack size of 1000.

The first parameter <*in_pipe*> specifies one or more data streams. The first parameter typically is an input channel pipe list, but it can be any data pipe with multiplexed data.

The second parameter <*n*> is a positive constant, less than or equal to 16, specifying the number of data streams read from <*in_pipe*>. If <*in_pipe*> is an input channel pipe list, <*n*> must be equal to the number of input channel pipes. For a single pipe, <*n*> must be 1. The data streams from the input pipe are numbered 0 through <*n*>-1.

Either one or multiple <*threshold*> parameters must appear next in the parameter list. Each <*threshold*> value is a nonnegative value. If a single <*threshold*> parameter is specified, it will apply to each stream in the input pipe list. If more than one <*threshold*> parameter is specified, there must be one <*threshold*> parameter for each of the <*n*> multiplexed data streams, in sequence.

Each change report written to <*out_pipe*> contains three values. The first value is a 16-bit integer from 0 to <*n*>-1, specifying the stream from which the threshold value was exceeded. A zero indicates the first stream, a 1 indicates the second stream, and so on. The second value is the new 16-bit input data value. The last field is a 32-bit unsigned integer specifying the sample number of the new value, where the data are counted in one channel starting from sample number 0.

Examples

```
COMPRESS (IP0, 1, 100, P1)
```

Send data block to pipe P1 when input values change by more than 100.

```
COMPRESS (IPIPES(0,1,2), 3, 64, 64, 1000, P1)
```

Send data block if input channel pipe 0 or 1 changes by more than 64 or if input channel pipe 2 changes by more than 1000.

See Also

AVERAGE, LIMIT

COPY

Define a task that transfers data from an input pipe to one or more output pipes.

COPY *(<in_pipe>, <out_pipe_1>, ... , <out_pipe_n>)*

Parameters

<in_pipe>
Input data pipe.
`WORD PIPE | LONG PIPE`

<out_pipe_1>
First output pipe for copied data.
`WORD PIPE | LONG PIPE`

<out_pipe_n>
Last output pipe for copied data.
`WORD PIPE | LONG PIPE`

Description

`COPY` transfers each value from *<in_pipe>* to one or more output pipes. As many as 64 output pipes are allowed. The data types of the *<in_pipe>* and *<out_pipe>* parameters must all match.

In addition to moving data between tasks, `COPY` also lets tasks in different configurations share the same data. This gets around the restriction that all tasks reading from a pipe must reside in the same processing procedure.

Example

```
COPY (P1, P2, P3, P4)
```

Copy each value from pipe `P1` to pipes `P2`, `P3`, and `P4`.

See Also

`COPYVEC`, `MERGE`

DEXPAND

Define a task that provides output expansion on a digital port.

DEXPAND *(<in_pipe>, <output_vector>, <out_pipe> [, <type>])*

Parameters

<in_pipe>
Input word pipe.
WORD PIPE

<output_vector>
A vector containing a list of the output pins to which data should be sent.
VECTOR

<out_pipe>
Output pipe for output data.
WORD PIPE

<type>
An optional parameter that specifies the type of output expansion boards.
WORD CONSTANT

Description

DEXPAND provides output expansion on the digital port. *<output_vector>* is a vector containing a list of the expanded output pins to which data should be sent. *<in_pipe>* is a word pipe that contains data to be sent to the output pins; data must appear in the order of the list in *<output_vector>*. For each data value read from *<in_pipe>*, four words specifying the output pin number and the data are written to *<out_pipe>*.

Optional parameter *<type>* specifies the type of output expansion boards. A value of 0 specifies digital output expansion boards. A value of 1 specifies analog output expansion boards. See the hardware documentation for details on output expansion boards.

DEXPAND is used only for synchronous output expansion. The **OUTPORT** command configures asynchronous output expansion. Asynchronous output to the digital output port is not available when **DEXPAND** is used.

DEXPAND is not predefined in DAPL — the code for **DEXPAND** must be downloaded to the Data Acquisition Processor from a disk file in the PC. Command downloading can be performed from DAPview for Windows or from CDLOAD32. The CDLOAD32 utility is available with the source code in the DAPDEV\EXAMPLES install directory on the DAPtools CD. When downloading **DEXPAND**, use a stack size of 1000.

Note: It is possible to stop an output configuration in the middle of a four word output expansion sequence. If another output configuration then is started, the first value written to the expanded output port may be incorrect.

Example

```
DEXPAND(P1, (4, 5, 6, 7), OPIPE0)
```

Prepare data from pipe `P1` for output to digital expansion ports 4, 5, 6, and 7, and send the data to synchronous output channel pipe 0.

See Also

DIGITALOUT, **DACOUT**, **ODEFINE**, **OPTIONS**, **OUTPORT**

MERGE

Define a task that merges data sequentially from multiple input pipes.

MERGE *(<in_pipe_0>, ... , <in_pipe_n-1>, <out_pipe>)*

Parameters

<in_pipe_0>
First input data pipe.
`WORD PIPE | LONG PIPE | FLOAT PIPE`

<in_pipe_n-1>
Last input data pipe.
`WORD PIPE | LONG PIPE | FLOAT PIPE`

<out_pipe>
Output pipe for merged data.
`WORD PIPE | LONG PIPE | FLOAT PIPE`

Description

`MERGE` reads data from one or more input pipes and places the data consecutively into an output pipe. When transferring a word value to a long output pipe, the value is sign extended. When transferring a long value to a word output pipe, two words are placed in the output pipe, with the least significant word first, followed by the most significant word.

It is necessary for the data arrival rates in all pipes *<in_pipe_0>* through *<in_pipe_n-1>* to be equal. If data volumes are different in different pipes, data will backlog in the pipes having higher volumes, causing processing to terminate when capacity is exhausted.

`MERGE` is useful for merging binary data from several pipes to a single communication pipe for transmission to the host computer. `MERGE` is the inverse of `SEPARATE`.

Examples

```
MERGE (P1,P2,P3,P4)
```

Read data from pipes `P1`, `P2`, and `P3`, and place the data consecutively into pipe `P4`.

```
MERGE (P5,P6,P7,$BINOUT)
```

Transfer data from pipes `P5`, `P6`, and `P7` to the binary output communication pipe; send the data values to the host.

See Also

BMERGE, BMERGEF, MERGEF, SEPARATE, SEPARATEF

SEPARATE

Define a task that distributes data consecutively into one or more output pipes.

SEPARATE *(<in_pipe>, <out_pipe_0>, ... , <out_pipe_n-1>)*

Parameters

<in_pipe>
Input data pipe.
`WORD PIPE | LONG PIPE`

<out_pipe_0>
First output pipe for separated data.
`WORD PIPE | LONG PIPE`

<out_pipe_n-1>
Last output pipe for separated data.
`WORD PIPE | LONG PIPE`

Description

`SEPARATE` reads data from *<in_pipe>* and places the data consecutively into one or more output pipes. The first data value is sent to *<out_pipe_0>*, the second data value is sent to *<out_pipe_1>*, etc.

When *<in_pipe>* is a word pipe and the output pipe is a long pipe, two consecutive words, low word then high word, are read from the input pipe and concatenated to form a long output value.

When *<in_pipe>* is a long pipe and an output is a word pipe, the low word of the long value is transferred, and the high word is ignored.

`SEPARATE` is useful for reading binary data from a host computer and splitting the binary data stream into several pipes for processing. `SEPARATE` is the inverse of `MERGE`.

Examples

```
SEPARATE (P1,P2,P3,P4)
```

Read data from pipe `P1` and place data consecutively into pipes `P2`, `P3`, and `P4`.

```
SEPARATE ($BININ,P5,P6,P7)
```

Transfer data from the binary input com pipe to pipes `P5`, `P6`, and `P7`.

See Also

`MERGE`, `MERGEF`, `SEPARATEF`

SKIP

Define a task that alternately copies and skips data.

SKIP *(<in_pipe>, <initial_skip>, <take_cnt>, <skip_cnt>*
<out_pipe>)

Parameters

<in_pipe>
Input data pipe.
WORD PIPE | LONG PIPE

<initial_skip>
A value that specifies the initial number of values to skip.
WORD CONSTANT | LONG CONSTANT

<take_cnt>
A value that specifies the number of values to move.
WORD CONSTANT | LONG CONSTANT

<skip_cnt>
A value that specifies the number of values to skip.
WORD CONSTANT | LONG CONSTANT

<out_pipe>
Output data pipe.
WORD PIPE | LONG PIPE

Description

SKIP moves selected data from *<in_pipe>* to *<out_pipe>* and provides flexible options for skipping blocks of data. After initially doing a one time skip of *<initial_skip>* values, SKIP repeatedly moves *<take_cnt>* values to *<out_pipe>* then ignores *<skip_cnt>* values.

Examples

```
SKIP (IP0, 0, 1000, 2000, P1)
```

Transfer 1000 values to `P1`, ignore a block of 2000 values, and repeat.

```
SKIP (IP0, 100, 500, 100, P1)
```

Ignore 100 values from `IP0`, transfer 500 values to `P1`, then repeat.

```
SKIP (P1, 50, 1, 0, P2)
```

Ignore first 50 values from `P1`, then continuously transfer remaining data.