
# M300 Reference Guide

Revision date: January 25, 2017 1:58 pm

## *Miscellaneous Reference*

This book contains vital information on the M300 Data Buffer, M300 Data Format, M300 Trigger structure, and the use of RPN (post-fix) computations in the M300.

## *Acquisition Reference*

This book contains information on each type of data acquisition that is available with the M300. Each acquisition type is listed along with its allowable parameters, data size, data format, type and any miscellaneous comments applicable to that specific type.

## *Function Reference*

This book contains the non mathematical functions which may be called based on the user's needs as specified in the user's formula tables. This reference is needed to create those tables correctly.

## *Math Function Reference*

This book contains the same type of information that is in the Functions Reference with the exception that these are mathematical only and operate directly on the floating point stack.

## *Command Manager Reference*

This book provides a general overview of how the M300 Command Manager operates and associated commands that are available for use.

## *Setup Tables Reference*

This book contains the setup table information that is used to drive the acquisition processes and configure M300 projects.

Please E-mail your comments on the documentation to docs@scieng.com.

# *Table of Contents*

## *Miscellaneous Reference*



## *Acquisition Reference*

## *Function Reference*

*Math Function Reference*

## *Command Manager Reference*

*Setup Tables Reference*

# Miscellaneous Reference

There are a number of aspects to the M300 that are quite technical and require a thorough understanding before the user is able to use the M300 software to it's maximum capability. This book describes many of these technical aspects to give the user a firm understanding of these concepts and technical aspects



**M300 Miscellaneous Reference**

# Data Buffer

## Data buffers explained

Data buffers are a very integral part of the M300 system, so having a basic understanding of the data buffers and some on the inner workings is critical.

The system gathers data around using the Acquisition Manager (m300m). The data is placed in shared memory buffers. Once the Acquisition Manager has acquired all the data for a particular buffer, it marks it as being done and it triggers all the process that have registered with it (m300r, m300b, M300, etc.).

From this point forward access to the data is read only. The M300 Record Process (m300r) can read and store this buffer. The M300 Broadcast Process (m300b) can read and broadcast this buffer. The M300 process can do computations and displays on this buffer. Since access to the data is read only all of these process can get going at once doing their jobs.

Once they are done they decrement a work counter for the buffer. If there are no processes left working on the buffer, the buffer gets marked as free and the Acquisition Manager can start using this buffer again.

There are different buffer types in the system, with two main categories, synchronous and asynchronous.

The main buffer type (0) is called the synchronous buffer because the system generates one buffer per second (1 hz). All timed acquisition events get assigned to this buffer type. Of the synchronous buffer type there are several buffers in the system. The number of synchronous buffers is controlled in the buffer setup dialog (buffer table, buf.300).

The other buffers in the system are called asynchronous buffers. This is because these buffers don't get done with respect to a timed event such as the synchronous buffer. The asynchronous buffers get done when all the data is acquired. For example, in the case of a serial stream of data, an asynchronous buffer would get done when the terminating characters for a block are received. The number of asynchronous buffers of each different type is also controlled in the buffer setup dialog (buffer table, buf.300).

## Why do we need to have two different buffer categories?

It's probably easy to see that we need at least one buffer type to put the data in. We get some data and we put it in the buffer. It doesn't matter what kind of data it is, we just put it in the buffer. Then we need to decide what to do with the buffer and when?

Since we are trying to build a real-time data acquisition system, it won't be long before the distinction between timed and un-timed events is made. So we put all our timed events in one buffer (synchronous) and all our un-timed events in another buffer (asynchronous).

## Why do we need more than one asynchronous buffer type?

Again, if we put all our un-timed data into one buffer we will run into several problems for a real-time data acquisition system. Just think of the implications of having serial data and 2D data in the same buffer. When do we terminate the buffer? When the serial block is acquired or when the 2D probe is done sending the data? It's very difficult to work out a solution to such an approach. This is why we need different types of asynchronous buffers. We must keep data of the same type together in manner that makes sense for the real-time data acquisition scheme.

## Understanding buffer types.

The synchronous buffer has type 0. In the M300 system the current synchronous buffer is 1 hz. It is possible to have other synchronous buffers in the system at a frequency other than 1 hz, although this is not in use at the moment.

Asynchronous buffers have buffer types other than 0. The asynchronous buffers get their type from the data type for the master acquisition event in the asynchronous buffer. For example, in the case of 2D data, the master acquisition event is the 2D Mono Image type (5). So the asynchronous buffer type for this buffer is type 5.

## Why are buffer types important?

Let's say we want to display the 2D Mono Image data. Do we send every buffer to the display manager to have the 2D Image data displayed and then we look for the 2D Image data tag? No. We can just look for buffers of type 5 (asynchronous 2D Mono buffers) and then send these to be displayed. This cuts down on the amount of work the system needs to do to look for the correct data, not to mention other problems.

## Understanding address matching

Just as we can perform a trigger based on the buffer type, we can also perform a trigger based on the acquisition event's address. This allows use to select data from a particular board/address.

## Frequency

We use the frequency as a way to throttle the amount of data we look at. In the case of the 2D Image data, we may have 20-50 buffers per second. During real-time acquisition we don't want to display 50 buffers per second, we can display a few based on the users needs. On the other hand the data processing can look at all the 2D Image data. So different parts of the system can select the type and amount of data that they need to look at to perform the necessary task.

This all works the same way for all different modes of operation (acquisition, playback and udp). There is only one data path through the system. There is only one software application to use one set

of code and data functions to maintain. The user can customize the type of data and the amount of data that different parts of the system get by using the trigger mechanism.

# Data Format

## Description

The basic structure in the Model 300 data is composed of two sections. The first part of the Model 300 data is called the Directory area. It contains information about the data recorded. The second part of the Model 300 data is called the Data area. It contains the actual data values. Any number of these structures (Model 300 Data Buffers) can be linked together to compose an entire data file.

The directory area is composed of one or more directory structures. Each directory structure represents a data type. The number of directory entries can change and therefore so can the size of the directory area.

The data area size is not static. The data area size changes depending on the number of different data types that compose it, the number of data samples and the number of bytes per sample.

## M300 Directory Structure

The Model 300 directory entry is composed of several elements that identify the data. The directory is 16 bytes (16 * 8 bits) long. Each directory contains a Tag Number, Data Offset, Number of Bytes, Number of Samples, Bytes per Sample, Data Type, Parameter One, Parameter Two, Parameter Three and Interface Address.

In C the data structure for a data directory entry is as follows:

```
struct datadir{
    unsigned int tagNumber;
    unsigned int dataOffset;
    unsigned int numberBytes;
    unsigned int samples;
    unsigned int bytesPerSample;
    unsigned char type;
    unsigned char paramete1;
    unsigned char parameter2;
    unsigned char parameter3;
    unsigned int address;
}    typedef DataDir;
```

***Note: 'unsigned int' is a sixteen bit unsigned number and 'unsigned char' is an eight bit unsigned number.***

## Reserved Tag Numbers

| Tag Type | Tag Number |
|---|---|
| Time | 0 |
| Next | 999 |
| Reserved | 65000-65529 |
| FileName | 65530 |
| FileData | 65531 |
| Command | 65532 |
| Error | 65533 |
| Same | 65534 |
| Last | 65535 |

**Table 1: Reserved Tag Numbers**

## Directory Fields

The following is a description of all the fields in the directory entry.

**Tag Number**

Each directory entry and its associated data is given a unique tag number. This tag number is specified by the user and may be a value in the range of 0 to 65535 (0, 999, 65534 and 65535 are reserved by the system and must not be used as general purpose tag numbers). The tag number must be used when searching for a particular data item. Other data fields may be used to double check directory and data integrity.

**Data Offset**

The data offset field is used to get a pointer to the data. This value specifies the number of bytes from the beginning of the buffer where the data is located. To get a pointer to a particular data item, first find its directory entry and then add the corresponding offset to the beginning of the buffer. The data offset of the next directory points to the first directory of the next data buffer.

**Number of Bytes**

The number of bytes field contains the actual number of data bytes acquired. This field will be zero if there is no data. The number of bytes will always be less than or equal to the number of samples times the bytes per sample.

**Number of Samples**

The samples field specifies the number of desired samples. The actual number of samples may be less than this field. To get the actual number of samples, divide the total number of bytes by the bytes per sample.

**Bytes per Sample**

The bytes per sample field is indicative of the data size. This field is used to jump from one data sample to the next. By adding to the data pointer the number of bytes per sample, one can obtain a pointer to the next sample of the same data type. The number of samples should be checked before incrementing the data pointer in order to avoid getting a pointer to the wrong type of data or just the wrong data.

**Data Type**

The data type can be used to identify the data and double check it against the known type for the tag number. Different data types have different data formats and therefore it is important to check the data type as well as the number of bytes before using the data. This is particularly important when checking data integrity.

**Parameter One, Two and Three**

Any data types may use these parameters to store encoded information regarding the data at any particular moment in time. When called for these fields must be checked to get a complete set of information regarding the data. Different data types make different uses of these parameters.

**Interface Address**

The interface address is primarily used to specify hardware data source. The interface address is also used as a data synchronization pattern. For data types with no hardware data source the interface address is set to 43605 (0xAA55). Time, Next, Same and Last tags all have an interface address of 43605, used for synchronization.

## Advantages of the Data Directories

The data directory structure is a fundamental part of the Model 300 Data Acquisition System. The data directory entries control the acquisition process and document how the data was acquired. By reviewing the information in these directories when processing the acquired data, important facts are available.

**Data Acquisition Self Documentation**

Sample frequency may be determined for each tag.
Acquisition type and parameter values for each tag.
Interface Address identifies which hardware device was used.

**Data Processing, During and Post Acquisition**

Fast and easy data access for all data.
Data can be selected for one or more tags.
Data can be selected within a time frame.

Smaller data sets of interest may be created for specific uses.

**Just in case of Power/Data Loss**

Data integrity may be checked.

Data recovery is possible.

The original reason for developing the data directory structure was the desire and necessity of facilitating generalized access to the recorded data. In the past data acquisition systems generally used a fixed data format where each recorded value had a fixed place in the buffer. One problem, that frequently occurred, was that someone would change a parameter or add additional information to the buffer. This meant that project data was recorded in more than one format. The problem this causes with post processing software is obvious.

# Trigger

## Triggers explained

The Trigger concept is one of the most important in the M300 system. This is a new concept/ feature that didn't exist in the M200 system, so every user needs to read this section and get familiar with this new idea.

If you are not familiar with the data buffers structures for the M300 system, please refer to the data buffer section (See "Data Buffer"), before you go on.

In the M200 Data Acquisition System and Playback the data path through the system was very specific but different in each instance. The M200 was designed first without the necessary considerations for the Playback system. When the Playback was built it required a different software package with a different way to handle the data. The M200 had nice structures with links to the data from all different kinds of places. This made data access very easy and fast, which is what we wanted for a real-time system. When the Playback came around, we could not build such structures and links from the data. The Playback had to look through the data and find the desired tags and data offsets. This meant two software packages, two different data paths, two sets of different data functions, etc.

Although this approach worked it proved to have several disadvantages. First, since the data paths were different, it was always very difficult to ensure that the same thing as being done. Second, it was necessary to have two sets of code, which were very similar, but with different functions. This was necessary to be able to handle the differences between Playback and Acquisition modes.

Our desire, initially when the Playback was built was to have one software package that handled the data in the same way for Acquisition and Playback modes. This was not possible given all the circumstances. When it came time to do the M300 system, these ideas were already in place and therefore we had to make some choices. This is how the Trigger concept was born, out of the need to be able to have the same data path in different modes of operation using a singular software package.

## The basics about Triggers

There are two triggers, the primary (or trigger 1) and secondary (or trigger 2) triggers.

Each trigger has five main properties that can be used for matching: type, life, address, formula, and frequency.

Each trigger occupies a full line (and only one). Some of the examples and documentation syntax seem to be in two lines, but that is only because we can't fit it all on the line.

## Basic Trigger Syntax

### Traditional Trigger

```
Trigger = " type1" frequency1 board1 " type2" frequency2 board2
```

This trigger entry doesn't specify life1, formula1, life2 and formula2. The life values are 0 by default and the formula values are -1. No life, no formula checks are performed for this type of trigger.

## Full Trigger

```
Trigger = " type1[life1]" frequency1 board1 [formula1]
          " type2[life2]" frequency2 board2 [formula2]
```

This trigger entry reflects the fact that life and formula members are optional (square brackets). If the life is set to zero, no life is displayed. If the formula is -1, then no formula is saved. This simplifies the trigger entry when possible.

The board trigger can also be called address trigger. They are one and the same. The board refers to the board name and the address to the hexadecimal address for the board. Since there is a unique address for each board, they can be used interchangeably. It basically refers to an interface card where the acquisition events came from. Almost all acquisition events are attached to a particular interface card. Some events, such as data from serial ports are attached to pseudo interface address (0xF000-0xF0FF).

We will see some real trigger entries in the samples to follow, but first here is what each member means.

## Triggers.

| Name | Description |
|---|---|
| Type | Acquisition/Buffer type |
| Life | Buffer life |
| Address | Board address |
| Formula | Formula reference |
| Frequency | Frequency value |

**Triggers**

## Trigger names/types.

| Name | Type (Trigger) |
|---|---|
| “Always” | -3 |
| “Never” | -2 |
| “Ignore” | -1 |
| “Sync” | 0 |
| “2D Image” | 5 |

**Trigger Names**

| Name | Type (Trigger) |
|---|---|
| “2G Image” | 20 |
| “Serial ASCII” | 37 |
| “Serial IEEE” | 38 |
| “Serial Integer” | 39 |
| “FALCON” | 41 |
| “SDSMT HVPS Image” | 52 |
| “SPEC HVPS Image” | 53 |
| “GPS Novatel” | 54 |
| “Serial Port” | 65 |
| “2G Advanced” | 66 |
| “NRC Parallel Transfer” | 70 |
| “CIP Image” | 78 |
| “Ballard 708” | 80 |
| “Tamdar” | 81 |
| “AIMMS” | 82 |
| “Network POSAV” | 83 |
| “Network ASCII” | 84 |
| “Network Binary” | 85 |
| “CIPGS Image” | 87 |
| “Serial Binary” | 89 |
| “Network Binary Buffered” | 90 |
| “ARINC Fifo” | 91 |
| “CAS DPOL” | 92 |
| “Piraq I, Q & P” | 100 |
| “Command” | 251 |
| “Error” | 252 |
| “Sec Acq” | 254 |

**Trigger Names (Continued)**

| Name | Type (Trigger) |
|---|---|
| "Tables" | 255 |

**Trigger Names (Continued)**

## Trigger Type

| Value | Key | Description |
|---|---|---|
| -3 | "Always" | Always trigger |
| -2 | "Never" | Never trigger |
| -1 | "Ignore" | Ignore type, use other properties |
| ≥ 0 | | Trigger on *type* |

**Trigger Type**

## Trigger Life

| Value | Description |
|---|---|
| 0 | Ignore life, use other properties |
| > 0 | Trigger on *life (for sync buffers)* |

**Trigger Life**

## Trigger Addresses

| Value | Description |
|---|---|
| -1 | Ignore address, use other properties |
| ≥ 0 | Trigger on *address* |

**Trigger Address**

## Trigger Formula (Number)

| Value | Description |
|---|---|
| -1 | Ignore formula number, use other properties |
| ≥ 0 | Trigger on *formula number* |

**Trigger Formula**

Trigger Frequency

| Value | Key | Description |
|---|---|---|
| -4 | OnceOnPlay | Trigger once on play |
| -1 | Ignore | Ignore frequency, use other properties |
| 0 | Once | Trigger once only |
| ≥ 0 | | Trigger on *frequency* |

**Trigger Frequency**

## Default trigger values

The default trigger values for primary trigger are 0 for type (sync), -1 for address (none), 1 for frequency (1 hz), 0 for life (ignore), and -1 for formula (ignore).

The default trigger values for secondary trigger are -2 for type (never), -1 for address (none), -1 for frequency (ignore), 0 for life (ignore) and -1 for formula (ignore).

## Where are trigger used

Triggers are used in several different places. The most common places are the formula table, (fml.300) and the window tables (*.wnd). Triggers are also used in the ASCII table (asc.300), in the trigger commands table (tic.300) and the label table (lbl.300) (See “Setup Tables Reference” )

Trigger entries used in the formula table control how often and when a block of formulas gets executed.

Trigger entries in the window tables have the same effect but they control how often the window display is performed. They basically drive the window pump.

## Why do we need all these members for the trigger and what do they do for me

Type

The trigger type allows us to trigger on a specific data buffer. We can pick a synchronous buffer or a 2D Mono buffer just by using the trigger type. This means that we can ignore all other data buffers and narrow down on a particular buffer type.

But there are times when we don’t want to do the type checking on the trigger. So that’s why we have the special option that ignores the trigger type checking (Ignore, -1).

At other times one trigger is enough. So we need to have an easy way to skip the other trigger (Never, -2).

Life

The buffer life became necessary when we introduced different synchronous buffers to the M300 system (so it applies to sync buffers only). Originally we only had the 1 Hz Sync buffer, so there was no need to pick amongst the different synchronous buffers. But now that we can have buffers faster and slower than 1 Hz, we need the buffer life to select which synchronous buffer we want to look at.

For asynchronous buffers the buffer life is zero and the trigger life is not used under those circumstances. Asynchronous buffer's have their life based on the master event and not on a fixed frequency.

So, if we setup a project with the traditional 1 Hz sync buffer and a faster 20 Hz buffer, we now have a mechanism in place to be able to select between these two types of buffers. In order to perform the trigger life check you must specify a buffer type (Sync, 0). If you don't do the trigger type check, then you can't do the buffer life check.

The buffer life value follows the type variable without a space. In this case say we have a system frequency of 20Hz. Then the buffer life for the 1 Hz sync buffer is 20 and the buffer life for the fast 20 Hz buffer is 1. So to look at the 1 Hz sync buffer you must use the "Sync" or "Sync20" trigger type/life value. In order to look at the faster sync buffer we would specify "Sync1".

The trigger life can be set to zero to disable the trigger life check (use "Sync" or "Sync0").

## Address

The address or board is another variable that allows us to narrow down on a particular data item. Say we have a system with 2DC and 2DP probes. If we use the trigger type we can get all the 2D Mono data by picking "2D Image", type 5. But if we want to narrow down on just 2DC or 2DP we need to use something else to pick our final desired data item. The address allows us to do just that. By using the correct trigger address we can ensure that we have a trigger that looks only at 2DC or 2DP data.

There are situations were we might want to specify the address as a hexadecimal value or just change the board address for the desired board. We can use the M300 for example to look at data from the M200 system. Even though the M200 data format has some features missing, it is still possible to get the desired data using the trigger mechanism. If you the RawView utility it will help you identify what to look for in the data to make the correct trigger.

There is a special value for address (-1) which will disable the address check feature for the trigger.

## Formula

The need for the formula trigger is less obvious, but the formula trigger is one of the most advanced features of the trigger mechanism. Until now we have been able to trigger on a particular data type and this works great for a large number of cases. But if you have a situation where you need to separate or validate the incoming data before using it for display purposes, the formula trigger is just the thing

For example, take the case of dropsonde data. All the data comes in the same serial port (data channel). We can setup a basic data trigger to get this data. Once we have the data the situation is a bit different. We need to be able to determine if the data is valid and separate the data into different drops. We can do this using several formula computations and generate a trigger formula value for different drops. These formulas can in turn be used in a List or Skew-T displays to trigger the displays only when we have the correct valid data points.

The formula used for the trigger must be of integer type. Don't use formulas with floating point or string types. If the formula value is a zero, the trigger is not executed. If the formula value is non-zero, then the trigger might execute, based on other properties as well.

There is a special value for the formula number trigger (-1). This can be used to skip the formula value check for the trigger.

Frequency

The frequency is the final item that determines when a trigger gets executed or not. When we look at data, there are times when we want to handle all the data. For example, take the case of computations on 2D Mono data. On the other side of the coin there are occasions where we only want to look at a few data buffers. For example, the 2D Mono Image display might only want to display one or two buffer per second. The frequency gives the capability to select how many of the actual data buffers generate a trigger.

If you set the trigger frequency equal to the system frequency then you ensure you will get the maximum number of buffers possible for the trigger.

The frequency has three special cases. Ignore (-1), Once (0) and OnceOnPlay (-4).

The first is we can set the frequency to Ignore, (-1). This will cause the trigger to skip the frequency check.

The second special case for frequency is Once, (0). This causes the trigger to execute only once (Once). This is a special type of trigger that can be used to initialize certain formula values. There are two benefits from doing this. The first is that we can save CPU time by not executing a sequence of formulas all the time, if the value doesn't change. The second is that there are times when we really only want to initialize a formula and make sure it doesn't run again.

The third case for frequency is OnceOnPlay, (-4). This is similar to the Once trigger and the trigger executes only once each time the play is hit.

## M300 Sync Buffer Information

### Sync 1 Hz buffer

The Sync 1 Hz buffer has a life equal to the system frequency from the system board table. So if the system frequency is set for 200 hz, then the life for the Sync 1 hz buffer is 200.

This would be the standard Sync 1 Hz trigger (once per second).

Trigger = "Sync" 1 None "Never" Ignore None

If you want a 0.5 hz (once every two seconds) trigger off the sync buffer, you can use.

Trigger = "Sync" 0.5 None "Never" Ignore None

If you want a faster trigger than 1 hz, you can't use the Sync 1 Hz buffer. The following is example will only trigger at 1 hz, even though the trigger is asking for 5 hz. The reason being that the Sync 1 hz buffer occurs only once per second, so that is the maximum possible trigger off the Sync 1 Hz buffer.

Trigger = "Sync" 5 None "Never" Ignore None

### Sync buffer faster than 1 hz

### Sync 10 Hz buffer (10 times per second)

Say we want a buffer that is faster than the 1 hz Sync buffer. Let's say we want a 10 hz buffer. We will assume a system frequency of 200 hz. To figure out the buffer life we take the system frequency and divide it by the desired buffer frequency.

200 hz % 10 hz = 20 life (for 10 hz Sync buffer)

So now if we want to trigger using the 10 hz Sync buffer at 10 hz, we would use the following trigger.

Trigger = "Sync20" 10 None "Never" Ignore None

The "Sync" parameter now needs the buffer life of 20 so it can select the correct buffer for the trigger.

This setup would have a Sync 1 Hz buffer (Life 200) and Sync 10 Hz buffer (Life 20).

With the Sync 10 Hz buffer you can trigger up to 10 hz but not more.

### Sync buffer slower than 1 hz

Sync 0.0666666667 Hz buffer (once every 15 seconds)

Let's address the case for a slow sync buffer. We will still use 200 hz for system frequency. Suppose we want a buffer to be done once every 15 seconds. So this gives us a 0.066666667 hz Sync buffer. Again, to figure out the buffer life we take the system frequency and divide it by the desired buffer frequency.

200 hz % 0.0666666667 = 3000 life (for the 0.066666667 hz Sync buffer)

So now if we want to trigger using the 0.066666667 hz Sync buffer, we would use the following trigger.

Trigger = "Sync3000" 1 None "Never" Ignore None

## A few trigger examples

Next we will be explaining how the Trigger affects computations in the formula manager with some examples.

```
;
; Initialization trigger based on sync buffer
;
Trigger = "Sync" Once None "Never" Ignore None
"DegToRad" "" F5 F[1] 0.01745329252
"RadToDeg" "" F6 F[1] 57.29577951
```

This is a basic trigger used to setup formulas that should be initialized once. The primary trigger uses the ‘Sync’ buffer, ‘Once’. The address is ‘None’, which will cause the address to be ignored. The secondary trigger is not used since we pick ‘Never’ for the trigger type. We don’t even have to look any further at the trigger frequency nor address.

```
;
; 1 Hz sync trigger
;
Trigger = "Sync" 1 None "Never" Ignore None
"Time" "" F0 S[10] Time(A0)
"Date" "" F1 S[10] Date(A0)
```

This is the basic 1 hertz trigger on the 'Sync' buffer. The primary address is not a factor nor is the secondary trigger. This trigger assumes a project with only one sync buffer (1Hz). No buffer life is used.

```
;
; Trigger on piraq data for PiraqA board, 1 Hz maximum
;
Trigger = "Piraq I, Q & P" 1 PiraqA "Never" Ignore None
"Timing Mode" "" F2100 l[1] PqConfig(A2001, 0)
"Delay"       "" F2101 l[1] PqConfig(A2001, 1)
"Gates"       "" F2102 l[1] PqConfig(A2001, 2)
```

This trigger is a bit more interesting. The trigger type is on the "Piraq I, Q & P" data. We have a 1 Hz frequency. The address/board is PiraqA. So this trigger will fire at most once per second on "Piraq I, Q & P" data for the PiraqA board only. The secondary trigger is not a factor.

Here is another example which shows how to handle 2DC data in the M300 system.

```
;
; 1 Hz sync trigger
;
Trigger = "Sync" 1 None "Never" Ignore None
"2DC Sizes" "" F1000 F[32] PrData(Pr.2dc, 2)
```

The primary trigger is set for 'Sync' buffer, 1 Hz and ignore address. The secondary trigger is not a factor. This basically gets the 2DC Sizes once per second and updates for F1000.

```
;
; Trigger 1 setup for 1 Hz sync trigger,trigger 2 set for 10 Hz 2D Mono Data
;
Trigger = "Sync" 1 None "2D Image" 10 2dc
"2DC Counts" "" F1001 F[32] MoSums(Pr.2dc, Aq.2DCImage, 0x00, 1)
```

Here the trigger needed to be changed to handle the 2DC data into the MoSums function. The MoSums function needs a 1 Hz 'Sync' trigger to produce a result once per second. It also needs the secondary trigger to fire for the '2D Image' for the '2dc' board. The secondary trigger in this case sets a maximum of 10 Hz, in other words a maximum of 10 buffers per second will be analyzed by the MoSums function.

```
;
; Trigger 1 Hz on 2D Mono data for 2dc board
;
Trigger = "2D Image" 1 2dc "Never" Ignore None
```

```
"2DC Tas Factors"         ""    F1002 [1] Aq.2DCTasFactors
"2DC Elapsed Time"        ""    F1003 l[1] Aq.2DCElapsedTime
"2DC Elapsed Tas"         ""    F1004 l[1] Aq.2DCElapsedTas
"2DC Elapsed Shadow Or"   ""    F1005 i[1] Aq.2DCElapsedShadowOr
"2DC Tas Mul Fac"         ""    F1100 i[1] F1002
"2DC Tas Div Fac"         ""    F1101 i[1] F1002 16 >>
"2DC Tas Clock In"        "MHZ" F1102 F[1] PrTasClockIn(Aq.2DCTasFactors)
"2DC Elapsed Time"        "s"   F1103 F[1] F1003 40000 /
"2DC Elapsed Tas"         "s"   F1104 F[1] F1004 100 * F1102 / 1.0e-6 *
```

This block of formulas gets data from the 2D buffer for the slave events. In this case we only need to set the primary trigger to fire at 1 Hz for the '2D Image' data type for the '2dc' board. The secondary trigger is not necessary.

```
;
; Low speed, 1Hz trigger
;
Trigger = "Sync20" 1 None "Never" Ignore None
;
; Heading
;
"Heading" "deg" F2000 F[20] A2000 0.00549316 *
"Heading" "deg" F2001 F[20] F2000 360 +
"Heading" "deg" F2000 F[20] Lt(F2000, 0, F2001, F2000)
"Heading" "rad" F2002 F[20] Units(F2000, "rad", "deg")
```

This is an example of a 1 Hz trigger for a sync buffer with a trigger life of 20. No secondary trigger.

```
;
;
; High speed, 20Hz trigger
;
Trigger = "Sync1" 20 None "Never" Ignore None
"Time"           "" F0    S[12] Time(A0)
"Date"           "" F1    S[14] Date(A0)
;
"GPS/SysTimeSel" "" F23   I[1]  F23 1 +
"GPS/SysTimeSel" "" F23   I[1]  Limit(F23, 0, 100)
;
"XmitParamTime"  "" F5405 S[9]  Le(F23, 50,  F5105, F0)
"XmitParamDate"  "" F5406 S[14] Le(F23, 50,  F5106, F1)
```

The example shows a 20 Hz trigger for the fast sync buffer with a trigger life of 1. No secondary trigger.

```
;
; skewt data for drop 1
;
Trigger = "Serial ASCII" 100 blueheatserial1 F5401 "Never" Ignore None
;
"AirPress"   "mb"    F8105 F[1] Ge(F6105, 9998, F8105, F6105)
```

```
"AirTemp"    "°c"    F8106 F[1] Ge(F6106, 98, F8106, F6106)
"RH"         "%"     F8107 F[1] Ge(F6107, 998, F8107, F6107)
"WindDir"    "deg"   F8108 F[1] Ge(F6108, 998, F8108, F6108)
"WindSpeed"  "m/s"   F8109 F[1] Ge(F6109, 998, F8109, F6109)
"DewPoint"   "°c"    F8150 F[1] RHToDewPoint(F8107, F8106)
"WindDir"    "rad"   F8151 F[1] Units(F8108, "rad", "deg")
"WindSpeed"  "knots" F8152 F[1] Units(F8109, "knots", "m/s")
```

No secondary trigger. The primary trigger is based on the “Serial ASCII” data type with a maximum frequency of 100 Hz for the “blueheatserial1” board. The interesting part is that this trigger uses F5401 as the trigger formula.

Finally, a trigger entry from the header of the drop1lst.wnd file. This trigger shows a “Serial ASCII” data type trigger at a maximum frequency of 100 Hz for the “blueheatserial1” card with F5311 as the formula trigger.

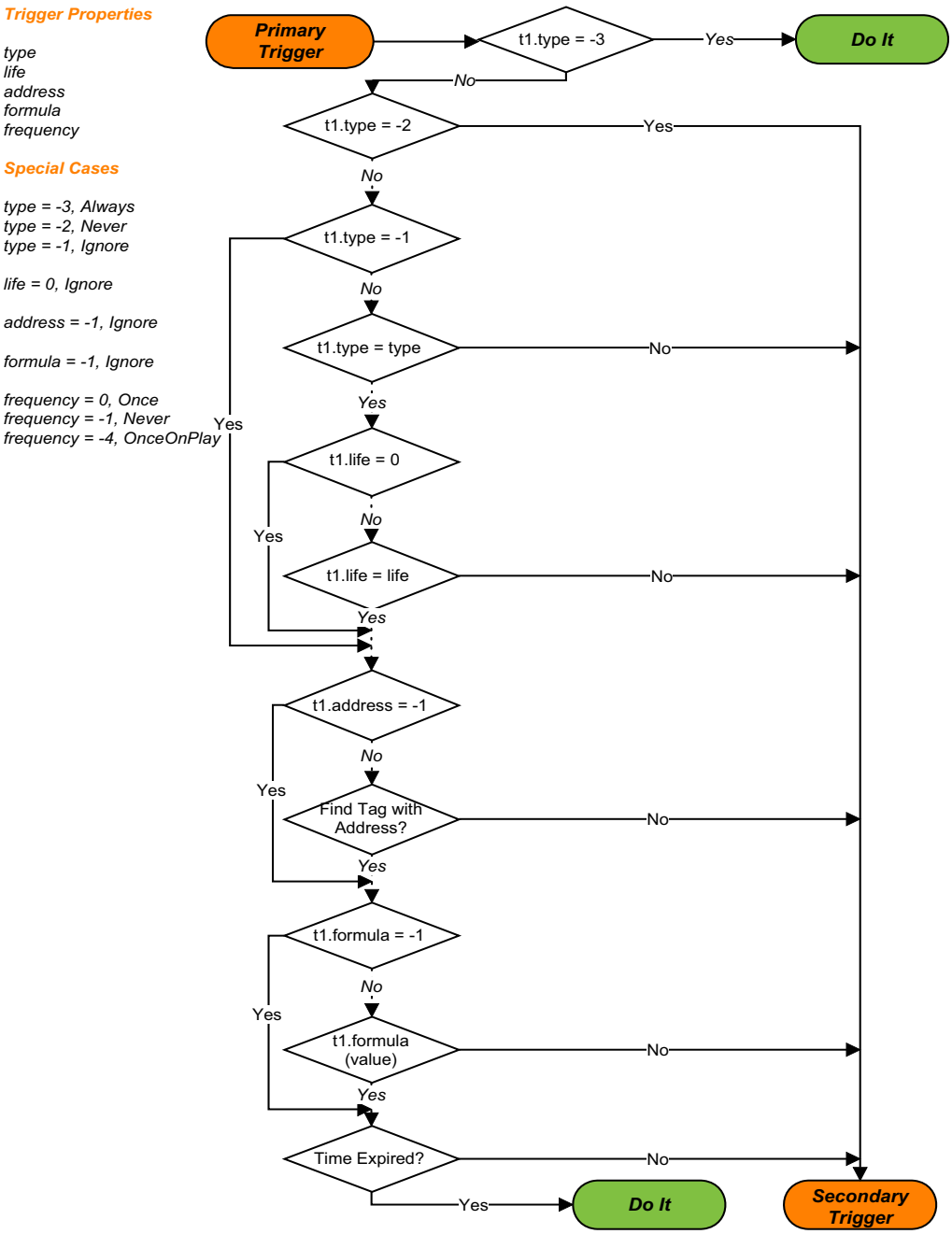
```
; drop1lst.wnd
Trigger = "Serial ASCII" 100 blueheatserial1  F5311 "Never" Ignore None
Area = 0 0 625 475
```

## Trigger operation

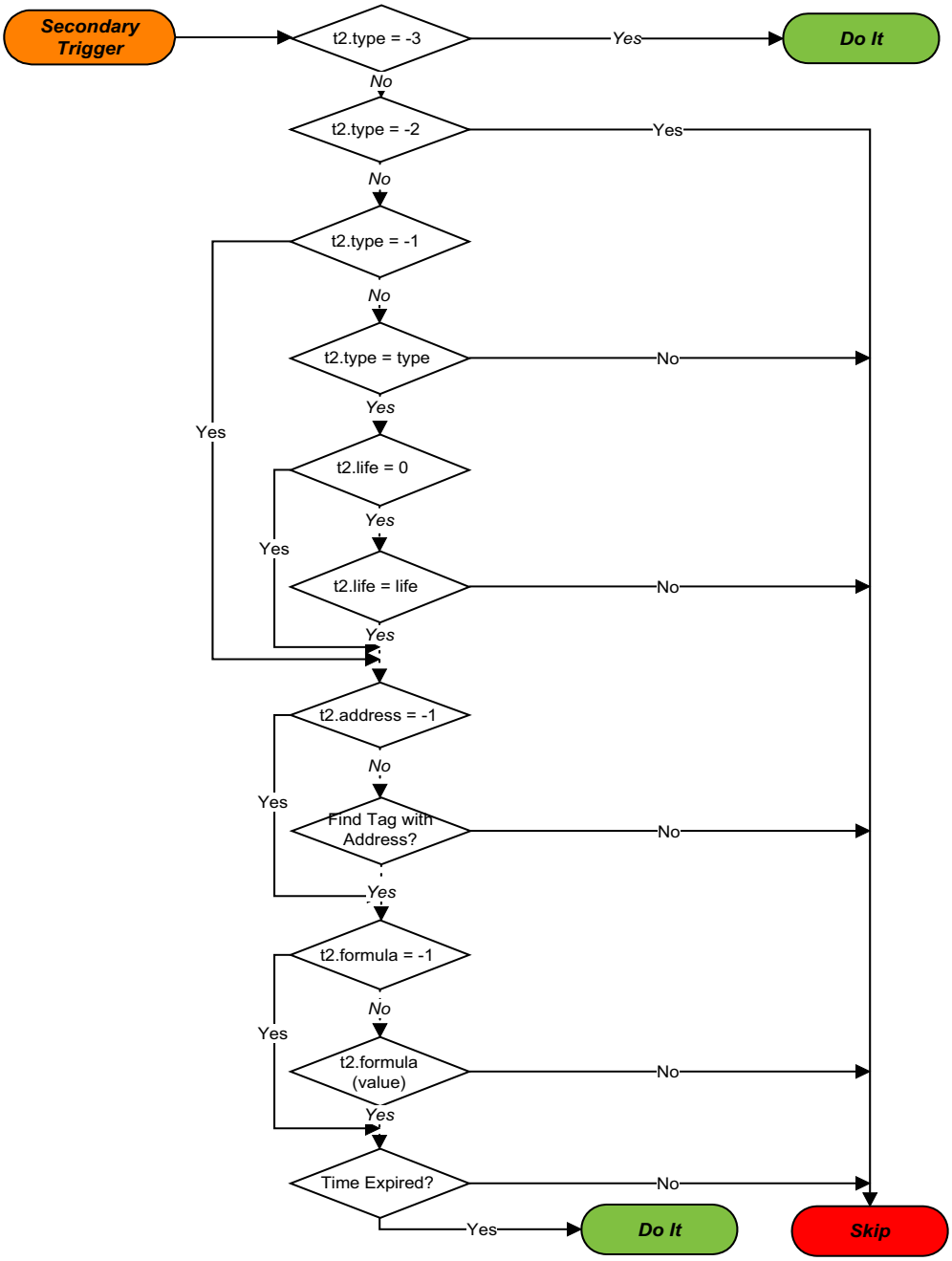
If the primary trigger doesn't fire, then the secondary trigger is checked. It the primary trigger fires, then the secondary trigger is ignored. So the system first checks for the primary trigger and then it moves to the secondary trigger if necessary.

1. If the trigger type is set to never (-2), we are done.
2. If the trigger type is set to ignore (-1), skip type check.
3. If the trigger type doesn't match the buffer type, we are done.
4. Do trigger on buffer life, if necessary.
5. If the address is set to ignore (-1), we skip address checking.
6. If the address doesn't match the ‘buffer address', we are done.
7. If the formula is set to ignore (-1), we skip formula checking.
8. If formula value is false (zero), we are done.
9. If frequency set to ignore (-1), we skip frequency checking.
10. If the time has expired, we fire the trigger.
11. Otherwise we will not fire the trigger.

## Trigger Operation Flowchart




**M300 Figure 1: Primary Trigger Operation Flowchart**




**M300 Figure 2: Secondary Trigger Operation Flowchart**

# Reverse Polish Notation

## What is RPN?

Reverse polish notation (RPN), also known as Postfix, is a system of mathematical notation that eliminates the need for brackets to identify evaluation order. This method of writing arithmetic expressions is particularly suited to computerized methods of evaluation because the computer can evaluate an equation from left to right without waiting for operators. RPN accomplishes this because it uses a stack-based method for performing operations on data. This stack based system is further explained in the next section.

## How does RPN work?.

In RPN, the numbers and operators are listed one after another, and an operator always acts on the most recent numbers in the list. The numbers can be thought of as forming a stack, like a pile of plates. The most recent number goes on the top of the stack. An operator takes the appropriate number of arguments from the top of the stack and replaces them by the result of the operation.

A simple expression such as [(**3 + 5**) **x** (7 - **2**)] would be written as [**3 5 + 7 2 - x**]. The computer evaluates the expression from left to right as follows.

1. Push 3 onto the stack.
2. Push 5 onto the stack. The stack now contains (3, 5).
3. Apply the operation: take the top two numbers off the stack, add them together, and put the result back on the stack. The stack now contains just the number 8.
4. Push 7 onto the stack.
5. Push 2 onto the stack. It now contains (8, 7, 2).
6. Apply the operation: take the top two numbers off the stack, subtract the top one from the one below, and put the result back on the stack. The stack now contains (8, 5).
7. Apply the operation: take the top two numbers off the stack, multiply them together, and put the result back on the stack. The stack now contains just the final answer, the number 40.

## Why does M300 use RPN?

The left-to-right precedence of RPN is very compatible with the sequential nature of operations in a computing device. LIFO (Last-In-First-Out) algorithms and sequential stacks are common place in compilers and computer languages. It is because of this connection RPN is a very efficient way for chip logic to perform calculations. In a short period of time a technical user can familiarize themselves with the structure of RPN and be able to perform more complicated calculations with fewer keystrokes and represent those calculations in a more straight forward manner.

# Color System

## The basics

The M300 color system is a system wide color recognition system. The user can enter a color name anywhere color values are used, and the M300 will recognize it and use the appropriate numeric color value internally. This also works in the other direction. That is, if a user puts a numeric color in any M300 table, the M300 will attempt to match it to the appropriate color name, and if a match is found, it will replace the numeric value with the color name for convenience purposes. This feature is also available when using the Command Manager prompt.

## M300 Colors

The table below shows the names of the colors that the M300 recognizes, as well as their hexadecimal numeric values.

| RGB (Hex) | Name | RGB (Hex) | Name | RGB (Hex) | Name |
|---|---|---|---|---|---|
| F0F8FF | aliceblue | FFFAF0 | floralwhite | 48D1CC | mediumturquoise |
| FAEBD7 | antiquewhite | 228B22 | forestgreen | C71585 | mediumvioletred |
| 7FFFD4 | aquamarine | DCDCDC | gainsboro | 191970 | midnightblue |
| F0FFFF | azure | F8F8FF | ghostwhite | F5FFFA | mintcream |
| F5F5DC | beige | FFD700 | gold | FFE4E1 | mistyrose |
| FFE4C4 | bisque | DAA520 | goldenrod | FFE4B5 | moccasin |
| 000000 | black | A9A9A9 | gray | BA55D3 | morchid |
| FFEBCD | blanchedalmond | 00A000 | green | 9370DB | mpurple |
| 0000FF | blue | ADFF2F | greenyellow | 3CB371 | mseagreen |
| 8A2BE2 | blueviolet | F0FFF0 | honeydew | 7B68EE | mslateblue |
| A52A2A | brown | FF69B4 | hotpink | 00FA9A | mspringgreen |
| DEB887 | burlywood | CD5C5C | indianred | 48D1CC | mturquoise |
| 5F9EA0 | cadetblue | 4B0082 | indigo | C71585 | mvioletred |
| 7FFF00 | chartreuse | FFFFF0 | ivory | FFDEAD | navajowhite |
| D2691E | chocolate | F0E68C | khaki | 000080 | navy |

**M300 Colors**

| RGB (Hex) | Name | RGB (Hex) | Name | RGB (Hex) | Name |
|---|---|---|---|---|---|
| FF7F50 | coral | E6E6FA | lavender | FDF5E6 | oldlace |
| 6495ED | cornflowerblue | FFF0F5 | lavenderblush | 808000 | olive |
| FFF8DC | cornsilk | 7CFC00 | lawngreen | 6B8E23 | olivedrab |
| DC143C | crimson | ADD8E6 | lblue | FFA500 | orange |
| 00FFFF | cyan | F08080 | lcoral | FF4500 | orangered |
| 00008B | darkblue | E0FFFF | lcyan | DA70D6 | orchid |
| 008B8B | darkcyan | FFFACD | lemonchiffon | EEE8AA | palegoldenrod |
| B8860B | darkgoldenrod | FAFAD2 | lgoldenrodyellow | 98FB98 | palegreen |
| 808080 | darkgray | D3D3D3 | lgray | AFEEEE | paleturquoise |
| 006400 | darkgreen | 90EE90 | lgreen | DB7093 | palevioletred |
| BDB76B | darkkhaki | ADD8E6 | lightblue | FFEFD5 | papayawhip |
| 8B008B | darkmagenta | F08080 | lightcoral | FFDAB9 | peachpuff |
| 556B2F | darkolivegreen | E0FFFF | lightcyan | CD853F | peru |
| FF8C00 | darkorange | FAFAD2 | lightgoldenrodyellow | FFC0CB | pink |
| 9932CC | darkorchid | D3D3D3 | lightgray | DDA0AA | plum |
| 8B0000 | darkred | 90EE90 | lightgreen | B0E0E6 | powderblue |
| E9967A | darksalmon | FFB6C1 | lightpink | 800080 | purple |
| 8FBC8F | darkseagreen | FFA07A | lightsalmon | FF0000 | red |
| 483D8B | darkslateblue | 20B2AA | lightseagreen | BC8F8F | rosybrown |
| 2F4F4F | darkslategray | 87CEFA | lightskyblue | 4169E1 | royalblue |
| 00CED1 | darkturquoise | 778899 | lightslategray | 8B4513 | saddlebrown |
| 9400D3 | darkviolet | B0C4DE | lightsteelblue | FA8072 | salmon |
| 00008B | dblue | FFFFE0 | lightyellow | F4A460 | sandybrown |
| 008B8B | dcyan | 00FF00 | lime | 2E8B57 | seagreen |
| FF1493 | deeppink | 32CD32 | limegreen | FFF5EE | seashell |
| 00BFFF | deepskyblue | FAF0E6 | linen | A0522D | sienna |
| B8860B | dgoldenrod | FFB6C1 | lpink | C0C0C0 | silver |

**M300 Colors (Continued)**

| RGB (Hex) | Name | RGB (Hex) | Name | RGB (Hex) | Name |
|---|---|---|---|---|---|
| 808080 | dgray | FFA07A | lsalmon | 87CEEB | skyblue |
| 006400 | dgreen | 20B2AA | lseagreen | 6A5ACD | slateblue |
| 696969 | dimgray | 87CEFA | lskyblue | 708090 | slategray |
| BDB76B | dkhaki | 778899 | lslategray | FFFAFA | snow |
| 8B008B | dmagenta | B0C4DE | lsteelblue | 00FF7F | springgreen |
| 1E90FF | dodgerblue | FFFFE0 | lyellow | 4682B4 | steelblue |
| 556B2F | dolivegreen | FF00FF | magenta | D2B48C | tan |
| FF8C00 | dorange | 66CDAA | maquamarine | 008080 | teal |
| 9932CC | dorchid | 800000 | maroon | D8BFD8 | thistle |
| 8B0000 | dred | 0000CD | mblue | FF6347 | tomato |
| E9967A | dsalmon | 66CDAA | mediumaquamarine | 40E0D0 | turquoise |
| 8FBC8F | dseagreen | 0000CD | mediumblue | EE82EE | violet |
| 483D8B | dslateblue | BA55D3 | mediumorchid | F5DEB3 | wheat |
| 2F4F4F | dslategray | 9370DB | mediumpurple | FFFFFF | white |
| 00CED1 | dturquoise | 3CB371 | mediumseagreen | F5F5F5 | whitesmoke |
| 9400D3 | dviolet | 7B68EE | mediumslateblue | FFFF00 | yellow |
| B22222 | firebrick | 00FA9A | mediumspringgreen | 9ACD32 | yellowgreen |

**M300 Colors (Continued)**

# Font System

## The basics

The M300 system actually used the same fonts provided by the GUI. It is recommend that you check the Font documentation using the online documentation for Photon (use helpviewer).

There is also a GUI configuration utility called 'fontcfg'. To run this utility to check or change setting just type 'fontcfg &' at a PtTerm window (command shell window).

To list the basic font names you can use the 'ls' command and look at the '/qnx4/photon/font' directory.

If necessary, you can use the Text window to select the desired font for a text label. Then check the 'txt.300' file and the appropriate label entry for the font name.

We recommend to use the Courier font, since it is a fixed font. True type fonts don't display as nice and they are a lot more expensive to render.

The basic naming convention is to use the Font name (or abbreviation) followed by font size and then any style modifier.

For example, for Courier you could use 'cour20b' or 'cour12'. The first would give you Courier, 20 point size and bold. The second would provide a basic 12 point size Courier font.

# Acquisition Reference

The following is a list of the acquisition type routines (different instruments) presently available/ supported.

| Type/Description | Parameter1 | Parameter2 | Parameter3 | Page |
|---|---|---|---|---|
| Type 0 (Date/Time, Reserved) | | | | 41 |
| Type 1 (CAMAC Analog E205/E210) | slot | channel | gain | 43 |
| Type 2 (CAMAC 1D Counts) | slot | command | interface | 44 |
| Type 3 (CAMAC Digital Events E100) | slot | port | | 46 |
| Type 4 (CAMAC Loran C/GPS) | slot | command | type | 47 |
| Type 5 (2D Mono Image) | interface | factors | rearm | 48 |
| Type 6 (2D Mono TAS Factors) | interface | | | 50 |
| Type 7 (2D Mono Elapsed Time) | interface | | | 51 |
| Type 8 (2D Mono Elapsed TAS/100) | interface | | | 52 |
| Type 9 (2D Mono Elapsed Shadow OR) | interface | | | 53 |
| Type 10 (2D Mono Total Shadow OR) | interface | | | 54 |
| Type 11 (2D Mono House Data) | interface | | type | 56 |
| Type 12 (DT2801 Analog) | channel | gain | | 57 |
| Type 13 (DT2801 Digital Events) | port | | | 58 |
| Type 14 (Loran C/GPS) | command | control | type | 59 |
| Type 15 (Encoding Altimeter) | | | | 62 |
| Type 16 (INS Arinc Serial) | label | port | | 63 |
| Type 17 (INS Synchro) | synchro | | | 65 |
| Type 18 (CAMAC INS ARINC Serial) | slot | label | | 66 |
| Type 19 (CAMAC INS Synchro) | slot | synchro | | 67 |
| Type 20 (2D Grey Image) | interface | factors | rearm | 68 |
| Type 21 (2D Grey TAS Factors) | interface | | | 71 |
| Type 22 (2D Grey Elapsed Time) | interface | | | 72 |
| Type 23 (2D Grey Elapsed TAS/256) | interface | | | 73 |

**Acquisition Reference**

| Type/Description | Parameter1 | Parameter2 | Parameter3 | Page |
|---|---|---|---|---|
| Type 24 (2D Grey Minimum Count) | interface | | | 74 |
| Type 25 (2D Grey Middle Count) | interface | | | 75 |
| Type 26 (2D Grey Maximum Count) | interface | | | 76 |
| Type 27 (2D Grey OR Slice) | interface | dma | | 77 |
| Type 28 (2D Grey Shadow Slice Count) | interface | | | 78 |
| Type 29 (2D Grey Probe Byte) | interface | command | | 79 |
| Type 30 (1D Counts) | interface | command | | 80 |
| Type 31 (Hail Spectrometer) | counters | counters | | 82 |
| Type 32 (Hail Events) | | | | 84 |
| Type 33 (Analog STB-TC Analog) | channel | mode | gain | 86 |
| Type 34 (Digital Input) | port | | | 88 |
| Type 35 (SEA Analog to Digital Input) | id | channel | gain | 89 |
| Type 36 (SEA 24 Counter) | | | | 91 |
| Type 37 (Serial ASCII Data) | block | | throttle | 92 |
| Type 38 (Serial IEEE Data) | swap | | throttle | 94 |
| Type 39 (Serial Integer Data) | data type | | throttle | 96 |
| Type 40 (Sonic Wind System) | trigger | control | port | 98 |
| Type 41 (Falcon Data) | mode | divider | throttle | 101 |
| Type 42 (INS Accelerometer) | counter | polarity | reset | 103 |
| Type 43 (1D256 Counts) | counts | interface | factors | 104 |
| Type 44 (1D256 Analog Input) | channel | range | gain | 106 |
| Type 45 (CAMAC VOR Data) | slot | channel | | 108 |
| Type 46 (1D256 Spare 0) | mode low | mode high | | 109 |
| Type 47 (1D256 Spare 1) | mode low | mode high | | 110 |
| Type 48 (1D256 House Data) | | interface | | 111 |
| Type 49 (1D256 Activity) | | | | 112 |
| Type 50 (1D256 Total Strobes) | | | | 113 |

**Acquisition Reference (Continued)**

| Type/Description | Parameter1 | Parameter2 | Parameter3 | Page |
|---|---|---|---|---|
| Type 51 (1D256 Total Counts) | | | | 114 |
| Type 52 (SDSMT HVPS Image Data) | dma | channel | throttle | 115 |
| Type 53 (SPEC HVPS Image Data) | interface | factors | rearm | 116 |
| Type 54 (Novatel GPS) | | | rearm | 118 |
| Type 55 (VAX Clock) | | | | 119 |
| Type 56 (CAMAC 1D256 Counts) | counts | interface | slot | 120 |
| Type 57 (CAMAC 1D256 Reference Voltage) | | | slot | 122 |
| Type 58 (CAMAC 1D256 Spare 0) | mode low | mode high | slot | 123 |
| Type 59 (CAMAC 1D256 Spare 1) | mode low | mode high | slot | 124 |
| Type 60 (CAMAC 1D256 House Data) | | interface | slot | 125 |
| Type 61 (CAMAC 1D256 Activity) | | | slot | 126 |
| Type 62 (CAMAC 1D256 Total Strobes) | | | slot | 127 |
| Type 63 (CAMAC 1D256 Total Counts) | | | slot | 128 |
| Type 64 (1D256 Ballard Counts) | counts | interface | factors | 129 |
| Type 65 (Serial Port DC 8 DADS Data) | identifier | terminate | throttle | 131 |
| Type 66 (2D Grey Advanced) | interface | dma | rearm | 133 |
| Type 67 (PMS 1058B 1D Data) | interface | PCW | channels | 137 |
| Type 68 (9513 Counters) | counter | | | 139 |
| Type 69 (BC620AT Time) | | | | 140 |
| Type 70 (DRV11 Data) | data type | dma | rearm | 141 |
| Type 71 (Pressure Multiplexer) | id | channel | gain | 142 |
| Type 72 (INS INI Synchro) | synchro | | | 144 |
| Type 73 (INS INI Serial) | label | card | | 145 |
| Type 74 (INS INI Flags) | | | | 146 |
| Type 75 (SPP/CDP Data) | interface | command | | 147 |
| Type 76 (CAS Serial Data) | interface | | control | 149 |
| Type 77 (CIP Serial Data) | interface | | | 151 |

**Acquisition Reference (Continued)**

| Type/Description | Parameter1 | Parameter2 | Parameter3 | Page |
|---|---|---|---|---|
| Type 78 (CIP Image Data) | | DMA | rearm | 152 |
| Type 79 (CAS PBP Data) | | | | 153 |
| Type 80 (Ballard 708 Data) | | | | 154 |
| Type 81 (Serial Port Tamdar Data) | stx | etx | type | 155 |
| Type 82 (Serial Port AIMMS Data) | type | samples | id | 156 |
| Type 83 (Network POSAV Data) | | | | 158 |
| Type 84 (Network ASCII Data) | block | | | 159 |
| Type 85 (Network Binary Data) | match | | | 160 |
| Type 86 (CIPGS Data) | | | | 161 |
| Type 87 (CIPGS Image Data) | | | | 162 |
| Type 88 (CIPGS Info Data) | | | | 163 |
| Type 89 (Serial Binary) | match | | | 164 |
| Type 90 (Network Binary Buffered Data) | expire | | | 165 |
| Type 91 (ARINC Fifo Data) | start | mode | end | 166 |
| Type 92 (CAS DPOL Data) | | | | 167 |
| Type 100 (PIRAQ I, Q and P) | | | | 169 |
| Type 101 (PIRAQ Config) | | | | 171 |
| Type 102 (PIRAQ Status) | | | | 173 |
| Type 250 (Status Info Data) | type | sub type | | 175 |
| Type 251 (Command Data) | | | | 177 |
| Type 252 (Error Data) | | | | 178 |
| Type 253 (Telemetric Data) | type | | | 179 |
| Type 254 (Secondary Acquisition) | type | | | 180 |
| Type 255 (Tables Data) | | | | 182 |

**Acquisition Reference (Continued)**

# Type 0 (Date/Time, Reserved)

## Description

This is a reserved type. It automatically puts the timestamp data for tag 0.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires eighteen bytes of data per sample. M300 buffers have both start and stop time for all buffers and therefor each buffer has two samples. This is different from the M200 where there was only a single sample. The time in the M200 was either at the start of the buffer or at the end depending on the buffer type.

## Data Format

The data acquired is in the following format:

| Byte Offset | Value |
|---|---|
| 0-1 | Year |
| 2-3 | Month |
| 4-5 | Day |
| 6-7 | Hour |
| 8-9 | Minute |
| 10-11 | Second |
| 12-13 | Fraction of a second |
| 14-15 | Maximum System Frequency |

**Data format**

| Byte Offset | Value |
|---|---|
| 16-17 | Buffer Life Span |

**Data format (Continued)**

## Comments

This acquisition type is necessary to date/time stamp the data buffer. The last three elements of this data defines the time frame of data buffers.

The system frequency value defines the number of ticks that occur during a second. This is therefore the maximum acquisition frequency. The buffer life span value defines how many ticks a buffer exists for. The fraction of second value defines on what tick of a second a buffer started.

You must not specify this acquisition event in the acquisition table, since it is automatically inserted in the data stream. Care should be taken, in order not to use tag 0.

Time data will always be represented by the reserved tag 0.

# Type 1 (CAMAC Analog E205/E210)

## Description

This acquisition type is used to acquire analog data from a DSP E205 analog to digital converter interface card mounted in a CAMAC crate. The E205 can support up to 256 analog channels using 16 E210 multiplexers.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | CAMAC Slot | 1-23 |
| 2 | E205 Channel | 0-255 |
| 3 | E205 Gain | 0, 1, 2 |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes of data should be allocated for each sample.

## Data Format

The data acquired is in the following format:

| | | |
|---|---|---|
| 0x7FFF | 32767 | +full scale |
| 0x0000 | 0 | zero |
| 0x8000 | -32768 | -full scale |

**Data Format**

## Type

Synchronous event.

## Comments

None.

# Type 2 (CAMAC 1D Counts)

## Description

This acquisition type is used to acquire particle sizing data from a CAMAC 1D interface card. The CAMAC 1D interface card is capable of interfacing to FSSP, ASASP, 1D-C, 1D-P, and IPC probes.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | CAMAC Slot | 1-23 |
| 2 | Probe command | 0-15 |
| 3 | 1D Interface number | 0-7 |

**Parameters**

## Data Size

This routine acquires forty two bytes of data for each sample and 42 bytes should be allocated for each sample.

## Data Format

The data acquired is in the following format:

| Byte Offset | Value |
|---|---|
| 0-1 | Size 1 Count |
| 2-3 | Size 2 Count |
| 4-5 | Size 3 Count |
| . | . |
| . | . |
| . | . |
| 28-29 | Size 15 Count |
| 30-31 | Strobe Count |

**Data Format**

| Byte Offset | Value |
|---|---|
| 32-33 | Spare 1 (total strobes) |
| 34-35 | Spare 2 (activity) |
| 36-37 | Spare 3 |
| 38-39 | Spare 4 |
| 40 | Range command value |
| 41 | Reference voltage value |

**Data Format (Continued)**

## Type

Synchronous event.

## Comments

None.

# Type 3 (CAMAC Digital Events E100)

## Description

This acquisition type is used to acquire digital event data from a DSP E100 digital event interface card mounted in a CAMAC crate. The DSP E100 contains two 16 digital input ports. This routine reads in sixteen bits of data from either of these two ports.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | CAMAC Slot | 1-23 |
| 2 | E100 Ports | 0-1 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires two bytes of data for each sample and two bytes should be allocated for each sample.

## Data Format

The data acquired is a packed sixteen bit event word with each event occupying one bit. Event 0 (1st event) occupies the lowest bit of the 16 bit word (b0), while event 15 (16th event) occupies the highest bit of 16 bit word (b15).

## Type

Synchronous event.

## Comments

None.

# Type 4 (CAMAC Loran C/GPS)

## Description

This acquisition type is used to acquire Loran/GPS data from a CAMAC Loran C/GPS interface card mounted in a CAMAC crate. The CAMAC Loran C/GPS interface card is a microprocessor controlled data preprocessor for serial RS232C Loran/GPS data streams. The on board preprocessor program can be changed to handle different Loran/GPS output formats. This modification is accomplished by replacing an EPROM.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | CAMAC Slot | 1-23 |
| 2 | Loran/GPS Command | 0-255 |
| 3 | Type | 0, 1, 2 |

**Parameters**

Parameter one is used to specify the CAMAC slot number. Parameter two must have the appropriate Loran/GPS command byte. Parameter three is used as the data type. Use zero for integer, one for float and two for character data types.

## Data Size

This routine acquires various data sizes depending on the command sent to the Loran/GPS card. Refer to the Loran/GPS documentation for information regarding data sizes.

## Data Format

The format of acquired data is dependent on the command sent to the Loran/GPS card. For data format information refer to the Loran/GPS documentation.

## Type

Synchronous event.

## Comments

This interface card converts incoming data stream to a standard Loran/GPS data format. This standard Loran/GPS data format is described in the Loran/GPS to CAMAC interface card documentation.

# Type 5 (2D Mono Image)

## Description

This acquisition type is used to acquire a 2D Mono image from a 2D Mono adapter. The adapter is a high performance 16 bit DMA interface using demand mode DMA. This design maximizes DMA performance while minimizing system bandwidth impact.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | DMA Channel | 5-7 (lower nibble) |
| 2 | Bit-Shift Divide | 0-F (upper nibble) |
| 3 | Rearm rate (Hz) | 1, 2,.. |

**Parameters**

The following table shows the possible values for the upper 4 bits of parameter two (bit shift). Parameter two should be entered as an hexadecimal number, for example 0x02 would give a 2MHz bit shift.

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 0 | 16 | 0.250 |
| 1 | 1 | 4.000 |
| 2 | 2 | 2.000 |
| 3 | 3 | 1.333 |
| 4 | 4 | 1.000 |
| 5 | 5 | 0.800 |
| 6 | 6 | 0.667 |
| 7 | 7 | .571 |
| 8 | 8 | .500 |

**Bit Shift**

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 9 | 9 | .444 |
| 0xA | 10 | .400 |
| 0xB | 11 | .364 |
| 0xC | 12 | .333 |
| 0xB | 13 | .307 |
| 0xE | 14 | .286 |
| 0xF | 15 | .267 |

**Bit Shift (Continued)**

The rearm rate, should be a non zero multiple of the system frequency. It represents the maximum rate at which 2D images will be recorded.

## Data Size

This routine acquires the 4096 byte image block of the 2D Mono probe. Each image contains 1024, 32 bit slices. Allocate 4096 bytes for this acquisition.

## Data Format

We keep the native data format from the instrument. Slices are 32 bits wide and first bit of each slice is stored in the lowest bit of a four byte slice, while the last bit is stored in the highest bit of a four byte slice.

## Type

Asynchronous Master event.

## Comments

The first bit out of the 2D Mono probe is termed the most significant. This acquisition stores this bit as the lowest with each sequential bit being stored one bit higher.

The bit shift rate should not be confused with the image strobe clock. The bit shift rate is the rate at which data is shifted out of the probe into the data system. It is constant and is set by the upper nibble of parameter two.

The strobe clock controls the rate at which image slices are shifted into the probe. It varies with true air speed and pixel size. The strobe clock is set by the control function Co2DTAS().

The maximum rate that can be used for the bit shift clock depends on the length of the cable between the probe and the data system. A 1 MHz (divide factor 4) should be adequate for the majority of installations where the cable length is less than 50 feet. If longer cables are used, the user should try slower rates. The most common symptoms of a too high a bit shift rate are image jitter or missing pixels.

# Type 6 (2D Mono TAS Factors)

## Description

This acquisition type is used to acquire a 2D Mono TAS factors from a 2D Mono adapter. These TAS factors are the multiply and divide factors used to generate the TAS clock need to strobe image slices into the 2D Mono probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires two 16 bit words, the multiply and divide factors respectively. Therefore 4 bytes should be reserved for each sample.

## Data Format

Each of the 16 bit factors are stored in two successive word locations. The multiply factor is stored first, followed by the divide factor.

## Type

Asynchronous slave event.

## Comments

The true air speed clock frequency is equal to the multiply factor times 50 KHz divided by the divide factor.

# Type 7 (2D Mono Elapsed Time)

## Description

This acquisition type is used to acquire a 2D Mono elapsed time value from a 2D Mono adapter. Elapsed time is the number of 25s increments that have passed since the time the probe was armed and the probe became full.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 32 bit word. Four bytes should be allocated for this sample.

## Data Format

The 32 bit word is an unsigned long integer whose value represents the number of 25 s ticks that have passed between the arming and filling of a 2D Mono probe.

## Type

Asynchronous slave event.

## Comments

This acquisition should be performed once at the end of each image. From the combination of the date time data and the elapsed time data one may determine the actual time when the probe became full.

# Type 8 (2D Mono Elapsed TAS/100)

## Description

This acquisition type is used to acquire a 2D Mono elapsed TAS/100 value from a 2D Mono adapter. Elapsed TAS/100 is the number of true airspeed clocks divided by 100 that have passed since the time the probe was armed and the probe became full.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 32 bit word. Four bytes must be allocated per sample.

## Data Format

The 32 bit word unsigned long integer counts the number of TAS clock tick that have passed between the arming and filling of the 2D Mono probe.

## Type

Asynchronous slave event.

## Comments

This acquisition should be taken at the end of each image. The true air speed clock gives an indication of the spatial separation between when the probe was armed and when the probe became full.

You can convert elapsed TAS/100 to a distance (in the same units as the pixel size) by using the following formula.

$$\Delta D = RawCounts \cdot 100 \cdot PixelSize$$

You can convert elapsed TAS/100 to a time by using the following formula.

$$\Delta T = RawCounts \cdot \frac{DivideFactor}{MultiplyFactor} \cdot 2{\times}10^{-3}(s)$$

# Type 9 (2D Mono Elapsed Shadow OR)

## Description

This acquisition type is used to acquire a 2D Mono elapsed shadow-or count form a 2D Mono adapter. Elapsed shadow-or count is the number of times the shadow-or output of the probe was active while the probe was armed.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The 16 bit word is an unsigned integer counting the number of times the shadow-or line when active while the 2D Mono probe was armed.

## Type

Asynchronous slave event.

## Comments

This acquisition should be taken at the end of each image. This provides and approximate check with the number of particles in the image buffer.

# Type 10 (2D Mono Total Shadow OR)

## Description

This acquisition type is used to acquire a 2D Mono total shadow-or count from a 2D Mono adapter. Total shadow-or count is the number of times the shadow-or output of the probe went active from the last time this data was acquired.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The 16 bit word is an unsigned integer counting the number times the shadow-or when active.

## Type

Synchronous event.

## Comments

This is an synchronous acquisition and should be acquired at regular intervals. This gives an approximate value to the number of particles that past through the probe.

The following equation shows how to arrive at shadow-or concentration in number of particles per liter from raw shadow-or counts (number particles).

$$Conc = \frac{Number\ Particles}{l}$$

$$TAS = \frac{m}{s}$$

$$Sample\ Area\ (SA) = mm^2$$

$$Sample\ Volume\ (SV) = m^3$$

$$Sample\ Period\ (\Delta t) = 1\ s$$

$$SV\ (m^3) = SA\ (mm^2) * (0.001 * \frac{m}{mm})^2 * TAS\ (\frac{m}{s}) * \Delta t\ (s)$$

$$SV\ (m^3) = 10^{-6} * SA\ (mm^2) * TAS\ (\frac{m}{s})$$

$$SV\ (l) = SV\ (m^3) * 10^3\ (\frac{l}{m3})$$

$$SV\ (l) = 10^{-6} * 10^3 * SA\ (mm^2) * TAS\ (\frac{m}{s})$$

$$SV\ (l) = 10^{-3} * SA\ (mm^2) * TAS\ (\frac{m}{s})$$

$$Conc\ (\frac{Number\ Particles}{l}) = \frac{Number\ Particles}{SV}$$

$$Conc\ (\frac{Number\ Particles}{l}) = \frac{Number\ Particles}{10^{-3} * SA\ (mm^2) * TAS\ (m/s)}$$

# Type 11 (2D Mono House Data)

## Description

This acquisition type is used to acquire a 2D Mono house data counts from a 2D Mono adapter. House data is an array of eight 16 bit words containing counts of a voltage to frequency convertor built into the probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | | |
| 3 | Probe Type | 0, 1 |

**Parameters**

Parameter three is used to select house data for the 2D Mono probes (zero), or the HVPS probes (one).

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The array of eight 16 bit words represents the counts of the output of an eight channel voltage to frequency converter.

## Type

Synchronous event.

## Comments

This is an synchronous acquisition and should be acquired at regular intervals. At every acquisition the house is advanced and the data for the next channel is recorded. Eight housekeeping channels are acquired in succession and stored in an array in memory. The acquisition returns this array. The values returned represent the most recent data for that channel. The data will be between 1 and 8 sample intervals old.

# Type 12 (DT2801 Analog)

## Description

This acquisition type is used to acquire an analog channel from a DT2801 analog to digital converter adapter.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Analog Channel | 0-7 |
| 2 | Gain | 0-3 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The data acquired is in two's complement integer coding and represents the signed digital value of the analog signal. The 16 bit word is stored as a signed integer.

| | | |
|---|---|---|
| 0x7FFF | 32767 | +full scale |
| 0x0000 | 0 | zero |
| 0x8000 | -32767 | -full scale |

**Data Format**

## Type

Synchronous event.

## Comments

This is a 12 bit two's complement number.

# Type 13 (DT2801 Digital Events)

## Description

This acquisition type is used to acquire a digital event port from the DT2801 adapter. There are 8 events per digital port.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Port | 0-1 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample. The low byte has the event data while the high byte is zero (high byte used for even pad).

## Data Format

The data acquired is in a simple bit per event format. There are 8 events per port (byte), where D0 corresponds to the first event and D7 corresponds to the last event.

## Type

Synchronous event.

## Comments

None.

# Type 14 (Loran C/GPS)

## Description

This acquisition type is used to acquire data from a Loran/GPS adapter mounted in the back plane. The Loran/GPS adapter is a microprocessor controlled data preprocessor for RS232C Loran/GPS data streams. The on board preprocessor program can be changed to handle different Loran/GPS output formats. This modification is accomplished by replacing the EPROM.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | LORAN/GPS Command | 0-255 |
| 2 | LORAN/GPS Control Byte | 0-255 |
| 3 | Type | 0, 1, 2 |

**Parameters**

The Loran/GPS control byte should be set up to have a value that will be compatible with the protocol of the data being received. The meaning of the control byte is as follows:

| Bit | Usage |
|---|---|
| $B_0$ | Parity on (1), parity off (0). |
| $B_1$ | Parity odd (1), parity even (0). |
| $B_6$ | 2 stop bits (1), 1 stop bit (0). |
| $B_7$ | 8 data bits (1), 7 data bits (0). |
| $B_2$ | Lowest bit for baud rate control nibble. |
| $B_3$ | Next to lowest. |
| $B_4$ | Next to highest. |
| $B_5$ | Highest bit for baud rate control nibble. |

**Control Byte Meaning**

The baud rates supported are as follows:

| Nibble Value | Baud Rate |
|---|---|
| 0000 | 19,200 |
| 0001 | 9,600 |
| 0010 | 7,200 |
| 0011 | 4,800 |
| 0100 | 3,600 |
| 0101 | 2,400 |
| 0110 | 2,000 |
| 0111 | 1,800 |
| 1000 | 1,200 |
| 1001 | 600 |
| 1010 | 300 |
| 1011 | 200 |
| 1100 | 150 |
| 1101 | 115,200 |
| 1110 | 57,600 |
| 1111 | 38,400 |

**Baud Rates**

Parameter two must have the appropriate Loran/GPS command byte. Parameter three is used as the data type. Use zero for integer, one for float and two for character data types.

## Data Size

This routine acquires various data sizes depending on the command sent to the Loran/GPS card. Refer to the Loran/GPS adapter documentation for information regarding data sizes.

## Data Format

The format of acquired data is dependent on the command sent to the Loran/GPS card. For data format information refer to the Loran/GPS adapter card documentation.

## Type

Synchronous event.

## Comments

This adapter converts incoming data stream to a standard Loran/GPS data format. This standard Loran/GPS data format is described in the Loran/GPS adapter documentation.

# Type 15 (Encoding Altimeter)

## Description

This acquisition type is used to acquire encoding altimeter data from an encoding altimeter adapter mounted in the back plane.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The format of acquired data is an unsigned two byte word returning the altitude in feet.

## Type

Synchronous event.

## Comments

This adapter converts the Grey coded encoding altimeter output into a unsigned integer represented the altitude in feet.

# Type 16 (INS Arinc Serial)

## Description

This acquisition type is used to acquire serial data from the Arinc and Arinc429 interfaces. These adapters contain on board RAM, which captures the serial data transmitted from the INS.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Arinc Label | 0-255 |
| 2 | Arinc Receiver | 0, 1 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires 32 bits of data. Four bytes should be allocated for each sample.

## Data Format

This acquisition event stores the 24 bits of ARINC raw data followed by 8 bits of an update count. The ARINC interface drops the ARINC label from the data and replaces that byte with an update count. The update counter is set to zero at power up and it increments each time a label is received.

| Byte Offset | Value |
|---|---|
| 0 | ARINC LSB |
| 1 | ARINC |
| 2 | ARINC MSB |
| 3 | update counter |

**Data Format**

## Type

Synchronous event.

## Comments

This adapter can be used to acquire either binary or BCD serial streams. Two adapters can be used to acquire both streams.

If the update counter does not change between acquisitions then the data returned was not updated by the INS during the time between the two acquisitions. If the update counter changes by more than one between acquisitions, then some data was updated more than once between acquisitions.

After reaching 255 the update counter rolls to 0 and continues counting.

# Type 17 (INS Synchro)

## Description

This acquisition type is used to acquire angular position from a synchro channel in the INS adapter. This adapter supports up to four synchro to digital converters. These synchro channels can be used to acquire pitch, roll and yaw or other synchro encoded information.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Synchro Channel | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The format of the acquired data is a 16 bit unsigned integer. This integer represents angles in the range of [0, 2π] radians.

## Type

Synchronous event.

## Comments

None.

# Type 18 (CAMAC INS ARINC Serial)

## Description

This acquisition type is used to acquire INS ARINC serial data from an CAMAC INS interface card. This adapter contains an on board RAM which captures the serial data transmitted from the INS system.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | CAMAC Slot | 1-23 |
| 2 | INS ARINC Label | 0-255 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires 32 bits of data. Four bytes should be allocated for each sample.

## Data Format

The format of the acquired data is 24 bits of data followed by 8 bits of an update count. The update count is zeroed at power up and it increments each time a label is updated by the INS. The first three bytes of data are the 24 bits of INS data. The update count is the last byte in memory.

## Type

Synchronous event.

## Comments

This interface card can be used to acquire either binary or BCD serial streams. Two interface card can be used to acquire both streams.

If the update counter does not change between acquisitions then the data returned was not updated by the INS during the time between the two acquisitions. If the update counter changes by more than one between acquisitions, then some data was updated more than once between acquisitions.

# Type 19 (CAMAC INS Synchro)

## Description

This acquisition type is used to acquire an INS synchro channel in the CAMAC INS interface card. This interface supports up to four synchro to digital converters. These synchro channels can be used to acquire pitch, roll and yaw information.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | CAMAC Slot | 1-23 |
| 2 | Synchro Channel | 0-3 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The format of the acquired data is a 16 bit unsigned integer. This integer represents angles from 0 - 2π radians.

## Type

Synchronous event.

## Comments

None.

# Type 20 (2D Grey Image)

## Description

This acquisition type is used to acquire a 2D Grey image form a 2D Grey adapter. This interface card is a high performance 16 bit DMA interface using demand mode DMA. This design maximizes DMA performance while minimizing system bandwidth impact.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | DMA Channel | 0x5-0x7 (lower nibble) |
| 2 | Bit Shift Divide | 0x8-0xF (upper nibble) |
| 3 | Rearm Rate (Hz) | 1, 2 ... |

**Parameters**

The following table shows the possible values for the upper 4 bits of parameter 2. The 2D Grey interface takes two clock cycles to unload one bit from the probe. In fact, the bit shift rate is half of the clock frequency.

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 0x0 | 16 | 0.250 |
| 0x1 | 1 | 4.000 |
| 0x2 | 2 | 2.000 |
| 0x3 | 3 | 1.333 |
| 0x4 | 4 | 1.000 |
| 0x5 | 5 | 0.800 |
| 0x6 | 6 | 0.667 |
| 0x7 | 7 | 0.571 |

**Bit Shift**

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 0x8 | 8 | 0.500 |
| 0x9 | 9 | 0.444 |
| 0xA | 10 | 0.400 |
| 0xB | 11 | 0.364 |
| 0xC | 12 | 0.333 |
| 0xD | 13 | 0.307 |
| 0xE | 14 | 0.286 |
| 0xF | 15 | 0.267 |

**Bit Shift (Continued)**

The rearm rate, should be a non zero multiple of the system frequency. It represents the maximum rate at which 2D Grey images will be recorded.

## Data Size

This routine acquires a variable length image block depending on the bytes per sample field of the acquisition table and the number of slices in an image. The actual data acquired is the minimum of the two sizes.

## Data Format

Slices are 128 bits wide and first bit of each slice is stored in the lowest bit of a sixteen byte slice, while the last bit is stored in the highest bit of a sixteen byte slice.

## Type

Asynchronous master event.

## Comments

The first slice contains a repeated 16 bit particle count value. This value is shifted so that it corresponds with the low byte high byte convention of the Intel data format. For each image slice, the first bit shifted in is stored in the lowest bit of the 128 bit slice and the last bit shifted in is stored in the highest bit of the 128 bit slice. This cause an order inversion of the shadow bit pairs (00=none, 10=min, 01=mid, 11=max).

The bit shift rate should not be confused with the image strobe clock. The bit shift rate is the rate at which data is shifted out of the probe into the data system. It is constant and is set by the upper nibble of parameter two.

The strobe clock controls the rate at which image slices are shifted into the probe. It varies with true air speed and pixel size. The strobe clock is set by the control function Co2GTAS().

The maximum rate that can be used for the bit shift clock depends on the length of the cable between the probe and the data system. A 1 MHz (divide factor of 4) should be adequate for the majority of installations where the cable length is less than 50 feet. If longer cables are used, the user should try slower rates. The most common symptoms of too high a bit shift rate are image jitter or missing pixels.

# Type 21 (2D Grey TAS Factors)

## Description

This acquisition type is used to acquire a 2D Grey TAS factors from a 2D Grey adapter. These factors are the multiply and divide factors used to generate the TAS clock need to strobe the 2D Grey probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires two 16 bit words, the multiply and divide factors respectively. Therefore 4 bytes should be reserved for each sample.

## Data Format

Each of the 16 bit factors are stored in two successive word locations. The first being the multiply factor followed by the divide factor.

## Type

Asynchronous slave event.

## Comments

The true air speed clock frequency is evaluated by the multiply factor times 50 KHz divided by the divide factor.

For certain applications, the actual TAS may be higher than the maximum TAS the probe can sample at. In these cases, the TAS out to the probe should be limited to the maximum. The actual particle sizes can be approximated by taking the actual TAS and divide it by the true air speed clock frequency.

# Type 22 (2D Grey Elapsed Time)

## Description

This acquisition type is used to acquire a 2D Grey elapsed time value from a 2D Grey adapter. Elapsed time is the number of 25 µs ticks that have passed since the time the probe was armed and when the particle image was recorded.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 32 bit word. Four bytes should be allocated for this sample.

## Data Format

The 32 bit word unsigned long word counts the number of 25 µs ticks that have passed while the probe was armed.

## Type

Asynchronous slave event.

## Comments

None.

# Type 23 (2D Grey Elapsed TAS/256)

## Description

This acquisition type is used to acquire a 2D Grey elapsed TAS/256 value from a 2D Grey adapter. Elapsed TAS/256 is the number of true airspeed clocks divided by 256 that have passed since the time the probe was armed and when the image was recorded.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 32 bit word. Four bytes must be allocated per sample.

## Data Format

The 32 bit unsigned long word contains the counts of the TAS clock divided by 256 while the 2D Grey probe was armed.

## Type

Asynchronous slave event.

## Comments

This acquisition should be taken at the end of each image. The true air speed clock gives and indication of the spatial separation between when the probe was armed and when the probe became full.

You can convert elapsed TAS/256 to a distance (in the same units as the pixel size) by using the following formula.

$$\Delta D = RawCounts \times 256 \times PixelSize$$

You can convert elapsed TAS/256 to a time by using the following formula.

$$\Delta T = RawCounts \times \frac{DivideFactor}{MultiplyFactor} \times 5.12 \times 10^{-3} \text{ (s)}$$

# Type 24 (2D Grey Minimum Count)

## Description

This acquisition type is used to acquire the number of pixels that where shaded at the minimum level in a 2D Grey image. This count is generated while the image is being shifted in from the probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

This 16 bit word is an unsigned integer containing the count of the number of pixels at the minimum level.

## Type

Asynchronous slave event.

## Comments

It is possible to approximate the particle area by adding the minimum, middle and maximum shadow.

# Type 25 (2D Grey Middle Count)

## Description

This acquisition type is used to acquire the number of pixels that where shaded at the middle level in a 2D Grey image. This count is generated while the image is being shifted in from the probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

This 16 bit word is an unsigned integer containing the count of the number of pixels at the middle level.

## Type

Asynchronous slave event.

## Comments

It is possible to approximate the particle area by adding the minimum, middle and maximum shadow.

# Type 26 (2D Grey Maximum Count)

## Description

This acquisition type is used to acquire the number of pixels that where shaded at the maximum level in a 2D Grey image. This count is generated while the image is being shifted in from the probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

This 16 bit word is an unsigned integer containing the count of the number of pixels at the maximum level.

## Type

Asynchronous slave event.

## Comments

It is possible to approximate the particle area by adding the minimum, middle and maximum shadow.

# Type 27 (2D Grey OR Slice)

## Description

This acquisition type is used to acquire a 2D Grey scale or slice from a 2D Grey adapter. This slice is the bit wise or of all the slices in an image. This produces an quick approximation to the horizontal dimension of a particle. This also allows for a quick check of whether or not the particle touches the edge.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | DMA Channel | 5-7 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 128 bit slice. Sixteen bytes should be allocated for this sample.

## Data Format

The 128 bit slice is stored in the same format as the slices of the image. The first bit is stored in the lowest bit position and the last bit is stored in the highest bit position.

## Type

Asynchronous slave event.

## Comments

The DMA channel selected must be the same as that specified for this interface card in the 2D Grey acquisition type. It must also be matched to the 2D Grey interface DIP switch settings.

# Type 28 (2D Grey Shadow Slice Count)

## Description

This acquisition type is used to acquire the number of slices that had a bit shadowed in a 2D Grey image. This becomes useful in determining vertical dimension of an image.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The 16 bit word is stored in low byte to high byte format with the first byte in memory being the lowest byte and the last byte in memory being the highest byte.

## Type

Asynchronous slave event. This acquisition event must follow acquisition type 20 (a master event) in an asynchronous buffer in the acquisition table. The number of samples can be one or two. One sample is always recorded at the end of the master acquisition. A second sample may also be taken at the beginning of the master acquisition.

## Comments

None.

# Type 29 (2D Grey Probe Byte)

## Description

This acquisition type is used to acquire the command word sent out to the 2D Grey scale probe. This command word can be useful in post processing to determine the mode the probe was in while taking data.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey Interface | 0-3 |
| 2 | Initial Command | 0-0xFF |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The 16 bit word is stored in low byte to high byte format with the first byte in memory being the lowest byte and the last byte in memory being the highest byte.

## Type

Synchronous event.

## Comments

This acquisition should be as a synchronous acquisition event and sampled at regular intervals. It also can be used as an asynchronous acquisition event and stored in the Grey scale image buffer (only the lower 8 bits are used by the 2D Grey probe).

# Type 30 (1D Counts)

## Description

This acquisition type is used to acquire the 1D spectral data from a 1D interface card. This single slot interface adapter card is capable of interfacing to FSSP, ASASP, 1D-C, 1D-P, and IPC probes.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 1D Interface | 0-7 |
| 2 | Probe Command | 0-15 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires forty two bytes of data for each sample and 42 bytes should be allocated for each sample.

## Data Format

The data acquired is in the following format:

| Byte Offset | Value |
|---|---|
| 0-1 | Size 1 count |
| 2-3 | Size 2 count |
| 4-5 | Size 3 count |
| . | . |
| . | . |
| . | . |
| 28-29 | Size 15 count |
| 30-31 | Strobe count |
| 32-33 | Spare 1 (total strobes) |

**Data Format**

| Byte Offset | Value |
|---|---|
| 34-35 | Spare 2 (activity) |
| 36-37 | Spare 3 |
| 38-39 | Spare |
| 40 | Range command value |
| 41 | Reference voltage value |

**Data Format (Continued)**

## Type

Synchronous event.

## Comments

The 1D interface card provides twenty sixteen bit counter channels. The first fifteen counter channels are used to record the fifteen sizes generated by the probes. The sixteenth channel is used to record the number of strobes. The remaining four channels may be used to count other events such as total strobe, total resets and or activity.

Please note that not all 1D probes have total strobes and activity in the order given above. You should check the 1D probe manual for information on which signals are available and where.

To control the probe command (probe range or pump on/off), use the Co1DCmd() function.

# Type 31 (Hail Spectrometer)

## Description

This acquisition type is used to acquire the hail spectral data from the Hail interface card. This single slot interface adapter card has fifteen sixteen bit counter channels and twenty four single bit events. This acquisition routine acquires the fifteen counter channel data.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Counters 8-14 | 0x7F |
| 2 | Counter 0-7 | 0xFF |
| 3 | | |

**Parameters**

This routine uses the first two parameters to control the reset mode for each counter. As shown above, each counter is controlled by a single bit in parameter one or two. If the control bit for a counter is zero, the counter will be reset to zero each time the counter value is acquired. The acquired count will represent the number of counts that occurred since the last acquisition. If the control bit for a counter is a one, the counter will not be reset during data acquisition. The count will then represent the total number of counts that have occurred since the system was started. Their control bits are assigned as follows:

## Data Size

This routine acquires thirty bytes of data for each sample and 30 bytes should be allocated for each sample.

## Data Format

| Byte Offset | Value |
|---|---|
| 0-1 | Size 1 count |
| 2-3 | Size 2 count |
| 4-5 | Size 3 count |
| . | . |

**Data Format**

| Byte Offset | Value |
|---|---|
| . | . |
| . | . |
| 28-29 | size 15 count |

**Data Format (Continued)**

## Type

Synchronous event.

## Comments

The Hail interface card provides fifteen sixteen bit counter channels. These fifteen counter channels could be used to count pulsed from any TTL source. The twenty four single bit events can be used to acquire TTL events.

# Type 32 (Hail Events)

## Description

This acquisition type is used to acquire the hail event data from a Hail interface card. This single slot interface adapter card has fifteen sixteen bit counter channels and twenty four single bit events. This acquisition routine acquires the twenty four event bit data.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires four bytes of data for each sample and four bytes should be allocated for each sample.

## Data Format

The data acquired is in the following format.

| Bye Offset | Value |
|---|---|
| 0 | Event bits 0-7 |
| 1 | Event bits 8-15 |
| 2 | Event bits 16-23 |
| 3 | byte pad |

**Data Format**

## Type

Synchronous event.

## Comments

The Hail interface card provides fifteen sixteen bit counter channels. These fifteen counter channels could be used to count pulsed from any TTL source. The twenty four single bit events can be used to acquire any TTL events.

# Type 33 (Analog STB-TC Analog)

## Description

This acquisition type is used to acquire analog data. It has support for the DT2817 and ATAQ141X boards.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Channel | 0-3 |
| 2 | Mode | See Parameter 2 |
| 3 | Gain | See Parameter 3 |

**Parameters**

Parameters 2 and 3 are used for documentation only. The M300 sets the values for them based on the board configuration.

| Parameter 2 | Mode |
|---|---|
| xx0 | Bipolar |
| xx1 | Unipolar |
| x0x | AT1412 |
| x1x | AT1411 |
| 0xx | Single ended |
| 1xx | Differential input |

**Parameter 2**

| Parameter 3 | Gain |
|---|---|
| 0 | User |
| 1 | 0.5 |
| 2 | 1 |
| 3 | 2 |

**Parameter 3**

| Parameter 3 | Gain |
|---|---|
| 4 | 4 |
| 5 | 5 |
| 6 | 8 |
| 7 | 10 |
| 8 | 50 |
| 9 | 100 |
| 10 | 500 |
| 11 | 1000 |

**Parameter 3 (Continued)**

## Data Size

This routine acquires a 16 bit word. Two bytes should be allocated for this sample.

## Data Format

The data acquired is in two's complement integer coding and represents the signed digital value of the analog signal.

| | | |
|---|---|---|
| 0x7FFF | 32767 | +full scale |
| 0x0000 | 0 | zero |
| 0x8000 | -32768 | -full scale |

**Data Format**

## Type

Synchronous event.

## Comments

None.

# Type 34 (Digital Input)

## Description

This acquisition type is used to acquire one byte of digital data from the digital input\output card. This works with the DT2817, CYPDISO, CYDIO24, PCIDAC, PMF and ATDAQ141X.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Input Port | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires a 8 bit byte. Two bytes of data should be allocated for each sample.

## Data Format

The data format represents the state of the digital input port. The state is usually '1' for on and '0' for off. Most of the boards have 8 input lines, represented by the corresponding bits in the data word. The majority of the board use only the lower byte leaving the upper byte at 0. In the case of the PMF board it uses all 16 bits, since the board has 16 digital lines.

## Type

Synchronous event.

## Comments

None.

# Type 35 (SEA Analog to Digital Input)

## Description

This acquisition type is used to acquire two bytes of data from the sea analog to digital input board. The PMF (PowerDAQ Multi-Function) board can also be used with this acquisition event (leave box id parameter set to 0). In differential input mode, the PMF board uses channels 0-7, 16-23, 32-39 and 48-55.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Box ID | 0-255 |
| 2 | Channel | 0-63 |
| 3 | Gain | 0-3 |

**Parameters**

## Data Size

This routine acquires two 8 bit bytes. Two bytes of data should be allocated for each sample.

## Data Format

The data acquired is in two's complement integer coding and represents the signed digital value of the analog signal.

| | | |
|---|---|---|
| 0x7FFF | 32767 | +Full Scale |
| 0x0000 | 0 | Zero |
| 0x8000 | -32767 | -Full Scale |

**Data Format**

## Type

Synchronous event.

## Comments

To obtain the correct voltage, multiply the raw count by the voltage range and divide by 65536.

| Voltage Range | Parameter 3 | Gain | Multiply By |
|---|---|---|---|
| ± 10 | 0 | 1 | 3.051757813E-4 |
| ± 5 | 1 | 2 | 1.525878906E-4 |
| ± 2.5 | 2 | 4 | 7.629394531E-5 |
| ± 1.25 | 3 | 8 | 3.814755474E-5 |

**Voltage Calculation**

The id parameter is the id code of the particular 32 channel converter box where the channel resides. At the present time the first A/D converter box is id 0, the second A/D converter box is id 1, etc.

# Type 36 (SEA 24 Counter)

## Description

This acquisition type is used to acquire data from the SEA 24 Counter board.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This acquisition type requires 96 bytes of data (24 * 4).

## Data Format

The data acquired is four bytes per counter.

## Type

Synchronous event.

## Comments

None.

# Type 37 (Serial ASCII Data)

## Description

This acquisition type is used to acquire serial ASCII data from the SEA serial interface card or any serial port including boards which add serial ports to the system. The serial data should be blocked by carriage return and line feed characters.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Block | 0-255 |
| 2 | | |
| 3 | Throttle | 1, 2,.. |

**Parameters**

The block parameter should be used to tell the system about the last character in a data block.

The throttle byte should be a non zero multiple of the system frequency. It represents the maximum rate at which ASCII data blocks may be recorded per second. It should be significantly larger than the maximum block rate that will be received. If the throttle rate is less than or equal to the actual block rate, the internal FIFO will never completely empty. This will increase data latency.

## Data Size

The data size specified in the acquisition table, should be equal to the number of bytes in the largest serial data block. The data size is automatically resized to the actual number of bytes in the serial block.

## Data Format

The data acquired corresponds to the ASCII block sent.

## Type

Asynchronous master event or synchronous event. This data type provides improved flexibility over the M200 version because it supports both Asynchronous and Synchronous events.

If an Asynchronous master event is used the buffer automatically resizes for the number of bytes coming in. This also implies that the bytes per sample are the largest block of data to be collected. This acquisition event must be the first event of an asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

If a Synchronous event is being used the buffer size is not dynamic and must have the bytes per sample set to a user specified value large enough to collect the data. This value is the maximum

number of bytes that can be acquired at one time. If the buffer is not filled during acquisition then all remaining bytes will be set to zero.

## Comments

None.

# Type 38 (Serial IEEE Data)

## Description

This acquisition type is used to acquire serial IEEE (floating point) data from the SEA serial interface card or any serial port including boards which add serial ports to the system. The serial data should be blocked by a non number IEEE value, followed by the word 0xFF55 (i.e. FFFFFFFFFF55).

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Swap bytes | 0, 1 |
| 2 | | |
| 3 | Throttle | 1, 2,.. |

**Parameters**

The high nibble for parameter one is reserved, and should be 0. Bit one of the lower nibble is used to specify data swap (0 no swap, 1 swap).

The throttle byte should be a non zero multiple of the system frequency. It represents the maximum rate at which IEEE data blocks may be recorded per second. It should be significantly larger than the maximum block rate that will be received. If the throttle rate is less than or equal to the actual block rate, the internal FIFO will never completely empty. This will increase data latency.

## Data Size

The data size specified in the acquisition table, should be equal to the number of bytes in the largest serial data block. The data size is automatically resized to the actual number of bytes in the serial block.

## Data Format

The data acquired corresponds to the IEEE data sent.

## Type

Asynchronous master event or synchronous event. This data type provides improved flexibility over the M200 version because it supports both Asynchronous and Synchronous events.

If an Asynchronous master event is used the buffer automatically resizes for the number of bytes coming in. This also implies that the bytes per sample are the largest block of data to be collected. This acquisition event must be the first event of an asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

If a Synchronous event is being used the buffer size is not dynamic and must have the bytes per sample set to a user specified value large enough to collect the data. This value is the maximum number of bytes that can be acquired at one time. If the buffer is not filled during acquisition then all remaining bytes will be set to zero.

## Comments

None.

# Type 39 (Serial Integer Data)

## Description

This acquisition type is used to acquire serial integer data from the SEA serial interface card or any serial port including boards which add serial ports to the system. The serial data should be blocked by nine 0xAA bytes followed by a 0x55 byte.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Data Type | 0-3 |
| 2 | | |
| 3 | Throttle | 1, 2,.. |

**Parameters**

The high nibble for parameter one is reserved, and should be 0. Bit zero of the lower nibble is used to specify data type, (0 integer 16 bits, 1 long integer 32 bits). Bit one is used to specify data swap (0 no swap, 1 swap).

The throttle byte should be a non zero multiple of the system frequency. It represents the maximum rate at which integer data blocks may be recorded per second. It should be significantly larger than the maximum block rate that will be received. If the throttle rate is less than or equal to the actual block rate, the internal FIFO will never completely empty. This will increase data latency.

## Data Size

The data size specified in the acquisition table, should be equal to the number of bytes in the largest serial data block. The data size is automatically resized to the actual number of bytes in the serial block.

## Data Format

The data acquired corresponds to the integer data sent.

## Type

Asynchronous master event or synchronous event. This data type provides improved flexibility over the M200 version because it supports both Asynchronous and Synchronous events.

If an Asynchronous master event is used the buffer automatically resizes for the number of bytes coming in. This also implies that the bytes per sample are the largest block of data to be collected. This acquisition event must be the first event of an asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

If a Synchronous event is being used the buffer size is not dynamic and must have the bytes per sample set to a user specified value large enough to collect the data. This value is the maximum number of bytes that can be acquired at one time. If the buffer is not filled during acquisition then all remaining bytes will be set to zero.

## Comments

None.

# Type 40 (Sonic Wind System)

## Description

This acquisition type is used to acquire serial binary data from the Sonic Wind System. This acquisition type can be used with the serial port and SEA serial interface

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Trigger | 0-255 |
| 2 | Control | 0-255 |
| 3 | Serial Port | 0-7 (lower nibble) |
| 3 | Segment | 0xC, 0xD, 0xE (upper nibble) |

**Table 2: Parameters**

Parameter one is the trigger character for the Sonic anemometer. Use ‘U' (0x55) for the CSAT3 Sonic and ‘T' (0x54) for the ATI Sonic.

Parameter two is the control byte, used to setup the communication protocol for the port. The serial control byte should be set to have a value that will be compatible with the protocol of the data being received.

Parameter three low nibble is the ACL port number. The port parameter is used to specify the communications port used to receive the data.

Parameter three upper nibble is the RAM segment for the ACL board.

The meaning of the control byte is as follows:

| Bit | Usage |
|---|---|
| $B_0$ | Lowest bit for baud rate control nibble. |
| $B_1$ | Next to lowest. |
| $B_2$ | Next to highest. |
| $B_3$ | Highest bit for baud rate control nibble. |
| $B_4$ | Parity odd (0), parity even (1). |
| $B_5$ | 2 stop bits (1), 1 stop bit (0). |

**Table 3: Control Byte Values**

| Bit | Usage |
|---|---|
| $B_6$ | Parity on (1), parity off (0). |
| $B_7$ | 8 data bits (1), 7 data bits (0). |

**Table 3: Control Byte Values (Continued)**

The baud rates supported are as follows:

| Nibble Value | Baud Rate |
|---|---|
| 1100 | 19200 |
| 1011 | 9600 |
| 1010 | 4800 |
| 1001 | 2400 |
| 1000 | 2000 |
| 0111 | 1800 |
| 0110 | 1200 |
| 0101 | 600 |
| 0100 | 300 |
| 0011 | 150 |
| 0010 | 135 |
| 0001 | 110 |
| 0000 | 75 |

**Table 4: Baud Rates**

The RAM segment specifies the high nibble for the dual RAM for the ACL communications board.

***Note: The user must specify the correct control byte in this parameter, in order to establish communication.***

## Data Size

The data size specified in the acquisition table, should be equal to the number of bytes in the largest serial data block. The actual number of bytes per sample, will always be the same, regardless of the number of bytes received.

For the ATI Sonic the maximum number of data bytes is 10. Two bytes for the synchronization pattern and two bytes for each different field (0x8000, U, V, W, T). This acquisition type should not be sampled at frequencies above 10 Hz for the ATI Sonic. The byte order for each field is reversed.

For the CSAT3 Sonic the maximum number of data bytes is 10. Two bytes for each different field (U, V, W, T, F).

## Data Format

The data acquired corresponds to the binary data sent by the Sonic instrument.

## Type

Synchronous event.

## Comments

None.

# Type 41 (Falcon Data)

## Description

This acquisition type is used to acquire data from the SEA Falcon interface card.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Clock Mode | 0-2 |
| 2 | Clock Divider | 0-255 |
| 3 | Throttle | 1, 2,.. |

**Parameters**

This routine uses parameter one as the clock mode, two as the frequency divider, and parameter three as the throttle.

The clock mode is used to specify the clock source for the Model 200 System. A value of zero, will use only the Model 200 internal clock. A value of one, will only use the external Falcon word rate as the clock. A value of two will cause the Model 200 internal clock to be used until the Falcon data begins arriving and will then switch to the Falcon word rate for the clock signal.

The clock divider is used to divide the Falcon word clock down, to match the system clock frequency in the system table (i.e. system frequency equals 100 Hz, Falcon data rate 3500 words/second, divider equals 35 equals Falcon data rate divided by system frequency).

The throttle byte should be a non zero multiple of the system frequency. It represents the maximum rate at which Falcon data blocks may be recorded per second. It should be significantly larger than the maximum block rate that will be received. If the throttle rate is less than or equal to the actual block rate, the internal FIFO will never completely empty. This will increase data latency.

## Data Size

The data size specified in the acquisition table, should be equal to the number of bytes in the largest data block times the sample frequency. This is normally 7000 bytes (350 words * 2 bytes per word * 10 samples). The data size is automatically resized to the actual number of bytes in the acquisition table, or the number of bytes taken, which ever is less.

## Data Format

The data acquired corresponds to a block of data as specified by the falcon data format.

## Type

Asynchronous master event.

## Comments

None.

# Type 42 (INS Accelerometer)

## Description

This acquisition type is used to acquire data from the INS Accelerometer interface card. There are three 32 bits counters, that can receive positive or negative counts.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Counter | 0-2 |
| 2 | Polarity | 0-3 |
| 3 | Reset | 0, 1 |

**Parameters**

The counter parameter is used to specify the acceleration direction for the data acquired. A value of zero for the X direction (counter 0), one for the Y direction (counter 1), and two for the Z direction (counter 2).

The polarity parameter is used to select the pulse polarity for both the up and down input signals for the specified channel. A value of zero, represents positive polarity for both signals. A value of one represents negative polarity for the up signal and positive polarity for the down signal. A value of two, represents positive polarity for the up signal and negative polarity for the down signal. Finally a value of three, represents negative polarity for both signals.

If the reset parameter is a zero, then the counts accumulate over time. A value of one will reset the counters after every acquisition.

## Data Size

The data size specified in the acquisition table, should be equal to four bytes.

## Data Format

The data acquired is a 32 bit integer value, representing the net counts in the specified direction.

## Type

Synchronous event.

## Comments

None.

# Type 43 (1D256 Counts)

## Description

This acquisition type is used to acquire 1D counts from the 1D256 interface card (sometimes also called 1D Advanced). This card is capable of interfacing with all 1D types probes. It provides a maximum of 256 channels of particle size information and a maximum of 256 channels of particle temporal spacing information. All counters are 32 bits in length.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Size Count | 0x0-F (lower nibble) |
| 1 | Strobe Counts | 0x0-F (upper nibble) |
| 2 | Probe Commands | 0x0-F (lower nibble) |
| 2 | 1D Interface | 0x0-7 (upper nibble) |
| 3 | Source | 0x0-4 (lower nibble) |
| 3 | Divide Factor | 0x0-F (upper nibble) |

**Parameters**

The acquisition parameters for the 1D256 Counts are not set up by the user. The system will copy the parameter information from the associated board table (*See "Board Table Configuration File, (*.brd)"*). We do this to document the way the board/probe is set up and to keep with M200 compatibility.

The lower nibble for parameter one is used to set the number of size channels to be acquired. The number of channels is the value specified in the lower nibble plus one, multiplied by 16.

The upper nibble for parameter one is used to set the number of strobe interval channels to be acquired. The number of strobe interval channels is the value specified in the upper nibble plus one and multiplied by 16. If no interval channels are desired, use a value of 255 for parameter three. Use parameter three to control the strobe interval frequency.

The lower nibble for parameter two is used to store the probe command value.

The upper nibble for parameter two is used as the 1D interface card number. Valid values for 1D interface cards are between zero and seven. This number must be unique and it is assigned in one of

the parameter fields of the acquisition table.The lower nibble for parameter three is used as the source frequency for the strobe interval counter. The range of frequency values to be selected are follows.

| Parameter 3 | Frequency |
|---|---|
| 0 | 4.0 MHZ |
| 1 | 400 KHz |
| 2 | 40 KHz |
| 3 | 4 KHz |
| 4 | 400 Hz |

**Parameters 3**

The upper nibble for parameter three is used as the divide factor for the strobe interval counter. A value of zero, divides the selected frequency by 16, while all other values, divide the frequency by the actual values specified.

Use a value of 255 for parameter three to skip acquisition of interval data.

## Data Size

The data size specified in the acquisition table, should be equal to the number of size channels times four plus the number of interval channels times four.

## Data Format

The data acquired consists of one or two blocks of 32 bit integer data. The first block always has the size counts. The second, optional, block has the interval counts.

## Type

Synchronous event.

## Comments

To obtain the 1D reference voltage use channel zero on the built in Analog to Digital Converter. See 1D256 Analog to Digital Converter acquisition. Please note that the reference voltage is internally divided by two compared to the other channels.

# Type 44 (1D256 Analog Input)

## Description

This acquisition type is used to acquire analog data from the 1D256 interface card (also known as 1D Advanced). There are eight 12 bit analog channels. Channel zero is reserved for the probe reference voltage.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Channel | 0-7 |
| 2 | Range | 1, 2 |
| 3 | Gain | 0-2 |

**Table 5: Parameters**

Parameter one is used as the channel number. It can be a value from zero to seven.

Parameter two is used as the voltage range. A value of zero specifies -5 to +5 voltage range, while a value of one specifies -10 to +10 voltage range.

Parameter three is used as the gain. A value of zero specifies a gain of one, a value of one specifies a gain of 10, and a value of two specifies 100.

***Note: That both the range and gain are set in the 1D256 board by jumpers. The values specified here are used by the data conversion utilities to obtain the right voltage values from the integer counts. When changing ranges you must change both the jumpers and the acquisition parameters.***

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is in two's complement integer coding and represents the signed digital value of the analog signal.

| | | |
|---|---|---|
| 0x7FFF | 32767 | +full scale |
| 0x0000 | 0 | zero |
| 0x8000 | -32767 | -full scale |

**Table 6: Data format**

Notice that 12 bit analog data is being placed into 16 bit integer values. The data is placed in the upper 12 bits, so the data can be treated as a 16 bit signed integer.

## Type

Synchronous event.

## Comments

Note that channel zero is used for 1D reference voltage and is divided by a factor of two prior to conversion. This means a 10 volt reference signal will be recorded as 5 volts. Multiply the 1D reference voltage by a factor of two in the formula table.

# Type 45 (CAMAC VOR Data)

## Description

This acquisition type is used to acquire VOR bearing data.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Slot | 1-23 |
| 2 | Channel | 1-4 |
| 3 | | |

**Parameters**

Parameter one represents the CAMAC slot number and can be a value from one to 23.

Parameter two represents the port number, (VOR/Heading channel) and can be a value between one and four.

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired represents a VOR bearing in the range of zero to 360 degrees. The actual data ranges from 0 to 4096. To convert from raw data to degrees, just multiply the raw value by 360 degrees and divide by 4096 counts.

## Type

Synchronous event.

## Comments

None.

# Type 46 (1D256 Spare 0)

## Description

This acquisition type is for the first of two spare 16 bit counter channels on the 1D256 interface (also known as 1D Advanced). The maximum counting rate is 7 MHZ. It can be used independently of the probe sizing functions.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Mode low byte | 0x00-0xFF |
| 2 | Mode high byte | 0x00-0xFF |
| 3 | | |

**Parameters**

The values for parameter one and two are used to program the mode register for the counter chip. For the regular count mode, use 0x28 for parameter one and 0x03 for parameter two. For other count modes consult with SEA for the appropriate values for these parameters.

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

None.

# Type 47 (1D256 Spare 1)

## Description

This acquisition type is for the second of two spare 16 bit counter channels on the 1D256 interface (also known as 1D Advanced). The maximum counting rate is 7 MHZ. It can be used independently of the probe sizing functions.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Mode low byte | 0x00-0xFF |
| 2 | Mode high byte | 0x00-0xFF |
| 3 | | |

**Parameters**

The values for parameter one and two are used to program the mode register for the counter chip. For the regular count mode, use 0x28 for parameter one and 0x04 for parameter two. For other count modes consult with SEA for the appropriate values for these parameters.

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

None.

# Type 48 (1D256 House Data)

## Description

This acquisition type controls the acquisition of internal probe house keeping data. Eight house keeping channels are recorded. Each channel is 16 bits in length

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | 1D Interface | 0-7 (upper nibble) |
| 3 | | |

**Parameters**

The upper nibble for parameter two is used as the 1D interface card number. Valid values for 1D interface cards are between zero and seven. This number must be unique and it is assigned in one of the parameter fields of the acquisition table.

## Data Size

The data size specified in the acquisition table, should be equal to 16 bytes.

## Data Format

The data acquired is composed of eight 16 bit integer values, representing the counts for the different house channels.

## Type

Synchronous event.

## Comments

None.

# Type 49 (1D256 Activity)

## Description

This acquisition type is used to acquire 1D probe activity data. Activity data is recorded in a 16 bit counter.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

None.

# Type 50 (1D256 Total Strobes)

## Description

This acquisition type records 1D probe total strobe data. Total strobe data is recorded in a 32 bit counter. Total strobe data is the sum of all particles that passed through the beam of the probe, regardless of whether or not the probe actually used all particles in the sizing.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to four bytes.

## Data Format

The data acquired is a 32 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

There are probes for which the total strobes are internally divided by ten. In these cases, the total strobes data must be corrected in the data system (multiply total strobes by ten). The actual total strobes should always be greater than or equal to the total counts.

# Type 51 (1D256 Total Counts)

## Description

This acquisition type records 1D probe total count data. Total count data is recorded in a 32 bit counter. Total count data is the sum of all normal strobes received. It should be equal to the total of all size channels from 0 to the maximum specified for the probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to four bytes.

## Data Format

The data acquired is a 32 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

Total counts represents the total valid particles that the probe could size. The total counts should be equal to the sum of all counts from all channels (also known as total valid counts).

# Type 52 (SDSMT HVPS Image Data)

## Description

This acquisition type acquires image data from the SDSMT High Volume Precipitation Spectrometer via the HVPS interface card.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | DMA Channel | 5, 6, 7 |
| 2 | Channel | 1-4 |
| 3 | Throttle | 1, 2,.. |

**Parameters**

The channel parameter is used to specify the High Volume Precipitation Spectrometer channel. Valid values are 1 through 4.

The throttle byte should be a non zero multiple of the system frequency. It represents the maximum rate at which High Volume Precipitations may be recorded.

## Data Size

The data size should be an even number that represents the maximum number of words to acquire. Common values range from 128 to 4096 bytes.

## Data Format

The data recorded has the same format as specified by the High Volume Precipitation Spectrometer documentation. It is encoded data of the number of pixels to clear and set in a given slice. Timing data and image data are mixed. They are identified by the upper bits in the word.

## Type

Asynchronous master event.

## Comments

None.

# Type 53 (SPEC HVPS Image Data)

## Description

This acquisition type acquires data from the SPEC High Volume Precipitation Spectrometer via the 2D interface card.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Mono Interface | 0-3 |
| 2 | DMA Channel | 5-7 (lower nibble) |
| 2 | Bit Shift Divide | 0-0xF (upper nibble) |
| 3 | Rearm Rate (Hz) | 1, 2,.. |

**Parameters**

The following table shows the possible values for the upper 4 bits of parameter two. Parameter two should be entered as an hexadecimal number, for example 0x07.

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 0 | 16 | 0.250 |
| 1 | 1 | 4.000 |
| 2 | 2 | 2.000 |
| 3 | 3 | 1.333 |
| 4 | 4 | 1.000 |
| 5 | 5 | 0.800 |
| 6 | 6 | 0.667 |
| 7 | 7 | 0.571 |
| 8 | 8 | 0.500 |
| 9 | 9 | 0.444 |
| 0xA | 10 | 0.400 |

**Bit Shift**

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 0xB | 11 | 0.364 |
| 0xC | 12 | 0.333 |
| 0xD | 13 | 0.307 |
| 0xE | 14 | 0.286 |
| 0xF | 15 | 0.267 |

**Bit Shift (Continued)**

The rearm rate, should be a non zero multiple of the system frequency. It represents the maximum rate at which HVPS data will be recorded.

## Data Size

This routine acquires 4096 bytes of data from the HVPS probe.

## Data Format

The data is stored in the native SPEC HVPS compressed data format. Image data, timing and diagnostics information are all stored under this acquisition type.

## Type

Asynchronous master event.

## Comments

The bit shift rate should not be confused with the image strobe clock. The bit shift rate is the rate at which data is shifted out of the probe into the data system. It is constant and is set by the upper nibble of parameter two.

The strobe clock controls the rate at which image slices are shifted into the probe. It varies with true air speed and pixel size. The strobe clock is set by the control function Co2DTAS( ).

The maximum rate that can be used for the bit shift clock depends on the length of the cable between the probe and the data system. A 1 MHZ (divide factor 4) should be adequate for the majority of installations where the cable length is less than 50 feet. If longer cables are used, the user should try slower rates. The most common symptoms of a too high a bit shift rate are image jitter or missing pixels.

# Type 54 (Novatel GPS)

## Description

This acquisition type is used to acquire raw binary data from the Novatel GPS card. Use the number of samples, bytes per sample and rearm parameter to control the amount of data acquired. This acquisition type cannot be used multiple times in the acquisition table to acquire data from more than one interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | Rearm Rate | 1, 2,.. |

**Parameters**

The rearm rate, should be a non zero multiple of the system frequency. It represents the maximum rate at which buffers will be recorded.

## Data Size

The data size varies from data block to data block and buffer to buffer.

## Data Format

The data format, is as specified in the Novatel GPS manual. One or more Novatel GPS data blocks exist in a data buffer.

## Type

Asynchronous master event.

## Comments

None.

# Type 55 (VAX Clock)

## Description

This acquisition type is responsible for acquiring the VAX clock data.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This acquisition type requires four bytes.

## Data Format

The data acquired is an unsigned 32 bit integer, which represents time in milliseconds since mid night.

## Type

Synchronous event.

## Comments

This clock value can be used to synchronize the Model 200 clock with the external VAX clock. Additional setup must be performed in the board table.

# Type 56 (CAMAC 1D256 Counts)

## Description

This acquisition type is used to acquire data from the CAMAC 1D256 advanced interface card. This card is capable of interfacing with all 1D types of probes. It provides a maximum of 256 channels of particle size information and a maximum of 256 channels of particle spacing information. All counters are 32 bits in length.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Size Counts | 0-0xF (lower nibble) |
| 1 | Strobe Counts | 0-0xF (upper nibble) |
| 2 | Probe Command | 0-0xF (lower nibble) |
| 2 | 1D Interface | 0-7 (upper nibble) |
| 3 | CAMAC Slot | 1-23 (lower 7 bits) |
| 3 | Strobes | bit seven |

**Parameters**

The acquisition parameters for the 1D256 Counts are not set up by the user. The system will copy the parameter information from the associated board table (*See "Board Table Configuration File, (*.brd)".* We do this to document the way the board/probe is set up and to keep with M200 compatibility.

The lower nibble for parameter one is used to set the number of size channels to be acquired. The number of channels is the value specified in the lower nibble, increments by one and multiplied by 16.

The upper nibble for parameter one is used to set the number of strobe interval channels to be acquired. The number of strobe interval channels is the value specified in the upper nibble, increments by one and multiplied by 16. If no interval channels are desired, set bit seven for parameter three. Use parameter three to control the strobe interval frequency.

The lower nibble for parameter two is used to store the probe command value.

The upper nibble for parameter two is used as the 1D interface number. Valid values for this parameter are between zero and seven. This value must define a unique 1D interface card, assigned in one of the parameter fields in the acquisition table.

The lower nibble for parameter three is used as the CAMAC slot number.

If bit seven is set, the strobe counts will not be acquired.

## Data Size

The data size specified in the acquisition table, should be equal to the number of size channels times four plus the number of interval channels times four.

## Data Format

The data acquired consists of one or two blocks of 32 bit integer data. The first block always has the size counts. The second, optional, block has the interval counts.

## Type

Synchronous event.

## Comments

None.

# Type 57 (CAMAC 1D256 Reference Voltage)

## Description

This acquisition type is used to acquire the 1D256 reference voltage.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is in two's complement integer coding and represents the unsigned digital value of the reference voltage.

| | | |
|---|---|---|
| 0x7FFF | 32767 | +full scale |
| 0x0000 | 0 | zero |

**Data Format**

Notice that 8 bit analog data is being placed into 16 bit integer values. The data is placed in the upper 8 bits, so the data can be treated as a 16 bit signed integer.

## Type

Synchronous event.

## Comments

None.

# Type 58 (CAMAC 1D256 Spare 0)

## Description

This acquisition type is for the first of two spare 16 bit counter channels on the CAMAC 1D256 interface. The maximum counting rate is 7 MHZ. It can be used independently of the probe sizing functions.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Mode low byte | 0x00-0xFF |
| 2 | Mode high byte | 0x00-0xFF |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

The values for parameter one and two are used to program the mode register for the counter chip. For the regular count mode, use 0x28 for parameter one and 0x03 for parameter two. For other count modes consult with SEA for the appropriate values for these parameters.

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

None.

# Type 59 (CAMAC 1D256 Spare 1)

## Description

This acquisition type is for the second of two spare 16 bit counter channels on the CAMAC 1D256 interface. The maximum counting rate is 7 MHZ. It can be used independently of the probe sizing functions.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Mode low byte | 0x00-0xFF |
| 2 | Mode high byte | 0x00-0xFF |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

The values for parameter one and two are used to program the mode register for the counter chip. For the regular count mode, use 0x28 for parameter one and 0x04 for parameter two. For other count modes consult with SEA for the appropriate values for these parameters.

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

None.

# Type 60 (CAMAC 1D256 House Data)

## Description

This acquisition type controls the acquisition of internal probe house keeping data. Eight house keeping channels are recorded. Each channel is 16 bits in length.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | 1D Interface | 0-7 (upper nibble) |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

Parameter two represents the 1D interface number. Valid values are between zero and seven. This value must be unique and assigned in one of the parameter fields in the acquisition table.

Parameter three represents the CAMAC slot number for the 1D interface card.

## Data Size

The data size specified in the acquisition table, should be equal to 16 bytes.

## Data Format

The data acquired is composed of eight 16 bit integer values, representing the counts for the different house channels.

## Type

Synchronous event.

## Comments

None.

# Type 61 (CAMAC 1D256 Activity)

## Description

This acquisition type is used to acquire 1D probe activity data. Activity data is recorded in a 16 bit counter.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

None.

# Type 62 (CAMAC 1D256 Total Strobes)

## Description

This acquisition type records 1D probe total strobe data. Total strobe data is recorded in a 32 bit counter. Total strobe data is the sum of all particles that passed through the beam of the probe, regardless of whether or not the probe actually used all particles in the sizing.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to four bytes.

## Data Format

The data acquired is a 32 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

There are probes for which the total strobes are internally divided by ten. In these cases, the total strobes data must be corrected in the data system (multiply total strobes by ten). The actual total strobes should always be greater than or equal to the total counts.

# Type 63 (CAMAC 1D256 Total Counts)

## Description

This acquisition type records 1D probe total count data. Total count data is recorded in a 32 bit counter. Total count data is the sum of all normal strobes received. It should be equal to the total of all size channels from 0 to the maximum specified for the probe.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | CAMAC Slot | 1-23 |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to four bytes.

## Data Format

The data acquired is a 32 bit integer value, representing the counter value.

## Type

Synchronous event.

## Comments

Total counts represents the total valid particles that the probe could size. The total counts should be equal to the sum of all counts from all channels (also known as total valid counts).

# Type 64 (1D256 Ballard Counts)

## Description

This acquisition type is used to acquire data from the 1D256 interface card. This card is capable of interfacing with all 1D types probes. It provides a maximum of 256 channels of particle size information and a maximum of 256 channels of particle temporal spacing information. All counters are 32 bits in length.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Size counts | 0-0xF (lower nibble) |
| 1 | Strobe counts | 0-0xF (upper nibble) |
| 2 | Probe command | 0-0xF (lower nibble) |
| 2 | 1D interface | 0-7 (upper nibble) |
| 3 | Source | 0-0xF (lower nibble) |
| 3 | Divide factor | 0-4 (upper nibble) |

**Parameters**

The lower nibble for parameter one is used to set the number of size channels to be acquired. The number of channels is the value specified in the lower nibble plus one, multiplied by 16.

The upper nibble for parameter one is used to set the number of strobe interval channels to be acquired. The number of strobe interval channels is the value specified in the upper nibble plus one and multiplied by 16. If no interval channels are desired, use a value of 255 for parameter three. Use parameter three to control the strobe interval frequency.

The lower nibble for parameter two is used to store the probe command value.

The upper nibble for parameter two is used as the 1D interface card number. Valid values for 1D interface cards are between zero and seven. This number must be unique and it is assigned in one of the parameter fields of the acquisition table.The lower nibble for parameter three is used as the source frequency for the strobe interval counter.

The range of frequency values to be selected are follows.

| Parameter 3 | Frequency |
|---|---|
| 0 | 4.0 MHz |

**Parameter 3**

| Parameter 3 | Frequency |
|---|---|
| 1 | 400 KHz |
| 2 | 40 KHz |
| 3 | 4 KHz |
| 4 | 4 Hz |

**Parameter 3 (Continued)**

The upper nibble for parameter three is used as the divide factor for the strobe interval counter. A value of zero, divides the selected frequency by 16, while all other values, divide the frequency by the actual values specified.

Use a value of 255 for parameter three to skip acquisition of interval data.

## Data Size

The data size specified in the acquisition table, should be equal to the number of size channels times four plus the number of interval channels times four.

## Data Format

The data acquired consists of one or two blocks of 32 bit integer data. The first block always has the size counts. The second, optional, block has the interval counts.

## Type

Synchronous event.

## Comments

To obtain the 1D reference voltage use channel zero on the built in Analog to Digital Converter. See 1D256 Analog to Digital Converter acquisition. Please note that the reference voltage is internally divided by two compared to the other channels.

This acquisition type also writes the low word of every size channel to the BALLARD interface card. The address used for data transfer is C000:0840. Data from channel zero is skipped. The last word is used as the new data counter. This location increments after new data has been placed in the memory.

# Type 65 (Serial Port DC 8 DADS Data)

## Description

This acquisition type, is used to acquire data from the serial port one, via interrupt four. Since the DC 8 DADS serial data is composed of several blocks, the user can specify the identifier for the last data block.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Identifying character | 0-255 |
| 2 | Terminating character | 0-255 |
| 3 | Throttle | 1, 2,.. |

**Parameters**

The Identifying character (Parameter 1) represents the first ASCII character of the last block of DC 8 DADS Serial data. The Terminating character (Parameter 2) represents the last ASCII character of the last block of DC 8 DADS Serial data. If Parameter 1 is set to zero and Parameter 2 to non-zero, the M300 will use Parameter 2 to delimit each block of serial data. If both Parameter 1 and 2 are set to zero, the M300 will use the buffer size to delimit each data block.

The throttle byte should be a non zero multiple of the system frequency. It represents the maximum rate at which DC 8 DADS serial data may be recorded per second. It should be set significantly faster than the actual data rate.

## Data Size

The data size specified in the acquisition table, should be equal to the total number of characters in the DC 8 DADS block, including carriage returns and line feeds.

## Data Format

The data acquired is ASCII data. The data format is not relevant to the acquisition type.

## Type

Asynchronous master event or synchronous event. This data type provides improved flexibility over the M200 version because it supports both Asynchronous and Synchronous events.

If an Asynchronous master event is used the buffer automatically resizes for the number of bytes coming in. This also implies that the bytes per sample are the largest block of data to be collected. This acquisition event must be the first event of an asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

If a Synchronous event is being used the buffer size is not dynamic and must have the bytes per sample set to a user specified value large enough to collect the data. This value is the maximum number of bytes that can be acquired at one time. If the buffer is not filled during acquisition then all remaining bytes will be set to zero.

## Comments

The is a maximum of 4096 characters per block. To do more, you need to request a higher number from SEA. Also, there is a maximum 1 sentence-sample per buffer. This can be used with the serial port, or a serial interface. It can also be used in a synchronous or asynchronous buffer.

# Type 66 (2D Grey Advanced)

## Description

This acquisition type is used to acquire 2D Grey images form a 2D Grey probe, as well as some other 2D Grey data. This acquisition type uses one DMA channel and one interrupt channel per probe / interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 2D Grey interface | 0-3 (b1, b0) |
| 1 | Elapsed time clock select | 0, 4, 8 and 12 (b3, b2) |
| 1 | Interrupt number | 10-12 (b7, b6, b5, b4) |
| 2 | DMA channel | 5-7 (lower nibble) |
| 2 | Bit shift divide | 0-0xF (upper nibble) |
| 3 | Rearm rate (Hz) | 1, 2, ... |

**Parameters**

Parameter 1 is used select the 2D Grey interface number, the elapsed time clock source and the interrupt number. Bits 0 and 1 select the interface number (0, 1, 2 and 3). Bits 2 and 3 select the elapsed time clock source. Bits 4, 5, 6 and 7 select the interrupt number.

The valid clock sources for the elapsed time are described in the following table.

| B3 | B2 | Description |
|---|---|---|
| 0 | 0 | Before version 2.50. 25µs clock fixed. Any data created with version 2.50 and above will not have zeros for these bits. |
| 0 | 1 | Version 2.5 and above. 25µs clock. Default value if b3 and b2 are both zero for version 2.5 and above. |
| 1 | 0 | 2.5µs clock. |
| 1 | 1 | 0.25µs clock. |

**Valid Clock Sources**

Prior to version 2.50 the elapsed counter since start of buffer (SOB) was a 32 bit counter counting TAS clock/256, gated by probe on. With the new version of the software, the elapsed counter SOB is a 32 bit counter counting the selected clock frequency (no gating on probe on). The elapsed counter SOB is always reset at the start of a new buffer.

With the new faster clock frequencies, there is the possibility for a counter overrun to occur. The user should be aware of this situation.

The interrupt channel and the DMA channel should match the switch settings in the interface card. Care should be taken to avoid duplicate use of these channels. The system will not operate properly, and may crash.

The following table shows the possible values for the upper 4 bits of parameter 2. The 2D Grey interface takes two clock cycles to unload one bit from the probe. In fact, the bit shift rate is half of the clock frequency.

| Value | Divide Factor | Frequency (MHz) |
|---|---|---|
| 0 | 16 | 0.250 |
| 1 | 1 | 4.000 |
| 2 | 2 | 2.000 |
| 3 | 3 | 1.333 |
| 4 | 4 | 1.000 |
| 5 | 5 | 0.800 |
| 6 | 6 | 0.667 |
| 7 | 7 | 0.571 |
| 8 | 8 | 0.500 |
| 9 | 9 | 0.444 |
| 0xA | 10 | 0.400 |
| 0xB | 11 | 0.364 |
| 0xC | 12 | 0.333 |
| 0xD | 13 | 0.307 |
| 0xE | 14 | 0.286 |
| 0xF | 15 | 0.267 |

**Bit Shift**

The rearm rate, should be a non zero multiple of the system frequency. It represents the maximum rate at which 2D Grey images will be recorded.

## Data Size

This routine acquires a variable length block of images. The user can select the maximum size of each image by providing the number of bytes per sample desired. The samples field will select the minimum number of images. The final data size will not be larger than the number of samples times the bytes per sample. The total number of images acquired may or may not be larger than the minimum number of images asked for. This depends on the data coming from the probe.

## Data Format

The 2D Grey advanced data is formed by one or more header / image blocks. Slices are 128 bits wide. The header is two slices long. The image data follows each header. The first slice of each image has the 16 bit particle count repeated 8 times. The image data follows the repeated particle count.

Each image header is composed of a 16 bit forward link, a 16 bit reserved field, a 32 bit elapsed time since start of image, a 32 bit elapsed time since start of buffer, a 16 bit slice count, a 16 bit multiply TAS factor, a 16 bit divide TAS factor, a 16 bit minimum pixel field, a 16 bit middle pixel field, a 16 bit maximum pixel field and a 64 bit reserved field.

The forward link represent the image size plus the header size. All other fields in the header block are as described for the regular 2D Grey acquisition types.

The following diagram should illustrate the header format. Each row represents an image slice (16 bytes). With version 2.50 and above the reserved fields are set to zero.

| | | | | | |
|---|---|---|---|---|---|
| **Forward Link (2 bytes)** | **Reserved (2 bytes)** | **Elapsed Time (SOI) (4 bytes)** | **Elapsed Time (SOB) (4 bytes)** | **Slice Count (2 bytes)** | **Multiply TAS Factor (2 bytes)** |
| **Divide TAS Factor (2 bytes)** | **Minimum Pixels (2 bytes)** | **Middle Pixels (2 bytes)** | **Maximum Pixels (2 bytes)** | **Reserved (4 bytes)** | **Reserved (4 bytes)** |

**Header Format**

## Type

Asynchronous master event. This acquisition event, is the only event of a 2D Grey asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

## Comments

The first slice contains a repeated 16 bit particle count value. This value is shifted so that it corresponds with the low byte high byte convention of the Intel data format. For each image slice, the first bit shifted in is stored in the lowest bit of the 128 bit slice and the last bit shifted in is stored in the highest bit of the 128 bit slice. This cause an order inversion of the shadow bit pairs (00=clear, 10=minimum, 01=middle, 11=maximum).

The bit shift rate should not be confused with the image strobe clock. The bit shift rate is the rate at which data is shifted out of the probe into the data system. It is constant and is set by the upper nibble of parameter two.

The strobe clock controls the rate at which image slices are shifted into the probe. It varies with true air speed and pixel size. The strobe clock is set by the control function Co2GTAS.

The maximum rate that can be used for the bit shift clock depends on the length of the cable between the probe and the data system. A 1 MHZ (Divide factor = 4) should be adequate for the majority of installations where the cable length is less than 50 feet. If longer cables are used, the user should try slower rates. The most common symptoms of too high a bit shift rate are image jitter or missing pixels.

# Type 67 (PMS 1058B 1D Data)

## Description

This acquisition type is used to acquire data from the PMS 1058B 1D interface card.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 1D interface | 0-7 |
| 2 | Probe control word | 0-0xFF |
| 3 | Size counts | 0, 1, 3 (upper nibble) |
| 3 | Auxiliary counts | 0-5 (lower nibble) |

**Parameters**

Parameter one is used to set the 1D interface. Valid values are between zero and seven. Each 1D acquisition type must have a different interface number.

Parameter two is used to specify the probe control word. This value specifies the number of size channels, the number of aux channels and the probe range.

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| aux4 | aux3 | aux2 | aux1 | ch1 | ch0 | range1 | range0 |

**Type 67 Channel Selection Values**

For the aux channels, use a one to disable the channel or a zero to enable it. The following table specifies the special case situations.

| Maximum | B7 | Channel |
|---|---|---|
| 64 channels no aux 4 & 5 | 1 | 00 |
| 32 channels no aux 5 | 0 | 01 |
| 16 channels aux 5 enabled | 0 | 10 |
| 16 channels aux 5 disable | 0 | 11 |

**Special Auxiliary Settings**

The upper nibble for parameter three is used as the number of sizes counts to be stored. A value of zero indicates 16 channels, a value of one indicates 32 channels and a value of three indicates 64 channels.

The lower nibble for parameter three is used as number of aux channels to be stored.

## Data Size

The data size specified in the acquisition table, should be equal to the number of size channels times two plus the number of aux channels times two.

## Data Format

The data acquired is composed of size 1 through size n (n less than or equal 32) followed by aux 1 through aux m (m less than or equal 5). The data for each channel is 16 bits wide.

## Type

Synchronous event.

## Comments

None.

# Type 68 (9513 Counters)

## Description

This acquisition type is used to acquire one selected counter from the basic 1D interface card or CYCTM board.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Counter | 1 - 10 |
| 2 | | |
| 3 | | |

**Parameters**

Parameter one is used to select the desired counter. For the 1D Board, Counter 1 selects the strobe, counter 2 selects aux 2, counter 3 selects aux 1, counter 4 selects spare 1 and counter five selects spare 2.

## Data Size

The data size specified in the acquisition table, should be equal to two bytes.

## Data Format

The data acquired is a 16 bit integer value, representing the counter value. If you acquire more than 1 Hz data, make sure you use the Sum() function to add up all values.

## Type

Synchronous event.

## Comments

None.

# Type 69 (BC620AT Time)

## Description

This acquisition type is used to acquire the current time from the BC620AT interface card.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size specified in the acquisition table, should be equal to eight bytes.

## Data Format

The data format is the same as specified by the BC620AT operational and technical manual.

## Type

Synchronous event.

## Comments

None.

# Type 70 (DRV11 Data)

## Description

This acquisition type is used to acquire data from the DRV11 interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Data type | 0-3 |
| 2 | DMA channel | 0-3 |
| 3 | Rearm rate (Hz) | 1, 2, ... |

**Parameters**

The high nibble for parameter one is reserved, and it must be zero. Bit zero of the lower nibble is used to specify data type, (0 integer 16 bits, 1 long integer 32 bits). Bit one is used to specify data swap (0 no swap, 1 swap).

The DMA channel must match the selected DMA channel on the DRV11 interface switches. This DMA channel must be unique to the DRV11 interface or a conflict will arise.

The rearm rate, should be a non zero multiple of the system frequency. It represents the maximum rate at which DRV11 data buffers will be recorded.

## Data Size

The data size must match the number of bytes being transferred by the DRV11 interface.

## Data Format

Data format is variable and defined by the user.

## Type

Asynchronous master event.

## Comments

It is possible to combine long integer and floating point data, since both data types have the same number of bytes. Use the “SERIALIEEE” and “SERIALINTEGER” functions to unpack the data.

# Type 71 (Pressure Multiplexer)

## Description

This acquisition type is used to acquire two bytes of data from the SEA Pressure Multiplexer box.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Box ID | 0-255 |
| 2 | Channel | 0-255 |
| 3 | Gain | 0-3 |

**Parameters**

## Data Size

This routine acquires two 8 bit bytes. Two bytes of data should be allocated for each sample.

## Data Format

The data acquired is in two's complement integer coding and represents the signed digital value of the analog signal.

| | | |
|---|---|---|
| 0x7FFF | 327678 | +full scale |
| 0x0000 | 0 | zero |
| 0x8000 | -32768 | -full scale |

**Data Format**

## Type

Synchronous event.

## Comments

To obtain the correct voltage, multiply the raw count by the voltage range and divide by 65536.

| Voltage | Parameter 3 | Gain | Multiplier |
|---|---|---|---|
| ± 10 | 0 | 1 | 3.051757813E-4 |
| ± 5 | 1 | 2 | 1.525878906E-4 |
| ± 2.5 | 2 | 4 | 7.629394531E-5 |
| ± 1.25 | 3 | 8 | 3.814697266E-5 |

**Voltage Matrix**

The id parameter is the id code of the particular Pressure Multiplexer box where the channel resides. At the present time the first Pressure Multiplexer box is id 16, the second Pressure Multiplexer box is id 17, etc.

You can use the 'Volts' function to convert from raw analog to volts if you set parameter 3 to the appropriate gain value and set the corresponding dip switch inside the Pressure Multiplexer box.

It is recommended that the acquisition events for the Pressure Multiplexer channels be placed at the end of the synchronous buffer in the acquisition table. This allows for maximum efficiency when sampling a variety of acquisition events.

The maximum sample rate for all channels is about 250 Hz.

# Type 72 (INS INI Synchro)

## Description

This acquisition type is used to acquire angular position from a synchro channel in the SEA Inertial Navigation Interface (INI). This adapter supports up to eight synchro to digital converters. These synchro channels can be used to acquire pitch, roll and yaw or other synchro encoded information.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Synchro-channel | 0-7 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires 16 bits of data. Two bytes should be allocated for each sample.

## Data Format

The format of the acquired data is a 14 bit unsigned integer (0 - 16384). This integer represents angles from 0-2π radians.

## Type

Synchronous event.

## Comments

None.

# Type 73 (INS INI Serial)

## Description

This acquisition type acquires binary or BCD ARINC 575 data from one of two boards located inside the SEA Inertial Navigation Interface (INI). This adapter contains an on board RAM which captures the serial data transmitted from the INS system.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | ARINC Label | 0-255 |
| 2 | Card | 0, 1 |
| 3 | | |

**Parameters**

## Data Size

This routine acquires 32 bits of data. Four bytes should be allocated for each sample. The actual data is only 24 bits inside a 32 bit word, with the most significant byte equal to zero.

## Data Format

The first word contains the least significant 16 bits of the ARINC data. The second word contains the most significant 8 bits of the ARINC data. The most significant byte is always zero.

## Type

Synchronous event.

## Comments

None.

# Type 74 (INS INI Flags)

## Description

This acquisition type acquires a group of eight discrete flags from the Litton LTN76 INS through the SEA Inertial Navigation Interface (INI).

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

This routine acquires 16 bits of data. Two bytes should be allocated for each sample. The eight flags are in the low byte, with the high byte set to zero.

## Data Format

Each bit in the low byte represents a flag from the Litton LTN76. Bit zero is the pitch flag (b0), bit one is the primary flag (b1), bit two is the aux flag (b2), bit three is the platform flag (b3), bit four is the roll flag (b4), bit five is the heading flag (b5), bit six is the digital flag (b6), bit seven is the spare flag (b7).

## Type

Synchronous event.

## Comments

None.

# Type 75 (SPP/CDP Data)

## Description

This acquisition type acquires all the binary serial data from the Signal Processing Package for Optical Particle Counters (SPP-100, SPP-200, SPP-300, CDP, CDPPBP). The SEA Serial Interface or Serial Port is used to communicate with the SPP/CDP Probe, by sending the necessary setup and data request commands.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 1D Interface | 0-7 |
| 2 | Command | 0-0xF |
| 3 | | |

**Parameters**

## Data Size

The data size varies with the number of channels acquired. The following table shows the appropriate number of bytes to specify for the different number of channels supported. In addition, the table also show the maximum theoretical sampling rate as well as the maximum suggested sampling rate.

| Channels | Bytes/Sample SPP100, SPP300, CDP | Bytes/Sample SPP200 | Bytes/Sample CDPPBP |
|---|---|---|---|
| 10 | 76 | 74 | 1106 |
| 20 | 116 | 114 | 1146 |
| 30 | 156 | 154 | 1186 |
| 40 | 196 | 194 | 1226 |

**Data Size**

## Data Format

The data format follows the exact description of the binary data retrieved by the SPP/CDP Probe (check the Probe manual). Use the SpData() function to retrieve individual data elements from the SPP/CDP data block.

## Type

Synchronous event.

## Comments

To determine the maximum allowed sample frequency, take the baud rate and divide it by 10. Then divide that by the data size. Your sample frequency has to be less that value.

# Type 76 (CAS Serial Data)

## Description

This acquisition type acquires all the binary serial data from the Cloud Aerosol Spectrometer (CAS) probe. The SEA CAPS Interface is used to communicate with the CAS section of the CAPS probe, by sending the necessary setup and data request commands.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | 1D interface | 0-7 |
| 2 | | |
| 3 | relay control | 0-0xFF |

**Parameters**

The CIP interface number must be different than the CAS interface number. Make sure that the correct interface number for CAS is also used in the CAS board entry.

The replay control parameter is passed to the CAS probe via the data request command. Please check the CAS probe manual for further information.

## Data Size

The data size varies with the number of channels acquired. The following table shows the appropriate number of bytes to specify for the different number of channels supported. In addition, the tables also show the maximum theoretical sampling rate as well as the maximum suggested sampling rate.

| Channels | Bytes/Sample | Maximum Theoretical Rate | Maximum Suggested Rate |
|---|---|---|---|
| 10 | 254 | 22 (Hz) | 20 (Hz) |
| 20 | 294 | 19 (Hz) | 16 (Hz) |
| 30 | 334 | 16 (Hz) | 16 (Hz) |
| 40 | 374 | 14 (Hz) | 10 (Hz) |

**Maximum Frequency**

## Data Format

The data format follows the exact description of the binary data sent by the CAS in response to the request data command (check the CAS manual). Use the CASData() function to retrieve individual data elements from the CAS data block.

## Type

Synchronous event.

## Comments

The CAS setup parameters are controlled through the Board Table. You must have an entry in the Board Table for the CAPS and CAS.

# Type 77 (CIP Serial Data)

## Description

This acquisition type acquires all the binary serial data from the Cloud Imaging Probe (CIP). The SEA CAPS interface is used to communicate with the CIP, by sending the necessary setup and data request commands.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | interface | 0-3 |
| 2 | | |
| 3 | | |

**Parameters**

In the M300 after (11/27/06), these parameters have status information for the acquisition.

Parameter 1 (setup = 1, data = 2). Parameter 2 (bad checksum = 1, invalid data size = 2, missing ack = 3, reset flat = 4).

## Data Size

The date size for the CIP acquisition event must be 180 bytes. The maximum theoretical acquisition frequency for CIP is 31 Hz. The maximum suggested acquisition frequency is 25 Hz.

## Data Format

The data format follows the exact description of the binary sent by the CIP in response to the data request command (check the CIP probe manual). Use the CIPData() function to retrieve the individual data elements from the CIP data block.

## Type

Synchronous event.

## Comments

None.

# Type 78 (CIP Image Data)

## Description

This acquisition type acquires the Cloud Imaging Probe (CIP) image. The SEA CAPS interface is used to communicate with the CIP.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | DMA | 5, 6, 7 |
| 3 | Rearm rate (Hz) | 1, 2, ... |

**Parameters**

## Data Size

The data size for the CIP image acquisition event must be 4098 bytes (4096 for image plus 2 for check sum.

## Data Format

The data format follows the exact description of the compressed image data from the CIP probe (check the CIP manual). Use the cip.300 file to display the CIP image data.

## Type

Asynchronous master event. This acquisition event, is the only event of a CIP asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

## Comments

The CIP setup parameters are controlled through the Board Table. You must have an entry in the Board Table for CAPS and CIP. The true air speed is controlled by the request data command for the synchronous CIP serial data. In order to acquire the CIP image properly, you must also have an entry in the acquisition table for the CIP serial data. There also must be an entry in the control table, to control the TAS to the CIP probe.

# Type 79 (CAS PBP Data)

## Description

This acquisition type acquires data from the Cloud Aerosol Spectrometer (CAS) probe, Particle by Particle (PBP). The SEA CAPS, Serial Port, or Serial interface can be used to interface to the CAS PBP.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size is dependent on the bytes per sample. The maximum data size is 1025 bytes.

## Data Format

The data format follows the exact description of the binary data sent by the CAS in response to the request data command (check the CAS PBP manual). Use the CasPbpData() function to retrieve individual data elements from the CAS PBP data block.

## Type

Synchronous event.

## Comments

The CAS PBP setup parameters are controlled through the Board Table. You must have an entry in the Board Table for the CAS PBP Board.

# Type 80 (Ballard 708 Data)

## Description

This acquisition type acquires data from the Ballard 708 interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size is 200 bytes per sample. This matches the specification for ARINC 708. Check the ARINC 708 specification for more information.

## Data Format

The data format follows the exact description of the ARINC 708 specification. Use the Ballard708Data() function to retrieve individual data elements from the data block.

## Type

Asynchronous master event.

## Comments

None.

# Type 81 (Serial Port Tamdar Data)

## Description

This acquisition type acquires data from the Tamdar.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | stx | 0-255 |
| 2 | etx | 0-255 |
| 3 | type | 0x02 |

**Parameters**

Parameter 1 is used for start of text (stx, usually equal 0x02). Parameter 2 is used for end of text (etx, usually equal to 0x03). Parameter 3 is used by the M300 to document the sentence type. Type 0x02 data is supported.

## Data Size

The data size is 27 bytes per sample. This matches the specification for Tamdar data. Check the Tamdar specification for more information.

## Data Format

The data format follows the exact description of the Tamdar specification. Use the TamdarData() function to retrieve individual data elements from the data block. Use the correct trigger prior to the Tamdar data function.

## Type

Asynchronous master event.

## Comments

None.

# Type 82 (Serial Port AIMMS Data)

## Description

This acquisition type acquires data from the AIMMS20, AIMMS10 and ADP.
With version 1.11.08 of the M300 this acquisition type was must be used with AIMMS board.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | type | 0-1 |
| 2 | samples | 1-255 |
| 3 | id | 0, 1, 2, 4, 5, 11, 12, 22, 28, 29 |

**Parameters**

Parameter 1 is the data type (0 for AIMMS20/ADP and 1 for AIMMS10).

Parameter 2 is the number of desired samples (since the number of samples in the acquisition entry must be 1). For normal AIMMS20 data, parameter 2 is a 1 (one sample per buffer). For the AIMMS20 ID22 we can acquire from 1-255 (maximum number of AIMMS20 ID22 samples/ sentences to acquire in the M300 buffer). If you look in the raw data buffer, parameter 2 will have the actual number of samples acquired. Since the buffer size is a limiting factor for the AIMMS20 ID22 data, sometimes the number of samples acquired is less than the number requested. When the buffer is full we need to terminate and start a new one.

Parameter 3 is set by the M300 to specify sentence id.

## Data Size

The data size is varies depending on the sentence type.

## Data Format

The data format follows the exact description of the AIMMS specification. Use the AIMMSData() function to retrieve individual data elements from the data block. To process AIMMS data first setup a general trigger for the AIMMS data (and specific board) to get the different sentence id formula triggers (using DirData() function). Then use the same trigger plus the different formula triggers for each different data block (*See "AIMMSData(), AIMMS Data Access"*). There is a sample of this in the /test/aimms20 project.

## Type

Asynchronous master event.

## Comments

To perform a purge on the AIMMS/ADP (*See "AIMMS Commands"*).

# Type 83 (Network POSAV Data)

## Description

This acquisition type acquires data from the POSAV.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

Varies depending on the data. The largest size supported is 1024. This is limited by the MTU size of about (1500 bytes).

## Data Format

The data format follows the exact description of group 1 for the POSAV data. Use the PosAvData() function to get the data.

## Type

Asynchronous master event.

## Comments

None.

# Type 84 (Network ASCII Data)

## Description

This acquisition type acquires ASCII data from a socket.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | block | 0-255 |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

Varies depending on the data. The largest size supported is 1024. This is limited by the MTU size of about (1500 bytes).

## Data Format

Varies depending on the data. Use the appropriate trigger to get the data in the formula table. Also use the serial ASCII data functions to get the data.

## Type

Asynchronous master event.

## Comments

None.

# Type 85 (Network Binary Data)

## Description

This acquisition type acquires binary data from a socket.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | match | 0-1 |
| 2 | | |
| 3 | | |

**Parameters**

Parameter 1 is used to match (value of 1) the size of the data. If the data read matches the data size the data is accepted. Otherwise the data is ignored. If match is zero, then all data read is returned.

## Data Size

The data size is variable depending on the data. The largest size supported is 1024. This is limited by the MTU size of about (1500 bytes).

## Data Format

The data format varies. To access data from this acquisition type you might want to use the GetData() function.

## Type

Asynchronous master event.

## Comments

None.

# Type 86 (CIPGS Serial Data)

## Description

This acquisition type acquires all the binary serial data from the Cloud Imaging Probe Grey Scale (CIPGS). The SEA CAPS interface is used to communicate with the CIPGS, by sending the necessary setup and data request commands.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

In the M300 these parameters have status information for the acquisition.

Parameter 1 (setup = 1, data = 2, setup grey scale = 3). Parameter 2 (bad checksum = 1, invalid data size = 2, missing ack = 3, reset flat = 4).

## Data Size

The date size for the CIPGS acquisition event must be 180 bytes. The maximum theoretical acquisition frequency for CIPGS is 31 Hz. The maximum suggested acquisition frequency is 25 Hz.

## Data Format

The data format follows the exact description of the binary sent by the CIPGS in response to the data request command (check the CIPGS probe manual). Use the CIPGSData() function to retrieve the individual data elements from the CIPGS data block.

## Type

Synchronous event.

## Comments

None.

# Type 87 (CIPGS Image Data)

## Description

This acquisition type acquires the Cloud Imaging Probe Grey Scale (CIPGS) image. The SEA CAPS interface is used to communicate with the CIPGS.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size for the CIPGS image acquisition event must be 4098 bytes (4096 for image plus 2 for check sum.

## Data Format

The data format follows the exact description of the compressed image data from the CIPGS probe (check the CIPGS manual). Use the cgs.300 file to display the CIPGS image data.

## Type

Asynchronous master event. This acquisition event, is the only event of a CIPGA asynchronous buffer in the acquisition table. The buffer number should be the next non zero integer increment of the highest buffer number used so far.

## Comments

The CIPGS setup parameters are controlled through the Board Table. You must have an entry in the Board Table for CAPS and CIPGS. The true air speed is controlled by the request data command for the synchronous CIPGS serial data. In order to acquire the CIPGS image properly, you must also have an entry in the acquisition table for the CIPGS serial data. There also must be an entry in the control table, to control the TAS to the CIPGS probe.

# Type 88 (CIPGS Info Data)

## Description

This acquisition type acquires all the binary serial data from the Cloud Imaging Probe Grey Scale (CIPGS). The SEA CAPS interface is used to communicate with the CIPGS, by sending the necessary setup and data request commands.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The date size for the CIPGS Info acquisition event must be 190 bytes. The frequency should be 1 hz. There is no reason to acquire this data any faster. At start up the probe sends this information. After this these values never change until we re-send the setup command to the CIPGS.

## Data Format

The data format follows the exact description of the binary sent by the CIPGS in response to the Grey Scale setup reply command (check the CIPGS probe manual). Use the CIPGSInfo() function to retrieve the individual data elements from the CIPGS data block.

## Type

Synchronous event.

## Comments

None.

# Type 89 (Serial Binary Data)

## Description

This acquisition type acquires binary data from a serial port or SEA serial interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | match | 0-1 |
| 2 | | |
| 3 | | |

**Parameters**

Parameter 1 is used to match (value of 1) the size of the data. If the data read matches the data size the data is accepted. Otherwise the data is ignored. If match is zero, then all data read is returned.

## Data Size

The data size is variable depending on the data.

## Data Format

The data format varies.

## Type

Asynchronous master event.

## Comments

None.

# Type 90 (Network Binary Buffered Data)

## Description

This acquisition type acquires binary data from a socket. Typical network data is less than 1024 bytes (MTU). Data for this acquisition type will be buffered up to the data size selected.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | expire (s) | 0-255 |
| 2 | | |
| 3 | | |

**Parameters**

Parameter 1 is used as number of seconds to expired data buffer. A zero will not expire buffers based on time (buffered data returned up to data size requested). A non-zero value will return all the data buffered so far up to the selected number of seconds.

## Data Size

The data size is variable depending on the data.

## Data Format

The data format varies.

## Type

Asynchronous master event.

## Comments

None.

# Type 91 (ARINC429 FIFO Data)

## Description

This acquisition type is used to acquire FIFO style data labels from the ARINC429 interface. This requires new firmware version on the interface card. This can used with Stormscope instrument.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Start Label | 0 to 255 |
| 2 | Mode | 0x80, 0x81, 0, 1 |
| 3 | End Label | 0 to 255 |

**Parameters**

We use parameter 1 for the start label in stream mode.

Parameter 2 control the stream mode and normal fifo mode as well as the receiver channel into the ARINC429 interface. Stream mode is achieved with setting bit 7 to 1 (0x80). Bit 0 control the receiver channel.

Parameter 3 can be used for the end label in stream mode.

## Data Size

The data size is variable and depends on the specific instrument connected.

## Data Format

See the documentation for the instrument in question for a better idea about the data format. This is going to be some sort of ARINC 429 data. If we are doing stream mode all the data from the start label to the end label should be acquired.

## Type

Asynchronous master event.

## Comments

None.

# Type 92 (CAS DPOL Data)

## Description

This acquisition type acquires data from the CAS DPOL serial probe. This probe combines CAS and PBP packets all into a high speed serial channel.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size is variable depending on the number of PBP packets selected and the number of data channels (usually 30). The best way to set the data size is to let the M300 pick the size based on the number of histogram channel and the particle by particle packets.

## Data Format

With regards to the data format see the CAS DPOL probe manual as well as the interface manual. We kept the data format from the CAS DPOL probe intact from what the instrument provides.

## Type

Asynchronous master event.

## Comments

The current baud rate for the CAS DPOL board is 460800 baud. With the PBP turned on this probe is capable of sending out a maximum of about 46080 bytes per second at this baud rate.

Since this is an asynchronous master event the sample frequency must be one. Then select the number of PBP packets in the board setup for CAS DPOL and setup the buffer life/frequency at a number close to the system frequency. Since the CAS DPOL probe has a high baud rate and his capable of a large data size the goal for the CAS DPOL acquisition event is to spread out the acquisition over the sampling period.

The theoretical maximum number of PBP packets per second is less than 900.

# Type 100 (PIRAQ I, Q and P)

## Description

This acquisition type is used to acquire I, Q and P data from the PIRAQ interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size is specified in bytes. This acquisition event can have a variable number for the data size. The valid data size for this acquisition event must be equal to the number of gates * 4 * 3. See data format for an explanation.

## Data Format

This acquisition event stores the I, Q and P data for every gate. The I, Q and P are converted from C40 float format to IEEE float format (4 bytes) and stored as the IEEE float format.

The I and Q data, represent the real and imaginary components of the average return in vector form. They are also known as the coherent power.

The P data, represents the incoherent power summed over the gate width and number of hits.

The following table helps visualizing the actual data stored by the M300. The top row and left column are not part of the data.

| Gates (*n*) | I | Q | P |
|---|---|---|---|
| **0** | $I_0$ | $Q_0$ | $P_0$ |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| **n-1** | $I_{n-1}$ | $Q_{n-1}$ | $P_{n-1}$ |

**Data format structure**

## Type

Asynchronous master event.

## Comments

None.

# Type 101 (PIRAQ Config)

## Description

This acquisition type is used to acquire configuration data from the PIRAQ interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

324 bytes.

## Data Format

This acquisition event stores the Piraq configuration data. The following C structure is used to specify the data. Please refer to the SEA data types section for information on the different data types used.

```
typedef struct PqConfig {
 lword timingMode;
 lword delay;
 lword gates;
 lword hits;
 lword gateWidth;
 lword pulseWidth;
 lword pulseRepetitionTime;
 lword watchdog;
 lword firstGateMode;
 lword phaseCorrectMode;
 lword clutterFilterMode;
 lword timeSeriesMode;
 lword timeSeriesGate;
 float scanRate;
 float pulseRate;
 float indexOfRefraction;
 float beamWidth;
 ubyte dsp[256];
} PqConfig;
```

## Type

Asynchronous slave event.

## Comments

None.

# Type 102 (PIRAQ Status)

## Description

This acquisition type is used to acquire status data from the PIRAQ interface.

## Description

This acquisition type is used to acquire status data from the PIRAQ interface.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

76 bytes.

## Data Format

This acquisition event stores the Piraq Status data. The following C structure is used to specify the data. Please refer to the SEA data types section for information on the different data types used.

```
typedef struct PqStatus {
 lword newBuffer;
 lword gates;
 lword hits;
 lword gateWidth;
 lword firstGateMode;
 lword phaseCorrectMode;
 lword clutterFilterMode;
 lword timeSeriesMode;
 lword timeSeriesGate;
 lword numeratorDiscriminator;
 lword denominatorDiscriminator;
 lword firstGateInverseMagnitude;
 lword spare1;
 lword spare2;
 lword spare3;
 lword spare4;
 lword status;
```

```
    lword pulseWidth;
    lword dataFormat;
  } PqStatus;
```

## Type

Asynchronous slave event.

## Comments

None.

# Type 250 (Status Info Data)

## Description

This acquisition type is reserved for storing any internal data values from the M300. These are usually for used for testing.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | type | 0-255 |
| 2 | sub type | 0-255 |
| 3 | | |

**Parameters**

| Description | Type | Sub Type |
|---|---|---|
| irq monitor | 0 | 0 |
| irq latency * 0.25 = 1 µs | 1 | 0 |
| irq duration * 0.25 = 1 µs | 2 | 0 |
| proxy latency * 0.25 = 1 µs | 3 | 0 |
| proxy duration * 0.25 = 1 µs | 4 | 0 |
| M300 Time (s) | 16 | 0 |
| M300 Time (ns) | 16 | 1 |
| QNX Time (s) | 17 | 0 |
| QNX Time (ns) | 17 | 1 |
| BC635/BC637 Time (s) | 18 | 0 |
| BC635/BC637 Time (ns) | 18 | 1 |

**Select Data**

## Data Size

The data size is four bytes.

## Data Format

See the data type select table above.

## Type

Synchronous event.

## Comments

This acquisition events must be added under the system board.

# Type 251 (Command Data)

## Description

This acquisition type is reserved for storing any command input to the M300 during the acquisition process. When a user inputs a command using the command manager, the M300 will then act on that command and subsequently store it into the acquisition data for later playback. Note that any string entered will be stored, including invalid commands. This allows the user to enter comments during an acquisition run. Note also that the command acquisition only works for acquisition mode and is not supported by the playback or UDP modes.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size varies depending on the length of the command. Commands are terminated with one or two zeroes depending on which number makes the length even.

## Data Format

ASCII data terminated by zeroes (C-style).

## Type

Asynchronous event (Not used in the acquisition table).

## Comments

This acquisition type describes the command data format. This cannot be used in the acquisition table. To turn command storage on or off, turn the command buffer (buffer 251) on or off respectively.

# Type 252 (Error Data)

## Description

This acquisition type is reserved for storing error information/messages into the acquired data. Once the error occurs, the M300 will sequence the error information into the data being acquired at the time. For most errors, M300 generates an error message. Certain errors might have a high frequency of occurrence, and the system keeps an error count and will generate that error message only once per second. If there are error messages, they must be corrected for proper operation.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size varies depending on the length of the command. Commands are terminated with one or two zeroes depending on which number makes the length even.

## Data Format

ASCII data terminated by zeroes (C-style).

## Type

Asynchronous event (Not used in the acquisition table).

## Comments

This acquisition type describes the error data format. This cannot be used in the acquisition table. To turn error storage on or off, turn the error buffer (buffer 252) on or off respectively.

# Type 253 (Telemetric Data)

## Description

This acquisition type is reserved for the telemetric data. Telemetric data is similar to secondary acquisition data. Telemetric data is acquired via a remote machine connected to the data acquisition system. The format for telemetric data is the same as for secondary acquisition data.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Type | 0, 1, 2 |
| 2 | | |
| 3 | | |

**Parameters**

Parameter one is used to specify the data type. For integer and long integer data the type is zero. For float data the type is one. For character data the type is two.

## Data Size

The data size varies depending on the data source. The bytes per sample is four bytes for floats, four bytes for long integers, two bytes for integers and one byte for characters.

## Data Format

The data format varies accordingly with the data source (float, integer, long, character).

## Type

Synchronous event.

## Comments

Note that float data and long integer data have the same size (4 bytes), but different values for parameter one. Also, integer and long integer data have the same value for parameter one, but the data size is two for integer data and four for long integer data.

# Type 254 (Secondary Acquisition)

## Description

This acquisition type is reserved for the secondary acquisition data. Secondary acquisition data is derived data, from the raw data formulas. The data source for the secondary acquisition data must come from one of the formulas in the formula table.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | Type | 0-9 |
| 2 | | |
| 3 | | |

**Parameters**

The following table describes the acceptable values for parameter 1 it's corresponding secondary acquisition type. These values are different from the M200 values. Be sure to check before using.

| Value | Type | Range |
|---|---|---|
| 1 | String (Variable bytes) | N/A |
| 2 | Character (1 byte) | [-128, 127] |
| 3 | Unsigned Character (1-byte) | [0, 255] |
| 4 | Integer (2-bytes) | [-32,768, 32,767] |
| 5 | Unsigned Integer (2-bytes) | [0, 65,535] |
| 6 | Long Integer (4-bytes) | [-2,147,483,648, 2,147,483,647] |
| 7 | Unsigned Long Integer (4-bytes) | [0, 4,294,967,295] |
| 8 | Float (4-bytes) | $\pm[1.17\times10^{-38}, 3.40\times10^{38}]$ |
| 9 | Double Precision Float (8-bytes) | $\pm[2.22\times10^{-308}, 1.79\times10^{308}]$ |

**Parameter 1**

## Data Size

The data size varies depending on the data source. The bytes per sample is eight bytes for doubles, four bytes for floats, four bytes for long integers, two bytes for integers and one byte for characters.

## Data Format

The data format varies accordingly with the data source (float, integer, long, character).

## Type

Asynchronous event (Not used in the acquisition table).

## Comments

To get the correct data, you must look at Parameter 1, and cast the data appropriately.

# Type 255 (Tables Data)

## Description

This acquisition type is reserved for the storage of all M300 configuration tables into the acquisition data file.

## Parameters

| Parameter | Usage | Limits |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |

**Parameters**

## Data Size

The data size varies as the length of the M300 tables vary.

## Data Format

ASCII data terminated by zeroes (C-style).

## Type

Asynchronous event (Not used in the acquisition table).

## Comments

This acquisition type describes the tables data format. This cannot be used in the acquisition table. To turn table storage on or off, turn the table buffer (buffer 255) on or off respectively.

# Function Reference

Functions are routines, which perform specific operations, so that they may be used over and over again. Each function needs a certain number of parameters (factors) to act on. These parameters are specified inside parenthesis, separated by commas. All functions return the result of the operation to the floating-point stack, allowing them to be used as an operand to a stack operation. Parameters to functions cannot be any kind of operation or other functions.

***Note: As part of the improvement process, engineers at SEA have updated and improved functions which were part of the M200 DAS. One of these improvements is a new function naming convention. For example, the initials "Sr" prefix all new M300 serial data related functions (i.e. SerialAscii() is now SrAscii()). This new convention was developed to help standardize function names; however, most of the older M200 functions are 100% compatible with the M300. Although not necessary for operation, users are encouraged to use the new M300 functions in place of the older M200 functions. Functions that have been updated with a new M300 replacement are indicated by the term 'Deprecated' in the function Synopsis. This is to indicate to the user that SEA highly recommends use of the new function (a link to the new function is included in the deprecation statement).***

The following is a list of the functions presently available.

| Function Name | Function Description | Page |
|---|---|---|
| Accumulate() | Accumulates array elements | 203 |
| Add() | Add two arrays of formulas or individual elements | 204 |
| AIMMSData() | Access AIMMS data | 205 |
| Alarm() | Timer alarm function | 211 |
| AltP() | Inverse Pressure Altitude function | 212 |
| Areas() | Sums to areas array | 213 |
| Arinc429Out() | ARINC 429 data output | 214 |
| Arinc708Data() | ARINC 708 data access function | 215 |
| Array() | Set up a value in an array | 217 |
| AsyncData() | Extract data from an asynchronous buffer | 218 |
| Average() | Average a value for a period of time | 219 |
| Avg() | Compute an average from an array of values | 220 |
| Bearing() | Calculates aircraft bearing | 221 |
| BufferTime() | Return buffer time | 222 |

**Table 7: M300 Function Reference**

| Function Name | Function Description | Page |
|---|---|---|
| CArray() | Extract an element from a character (string) array | 223 |
| CasData() | CAS data access function | 224 |
| CASDPOL() | CAS DPOL PBP fuction | 224 |
| CASDPOLData() | CAS DPOL data access function | 229 |
| CASDPOLSums() | CAS DPOL sums function | 230 |
| CASPBPData() | CAS PBP (Particle-by-Particle) data access function | 231 |
| CIndex() | Character Index function | 232 |
| CIPData() | CIP data access function | 233 |
| CIPGSData() | CIPGS data access function | 235 |
| CIPGSInfo() | CIPGS info data access function | 237 |
| CIPGSMonitor | CIPGS monitor function | 238 |
| CIPGSSums | GIPGS sums function | 239 |
| CIPMonitor | CIP monitor function | 240 |
| CIPSums | CIP sums function | 241 |
| Cmd1D() | Command from 1D data | 242 |
| Co1DCmd | Control 1D Command Function | 243 |
| Co2DTAS() | Control 2D Probe function | 244 |
| Co2GCmd() | Control 2G Probe Command | 245 |
| Co2GTAS() | Control 2G Probe TAS | 246 |
| CoATDAQ141XDA() | Control ATDAQ141X D/A Board | 247 |
| CoCIPGSTAS | Control TAS to CIPGS Probe | 248 |
| CoCIPTAS | Control TAS to CIP Probe | 249 |
| CoCYDDA() | Control CYDDA (D/A) | 250 |
| CoDo() | Control Digtal Board | 251 |
| CoDT2817 | Control DT2817 digital I/O | 252 |
| CoFile | Controls M300 recording | 253 |
| Color | Returns a color value. | 254 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| Comb() | Combine two formula arrays | 255 |
| Concs() | Compute concentrations | 256 |
| CoPCIDACDA | Control PCIDAC D/A voltages | 258 |
| CoPMFDA | Control PMF D/A voltages | 259 |
| Copy() | Copy data from formula | 260 |
| CoQuit | Controls M300 termination | 261 |
| CoRTI802 | Controls RTI802 D/A voltages | 262 |
| CoSeaDA | Coltrols SEA D/A voltages | 263 |
| CoShutDown | Controls M300 System (QNX) Shutdown | 264 |
| CountBy | Count By Bins | 265 |
| CountEdges | Count Edges | 266 |
| Cumulative | Cumulative | 267 |
| Date() | Creates a string based on the current date | 268 |
| DateTime() | Creates a string based on the current date and time | 269 |
| DayOfYear | Returns day of year | 270 |
| Delay() | Delay for a formula for a specified period of time | 271 |
| Delta() | Delta value for a formula (based on time) | 272 |
| DewPointToRH() | Converts Dew Point temperature to Relative Humidity | 273 |
| DFault() | Default value | 274 |
| DIndex() | Double Index function | 275 |
| DirData() | Gets specfic data parameters from an acquisition directory | 276 |
| Div() | Returns the quotient two arrays of formulas | 277 |
| DToF() | Converts double data to float type | 278 |
| Eq() | Boolean comparison for equality | 279 |
| Esi() | Vapor pressure of water with respect to ice | 280 |
| Esw() | Vapor pressure of water with respect to water | 281 |
| EvtStr() | Event String | 282 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| EvtVal() | Event Value | 283 |
| FalconData() | Extracts Falcon Data | 284 |
| FalconDay() | Extracts Falcon Day data | 285 |
| FalconTime() | Extracts Falcon Time data | 286 |
| FArray() | Extracts data from a floating point type array | 287 |
| FIndex() | Extracts data from a floating point type array | 288 |
| Ge() | Boolean comparison for greater than or equal to | 289 |
| GetData() | Get Data | 290 |
| GrData() | 2D Grey access function | 291 |
| GrSums() | 2D Grey Sums function | 293 |
| Gt() | Boolean comparison for greater than | 296 |
| HSAnalog() | Converts an array of analog values to a floating point value | 297 |
| HvMask | Get the HVPS Mask data | 298 |
| HvpsMask() | Get the HVPS Mask data | 299 |
| HvpsTiming() | Get the HVPS Timing data | 300 |
| HvSums | Sums up channels of HVPS data | 301 |
| HvTiming | Gets the HVPS Timing data | 302 |
| IArray() | Extracts data from an integer (2-bytes) type array | 303 |
| IasP() | Inverse Pressure Indicated Airspeed function | 304 |
| IIndex() | Extracts data from an integer (2-bytes) type array | 305 |
| Incloud() | In cloud prediction function | 306 |
| Ins429Bin() | Extracts INS 429 BIN data | 307 |
| InsBCD() | INS BCD data to -1...+1 | 308 |
| InsBin() | INS BIN data to -1...+1 | 309 |
| InsBin2() | P3 INS BIN data to -1...+1 | 310 |
| InsPos() | Converts INS BCD lattitude and longitude to radians | 311 |
| IntegerData() | Gets integer data | 312 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| Intercept() | Calculates the point at which a line intersects the y-axis | 313 |
| IR() | In Range check | 314 |
| IVar1D() | Calculates inverse velocity acceptance ratio (1D) | 315 |
| Ivar1DAdv() | Calculates inverse velocity acceptance ratio (1D advanced) | 316 |
| KeyIndex() | Manages a sorted array of values | 317 |
| LArray() | Extracts data from an long integer (4-bytes) type array | 318 |
| LatStr() | Creates a string containing latitude | 319 |
| Le() | Boolean comparison for less than or equal to | 320 |
| LeastSqReg | Least square regression | 321 |
| Limit() | Limit data value | 322 |
| LIndex() | Extracts data from a long type array | 323 |
| LonStr() | Creates a string containing longittude | 324 |
| Lookup() | Returns the lookup interpolation value | 325 |
| LookupGet() | Get values from lookup entry | 326 |
| LookupSet() | Set values from lookup entry | 327 |
| LrnPos() | Converts LORAN/GPS latitude and longitiude to radians | 328 |
| Lt() | Boolean comparison for less than | 329 |
| LToF() | Converts long integer data to float type | 330 |
| Masses() | Compute masses | 331 |
| Max() | Finds the maximum value from an array of values | 332 |
| MaxSiz() | Returns the maximum size | 333 |
| MaxTim() | Returns the time for the maximum value | 334 |
| MaxVal() | Returns the maximum value from a given period of time | 335 |
| Mean() | Calculates mean X value given X and Y arrays | 336 |
| Median() | Calculates median X value given X and Y arrays | 337 |
| Min() | Returns the minimum value from an array of values | 338 |
| MinSiz() | Returns the minimum size | 339 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| MinTim() | Returns the time for the minimum value | 340 |
| MinVal() | Returns the minimum value from a given period of time | 341 |
| Mode() | Returns mode X value given X and Y arrays | 342 |
| MoSums() | 2D Mono Sums | 343 |
| Mul() | Returns the product of two arrays of formulas | 344 |
| Nmea() | NMEA Sentence Function | 345 |
| OdCmd() | 1D Command | 347 |
| OdIVar() | Inverse Velocity Acceptance Ratio | 348 |
| OdIVarAdv() | 1D Advanced Inverse velocity Accept Ratio | 349 |
| OdRef() | 1D Refference voltage | 350 |
| OdSums() | 1D Sums | 351 |
| PAlt() | Calculates pressure altitude | 352 |
| PIas() | Calculates pressure indicated airspeed | 353 |
| Poly() | Polynomial computation | 354 |
| PosAvData() | POSAV data access | 355 |
| Power() | Power ($x^y$) for arrays | 357 |
| PqConfig() | Piraq Config access | 358 |
| PqPower() | Compute Piraw Power | 359 |
| PqRange() | Compute Piraq Range | 360 |
| PqRaw | Compute Piraq Raw Data | 361 |
| PqReflectivity() | Compute Piraq Reflectivity | 362 |
| PqStatus() | Piraq status data access | 363 |
| PrData() | Extracts data from a probe table | 364 |
| ProbeData() | Extracts data from a probe table | 365 |
| PromoBins() | Compute Promo bins | 366 |
| PromoData() | Promo data access | 367 |
| Protect() | Protect value | 368 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| PrTasClockIn() | Probe TAS Clock in | 369 |
| PrTasClockOut() | Probe TAS Clock out | 370 |
| PTas() | Caluclates pressure true air speed | 371 |
| RaConstant() | Computes radar constant | 372 |
| Rand() | Random number generator | 373 |
| RandData() | Random data generator | 374 |
| RandSeed | Set random seed | 375 |
| Range() | Calculates aircraft range | 376 |
| Ref1D() | 1D Reference volatge | 377 |
| RHToDewPoint() | Converts Relative Humidity to Dew Point temperature | 378 |
| Scale() | Scaling function (first order) arrays | 379 |
| Scale2 | Scaling function (second order) arrays | 380 |
| Scale3 | Scaling function (third order) arrays | 381 |
| ScaleArray() | Scale array function (first order) | 382 |
| ScaleArray2() | Scale array function (second order) | 383 |
| ScaleArray3() | Scale array function (third order) | 384 |
| Seconds() | Returns seconds since Jan 1st 1970 | 385 |
| SerialASCII() | Get Serial ASCII data | 386 |
| SerialDADS() | Get Serial DADS data | 387 |
| SerialIEEE() | Get Serial IEEE data | 388 |
| SerialInteger() | Get Serial Integer data | 389 |
| SrNmea() | Get Serial NMEA data | 404 |
| SerialVAX() | Get Serial VAX data | 390 |
| Set() | Sets formula value | 391 |
| Sizes() | Returns the channel sizes from a probe table | 392 |
| Skip() | Skips to a particular point in the formula table | 393 |
| Slope | Return slope of a line | 394 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| SpData() | Access SPP100, SPP200, SPP300, CDP, CDPPBP data | 395 |
| Spp100Data() | Accesses SPP100 data | 397 |
| SrASCII() | Gets tokens from serial ASCII data | 399 |
| SrDADS() | Gets DC 8 DADS Serial data | 400 |
| SrData | Gets serial data | 401 |
| SrIEEE() | Gets IEEE data | 402 |
| SrInteger() | Gets integer data | 403 |
| SrVAX() | Gets float values from micro VAX data | 406 |
| StDev | Standard Deviation | 407 |
| STemp() | Gets static temperature | 408 |
| Stormscope | Get latitude and longitude pairs | 409 |
| StrCat() | String Concatenate | 410 |
| StrCmp() | Boolean comparison of two strings for equality | 411 |
| StrCpy | String Copy | 412 |
| StrParameters | String Parameters | 413 |
| StrPrt | String Print | 414 |
| StrSel() | Select string based on comparison | 415 |
| StrToD() | Converts string to a double precision floating point value | 416 |
| StrTok | Parses a string token | 417 |
| StrToL() | Converts string to a long integer value | 418 |
| StrToUL() | Converts string to an unsigned long integer value | 419 |
| StrXmlProtect | Protect XML string | 420 |
| Sub() | Returns the difference of two arrays of formulas | 421 |
| Sum() | Returns the sum of an array of data | 422 |
| Sums1D() | Sums up channels for data | 423 |
| Sums2D() | Sums up channels for 2D Mono data | 424 |
| Sums2G() | Sums up channels for 2D Grey data | 425 |

**Table 7: M300 Function Reference (Continued)**

| Function Name | Function Description | Page |
|---|---|---|
| Sums2GAdv() | Sums up channels for 2D Grey Advanced data | 428 |
| SumsHVPS() | Sums up channels for HVPS data | 431 |
| System() | Access M300 system information | 432 |
| TamdarData() | Tamdar data access | 433 |
| TasP | Pitot Pressure from TAS | 435 |
| Test() | Returns a predicted set values | 436 |
| Time() | Creates a string containing the time | 437 |
| Timer() | Timer fuction | 438 |
| TTemp | Total Air Temperature calculations | 439 |
| Unfold() | Performs a Doppler Unfolding computation | 440 |
| Units() | Converts one unit of measure to another | 441 |
| VaxTime() | Creates a string containing the VAX Time | 443 |
| VaxTimeDiff() | Returns the difference between VAX and M300 time | 444 |
| VectorAngle() | Calculates a vector angle | 445 |
| VectorLen() | Calculates a vector length | 446 |
| Vols() | Computes volumes | 447 |
| Volts() | Converts analog values to volts | 448 |

**Table 7: M300 Function Reference (Continued)**

# Function Prototype Quick Reference

The following table lists the functions prototypes for quick reference purposes.

| Function Prototype | Page |
|---|---|
| **Accumulate**(A) | 203 |
| **Add**(A, B) | 204 |
| **AIMMSData**(A, SELECT) | 205 |
| **Alarm**(HOUR, MINUTE, SECOND, OFFSET, DURATION) | 211 |
| **AltP**(ALTITUDE) | 212 |
| **Areas**(PROBE, F, CFAC, TAS)<br>**Areas**(PROBE, F, CFAC, TAS, FREQUENCY)<br>**Areas**(F, PROBE, RANGE, CFAC, TAS, INTERVAL) | 213 |
| **Arinc429Out**(BOARD, DATA, LABEL, BITS, RANGE) | 214 |
| **Arinc708Data**(A, SELECT) | 215 |
| **Array**(F, INDEX, VALUE) | 217 |
| **AsyncData**(A, OFFSET) | 218 |
| **Average**(F, CYCLES, STATE) | 219 |
| **Avg**(F) | 220 |
| **Bearing**(LATFROM, LONFROM, LATTO, LONTO) | 221 |
| **BufferTime**(SELECT) | 222 |
| **CArray**(F, INDEX) | 223 |
| **CASData**(A, SELECT) | 224 |
| **CASDPOL**(A, SELECT, BOARD) | 227 |
| **CASDPOLData**(A, SELECT, BOARD) | 229 |
| **CASDPOLSums**(A, INTERVAL, STATE, BOARD) | 230 |
| **CASPBPData**(A, SELECT) | 231 |
| **CIndex**(F, INDEX) | 232 |
| **CIPData**(A, SELECT) | 233 |
| **CIPGSData**(A, SELECT) | 235 |

**Function Prototype Quick Reference**

| Function Prototype | Page |
|---|---|
| **CIPGSInfo**(A, SELECT) | 237 |
| **CIPGSMonitor**(A, SHADOW) | 238 |
| **CIPGSSums**(PROBE, A, MODE, FREQUENCY)<br>**CIPGSSums**(A, MODE, PROBE, INTERVAL, PIXELSIZE) | 239 |
| **CIPMonitor**(A) | 240 |
| **CIPSums**(PROBE, A, MODE, FREQUENCY)<br>**CIPSums**(A, MODE, PROBE, INTERVAL, PIXELSIZE) | 241 |
| **Cmd1D**(A) | 242 |
| **Co1DCmd**(BOARD, COMMAND) | 243 |
| **Co2DTAS**(BOARD, FREQUENCY) | 244 |
| **Co2GCmd**(BOARD, COMMAND) | 245 |
| **Co2GTAS**(BOARD, FREQUENCY) | 246 |
| **CoATDAQ141X**(BOARD, VOLTAGE, CHANNEL ) | 247 |
| **CoCIPGSTAS**(BOARD, FREQUENCY) | 248 |
| **CoCIPTAS**(BOARD, FREQUENCY) | 249 |
| **CoCYDDA**(BOARD, VOLTAGE, CHANNEL, MODE) | 250 |
| **CoDo**(BOARD, PORT, BIT, VALUE)) | 251 |
| **CoDT2817**(BOARD, PORT, BIT, VALUE) | 252 |
| **CoFile**(STATE) | 253 |
| **Color**(COLORNAME) | 254 |
| **Comb**(A, AINDEX, AELEMENTS, B, BINDEX, BELEMENTS) | 255 |
| **Concs**(PROBE, F, CFAC, TAS, MODE)<br>**Concs**(F, PROBE, RANGE, CFAC, TAS, INTERVAL, MODE) | 256 |
| **CoPCIDACDA**(BOARD, VOLTAGE, CHANNEL ) | 258 |
| **CoPMFDA**(BOARD, VOLTAGE, CHANNEL) | 259 |
| **Copy**(F, INDEX, ELEMENTS) | 260 |
| **CoQuit**(STATE) | 261 |
| **CoRTI802**(BOARD, VOLTAGE, CHANNEL, MODE) | 262 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **CoSeaDA**(BOARD, VOLTAGE) | 263 |
| **CoShutdown**(STATE) | 264 |
| **CountBy**(DATA, BINS) | 265 |
| **CountEdges**(F)<br>**CountEdges**(F, DIR) | 266 |
| **Cumulative**(F) | 267 |
| **Date**(A0) | 268 |
| **DateTime**(A0) | 269 |
| **DayOfYear**(A0)<br>**DayOfYear**(YEAR, MONTH, DAY) | 270 |
| **Delay**(F, CYCLES) | 271 |
| **Delta**(F, CYCLES) | 272 |
| **DewPointToRH**(DP, TEMP) | 273 |
| **DFault**(F, LOW, HIGH, DEFAULT) | 274 |
| **DIndex**(F, INDEX) | 275 |
| **DirData**(A, SELECT) | 276 |
| **Div**(A, B) | 277 |
| **DToF**(A) | 278 |
| **Eq**(A, B, FTRUE, FFALSE) | 279 |
| **Esi**(T) | 280 |
| **Esw**(T) | 281 |
| **EvtStr**(EVENT, BIT, STR0, STR1) | 282 |
| **EvtVal**(EVENT, BIT, STATE) | 283 |
| **FalconData**(A, TITLE) | 284 |
| **FalconDay**(A, OFFSET) | 285 |
| **FalconTime**(A, OFFSET) | 286 |
| **FArray**(F, INDEX) | 287 |
| **FIndex**(F, INDEX) | 288 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **Ge**(A, B, FTRUE, FFALSE) | 289 |
| **GetData**(A, OFFSET, COUNT, TYPE)<br>**GetData**(A, OFFSET, COUNT, TYPE, SWAP) | 290 |
| **GrData**(A, SELECT) | 291 |
| **GrSums**(PROBE, A, MODE, FREQUENCY)<br>**GrSums**(A, MODE, PROBE, INTERVAL, STATE, PIXELSIZE) | 293 |
| **Gt**(A, B, FTRUE, FFALSE) | 296 |
| **HSAnalog**(A, X, B) | 297 |
| **HvMask**(A) | 298 |
| **HvpsMask**(A) | 299 |
| **HvpsTiming**(A, TASFACTORTAG) | 300 |
| **HvSums**(A, PROBE, MODE, FREQUENCY)<br>**HvSums**(A, PROBE, INTERVAL) | 301 |
| **HvTiming**(A, TASFACTORTAG) | 302 |
| **IArray**(F, INDEX) | 303 |
| **IasP**(IAS) | 304 |
| **IIndex**(F, INDEX) | 305 |
| **Incloud**(STDEV, STDEVTHRES, SPOWER, PPOWER, POWERTHRES, TIME) | 306 |
| **Ins429Bin**(A, BITS, RANGE) | 307 |
| **InsBCD**(A) | 308 |
| **InsBin**(A) | 309 |
| **InsBin2**(A) | 310 |
| **InsPos**(A) | 311 |
| **IntegerData**(A, INDEX, SCALE, OFFSET, SWAP) | 312 |
| **Intercept**(KNOWYS, NKOWNXS, STATE) | 313 |
| **IR**(F, LOW, HIGH) | 314 |
| **IVar1D**(A, STRINDEX, TOTSTRINDEX, CFAC, INTERVAL) | 315 |
| **IVar1DAdv**(STROBE, TOTSTROBES, CFAC, INTERVAL) | 316 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **KeyIndex**(KEY)<br>**KeyIndex**(KEY, COUNT) | 317 |
| **LArray**(F, INDEX) | 318 |
| **LatStr**(LATITUDE) | 319 |
| **Le**(A, B, FTRUE, FFALSE) | 320 |
| **LeastSqReg**(KNOWNYS, KNOWNXS, STATE, COUNT) | 321 |
| **Limit**(F, LOW, HIGH) | 322 |
| **LIndex**(F, INDEX) | 323 |
| **LonStr**(LONGITUDE) | 324 |
| **Lookup**(F, LOOKUP) | 325 |
| **LookupGet**(LOOKUP, ROW, COLUMN) | 326 |
| **LookupSet**(LOOKUP, ROW, COLUMN, F) | 327 |
| **LrnPos**(A) | 328 |
| **Lt**(A, B, FTRUE, FFALSE) | 329 |
| **LToF**(A) | 330 |
| **Masses**(F, PROBE, RANGE, CFAC, TAS, INTERVAL, LINEAR, EXP)<br>**Masses**(PROBE, F, CFAC, TAS, FREQUENCY, LINEAR, EXP) | 331 |
| **Max**(F) | 332 |
| **MaxSiz**(X, Y)<br>**MaxSiz**(X, Y, MODE) | 333 |
| **MaxTim**(F, STATE) | 334 |
| **MaxVal**(F, STATE) | 335 |
| **Mean**(X, Y) | 336 |
| **Median**(X, Y)<br>**Median**(X, Y, MODE) | 337 |
| **Min**(F) | 338 |
| **MinSiz**(X, Y)<br>**MinSiz**(X, Y, MODE) | 339 |
| **MinTim**(F, STATE) | 340 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **MinVal**(F, STATE) | 341 |
| **Mode**(X, Y) | 342 |
| **MoSums**(PROBE, A, MODE, FREQUENCY)<br>**MoSums**(A, ELAPSED, MODE, PROBE, INTERVAL) | 343 |
| **Mul**(A, B) | 344 |
| **Nmea**(F, IDSTR, SELSTR) | 345 |
| **OdCmd**(PROBE, A) | 347 |
| **OdIVar**(A, STROBEINDEX, TOTALSTROBEINDEX, CFAC, INTERVAL) | 348 |
| **OdIVarAdv**(STROBETAG, TOTALSTROBETAG, CFAC, INTERVAL) | 349 |
| **OdRef**(A) | 350 |
| **OdSums**(A, FIRST, FREQUENCY)<br>**OdSums**(A, INTERVAL, STATE, FIRST) | 351 |
| **PAlt**(SPRES) | 352 |
| **PIas**(PPRES) | 353 |
| **Poly**(X, $A_0$, $A_1$, ..., $A_n$) | 354 |
| **PosAvData**(A, SELECT) | 355 |
| **Power**(A, B) | 357 |
| **PqConfig**(A, SELECT) | 358 |
| **PqPower**(A, GATEWIDTH, HITS, SCALE, OFFSET, MODE) | 359 |
| **PqRange**(GATEWIDTH, GATES)<br>**PqRange**(GATEWIDTH, GATES, CLOCK) | 360 |
| **PqRaw**(A, SELECT) | 361 |
| **PqReflectivity**(POWER, RANGE, RCON) | 362 |
| **PqStatus**(A, SELECT) | 363 |
| **PrData**(PROBE, SELECT) | 364 |
| **ProbeData**(PROBE, RANGE, SELECT) | 365 |
| **PromoBins**(AMP, TTIME, AMPBINS, POINTS, TTIMEMIN, TTIMEMAX)<br>**PromoBins**(TTIME, TTIMEBINS, POINTS, TTIMEMIN, TTIMEMAX) | 366 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **PromoData**(A, SELECT) | 367 |
| **Protect**(A, B) | 368 |
| **PrTasClockIn**(A) | 369 |
| **PrTasClockOut**(PROBE, TAS) | 370 |
| **PTas**(STEMP, PPRES, SPRES) | 371 |
| **RaConstant**(RADAR, WAVEGUIDELOSS, K2) | 372 |
| **Rand**(SELECT) | 373 |
| **RandData**(SCALE, OFFSET, MINIMUM, MAXIMUM ) | 374 |
| **RandSeed**(SEED) | 375 |
| **Range**(REFLAT, REFLON, LAT, LON) | 376 |
| **Ref1D**(A) | 377 |
| **RHToDewPoint**(RH, TEMP) | 378 |
| **Scale**(X, A, B) | 379 |
| **Scale2**(X, A, B, C) | 380 |
| **Scale3**(X, A, B, C, D) | 381 |
| **ScaleArray**(X, A, B) | 382 |
| **ScaleArray2**(X, A, B, C) | 383 |
| **ScaleArray3**((X, A, B, C, D) | 384 |
| **Seconds**(A0) | 385 |
| **SerialASCII**(A, INDEX, DELIMITER, COUNT, MODE) | 386 |
| **SerialDADS**(A, INDEX, IDENTIFIER) | 387 |
| **SerialIEEE**(A, INDEX, COUNT) | 388 |
| **SerialInteger**(A, INDEX, COUNT) | 389 |
| **SrNmea**(F, IDSTR, INDEX, COUNT, MODE)<br>**SrNmea**(F, IDSTR, INDEX, COUNT, MODE, HEX) | 404 |
| **SerialVAX**(A, INDEX, COUNT) | 390 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **Set**(INIT)<br>**Set**(INIT, INC)<br>**Set**(INIT, INC, COUNT) | 391 |
| **Sizes**(PROBE, RANGE) | 392 |
| **Skip**(VALUE, SKIPTO) | 393 |
| **Slope**(KNOWNYS, KNOWNXS, STATE) | 394 |
| **SpData**(A, SELECT) | 395 |
| **Spp100Data**(A, SELECT) | 397 |
| **SrASCII**(A, INDEX, DELIMITER, COUNT, MODE) | 399 |
| **SrDADS**(A, INDEX, IDENTIFIER) | 400 |
| **SrData**(TAG, OFFSET, COUNT, MODE, SWAP) | 401 |
| **SrIEEE**(A, INDEX, COUNT) | 402 |
| **SrInteger**(A, INDEX, COUNT) | 403 |
| **SrVAX**(A, INDEX, COUNT) | 406 |
| **StDev**(X, STATE) | 407 |
| **STemp**(TTEMP, PPRES, SPRES, RECOVERY) | 408 |
| **Stormscope**(LATITUDE, LONGITUDE, COUNT, STATE) | 409 |
| **StrCat**(STRING1, STRING2)<br>**StrCat**(STRING1, STRING2, LENGTH) | 410 |
| **StrCmp**(STRING1, STRING2)<br>**StrCmp**(STRING1, STRING2, LENGTH) | 411 |
| **StrCpy**(STRING)<br>**StrCpy**(STRING, LENGTH) | 412 |
| **StrParameters**(STRING, DELIMETER) | 413 |
| **StrPrt**(FORMAT, VALUE) | 414 |
| **StrSel**(VALUE, SELECT, STRING) | 415 |
| **StrToD**(STRING)<br>**StrToD**(STRING, OFFSET) | 416 |
| **Strtok**(STRING, TOKENS) | 417 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **StrToL**(STRING)<br>**StrToL**(STRING, OFFSET)<br>**StrToL**(STRING, OFFSET, BASE)<br>**StrToL**(STRING, OFFSET, BASE, STRLEN, STRCOUNT) | 418 |
| **StrToUL**(STRING)<br>**StrToUL**(STRING, OFFSET)<br>**StrToUL**(STRING, OFFSET, BASE)<br>**StrToUL**(STRING, OFFSET, BASE, STRLEN, STRCOUNT) | 419 |
| **StrXmlProtect** (STRING) | 420 |
| **Sub**(A, B) | 421 |
| **Sum**(F) | 422 |
| **Sums1D**(A, INTERVAL, STATE, SKIP) | 423 |
| **Sums2D**(2D, ELAPSED, MODE, PROBE, INTERVAL) | 424 |
| **Sums2G**(ORTAG, SLICETAG, ELAPSEDTAG, MODE, PROBE, INERVAL, STATE)<br>**Sums2G**(ORTAG, SLICETAG, ELAPSEDTAG, MODE, PROBE, INERVAL, STATE, YSIZE)<br>**Sums2G**(ORTAG, MINTAG, MIDTAG, MAXTAG, ELAPSEDTAG, MODE, PROBE, INTERVAL, STATE, YSIZE) | 425 |
| **Sums2GAdv**(TAG, MODE, PROBE, INTERVAL, STATE, YSIZE) | 428 |
| **SumsHVPS**(A, PROBE, INTERVAL) | 431 |
| **System**(SELECT) | 432 |
| **TamdarData**(A, SELECT) | 433 |
| **TasP**(STEMP, TAS, SPRES) | 435 |
| **Test**($P_1$, $P_2$, ... , $P_n$) | 436 |
| **Time**(A0) | 437 |
| **Timer**(DELAY, ON, OFF) | 438 |
| **TTemp**(STEMP, PPRES, SPRES, RECOVERY) | 439 |
| **Unfold**(V1, V2) | 440 |
| **Units**(X, TO, FROM) | 441 |
| **VaxTime**(A) | 443 |

**Function Prototype Quick Reference (Continued)**

| Function Prototype | Page |
|---|---|
| **VaxTimeDiff**(A) | 444 |
| **VectorAngle**(X, Y) | 445 |
| **VectorLen**(X, Y) | 446 |
| **Vols**(PROBE, F, CFAC, TAS)<br>**Vols**(F, PROBE, RANGE, CFAC, TAS, INTERVAL) | 447 |
| **Volts**(A) | 448 |

**Function Prototype Quick Reference (Continued)**

# Accumulate(), Accumulate Arrays

## Synopsis

**Accumulate**(A)

**A**[*n*] Formula of an array of values to be accumulated (*n*≥2).

## Description

The resulting array of values is explained via the following equation.

$$f[n-1] = A[n-1]$$

$$f[i] = f[i+1] + A[i]$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"Accumulate"    ""     F300    F[15]   Accumulate(F100)
```

# Add(), Add Arrays

## Synopsis

**Add**(A, B)

**A**[*m*] Formula of an array of values (*m*≥1).

**B**[*p*] Formula of an array of values (*p*≥1).

*Note: Deprecated [M300 replacement function - See "+, Add"]*

## Description

This function returns an array of values representing the addition of the two given arrays, element by element. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i] + B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*, *p*)

## Example

```
; Name     Units  Number  Result  Computations
"Add"      ""     F300    F[15]   Add(F200, F201)
```

# AIMMSData(), AIMMS Data Access

## Synopsis

**AIMMSData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for AIMMS/ADP data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..23). |

## Description

This function allows access to individual items of the AIMMS/ADP data block. The following table shows the different **SELECT** values for the different AIMMS/ADP data fields. The function will returns the value of a user specified item from a AIMMS/ADP buffer. Please check the AIMMS/ADP manual for further information.

| Data Field | SELECT | Result |
|---|---|---|
| Time (HH:MM:SS) | 0 | S[12] |
| Temperature (Celsius) | 1 | F[1] |
| Relative Humidity (%) | 2 | F[1] |
| Barometric Pressure (pa) | 3 | F[1] |
| Wind Flow Vector NS (m/s) | 4 | F[1] |
| Wind Flow Vector EW (m/s) | 5 | F[1] |
| Wind Speed (m/s) | 6 | F[1] |
| Wind Direction (deg) | 7 | F[1] |
| Wind Solution Flag | 8 | I[1] |

**AIMMS 20 Id 00 (Standard Meteorology Packet, para3 = 00) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Time (HH:MM:SS) | 0 | S[12] |
| Latitude (deg) | 1 | F[1] |
| Longitude (deg) | 2 | F[1] |

**AIMMS 20 Id 01 (Aircraft Data Packet, para3 = 01) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Altitude (m) | 3 | F[1] |
| Velocity NS (m/s) | 4 | F[1] |
| Velocity EW (m/s) | 5 | F[1] |
| Velocity UD (m/s) | 6 | F[1] |
| Roll (deg) | 7 | F[1] |
| Pitch (deg) | 8 | F[1]] |
| Yaw (deg) | 9 | F[1] |
| TAS (m/s) | 10 | F[1] |
| Vertical Wind (m/s) | 11 | F[1] |
| Side Slip (deg) | 12 | F[1] |
| AOA Pressure Diff (pa) | 13 | F[1] |
| Side Slip Diff (pa) | 14 | F[1] |

**AIMMS 20 Id 01 (Aircraft Data Packet, para3 = 01) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Time (HH:MM:SS) | 0 | S[12] |
| Latitude (rad) | 1 | D[1] |
| Longitude (rad) | 2 | D[1] |
| Altitude (m) | 3 | F[1] |
| Ground Speed (m/s) | 4 | F[1] |
| Ground Track (deg) | 5 | F[1] |
| HFOM (m) | 6 | F[1] |
| VFOM (m) | 7 | F[1] |
| Navigation Mode | 8 | L[1] |
| Satellites | 9 | F[1] |
| Datum Number | 10 | F[1] |

**AIMMS 20 Id 02 (GPS Navigation Packet, para3 = 02) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Flow Rate (ml/min) | 0 | F[1] |

**AIMMS 20 Id 04 (Purge Flow, para3 = 04) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Heater Block Temperature Forward (Celsius) | 0 | F[1] |
| Heater Block Temperature Aft (Celsius) | 1 | F[1] |
| Low Temperature Threshold (Celsius) | 2 | F[1] |

**AIMMS 20 Id 05 (Internal Probe Temperature, para3 = 05) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Temperature (Celsius) | 0 | F[1] |
| Relative Humidity (%) | 1 | F[1] |
| Barometric Pressure (pa) | 2 | F[1] |
| Pitot Pressure (pa) | 3 | F[1] |
| AOA Pressure Diff (pa) | 4 | F[1] |
| Sideslip Pressure Diff (pa) | 5 | F[1] |

**AIMMS 20 (ADP) Id 11 (ADP Raw Data Packet, para3 = 11) Select**

| Data Field | SELECT | Result |
|---|---|---|
| Temperature (Celsius) | 0 | F[1] |
| Relative Humidity (%) | 1 | F[1] |
| Barometric Pressure (pa) | 2 | F[1] |
| TAS (m/s) | 3 | F[1] |

**AIMMS 20 (ADP) Id 12 (Airdata Corrected for Dynamic Effects, para3 = 12) Select**

## Result Type/Space

See select tables above.

## Example

```
; Name       Units     Number  Result  Computations
;
; Aimms 20 Trigger
;
Trigger = "AIMMS" 10 Aimms20 "Ignore" Never None
;
; para 3 is the aimms id
"ID" "" F1999 L[1] DirData(A1000, 7)
; individual id flags for each id trigger
"ID0" "" F1098 L[1] Eq(F1999, 0, 1, 0)
"ID1" "" F1198 L[1] Eq(F1999, 1, 1, 0)
"ID2" "" F1298 L[1] Eq(F1999, 2, 1, 0)
;
; Id=0, Standard Meteorology Packet
;
Trigger = "AIMMS" 10 Aimms20 F1098 "Ignore" Never None
;
"Time" "" F1000 S[12] AimmsData(A1000, 0)
"Temp" "øc" F1001 F[1] AimmsData(A1000, 1)
"RH1" "%" F1002 F[1] AimmsData(A1000, 2)
"BaroPress" "pa" F1003 F[1] AimmsData(A1000, 3)
"WindFlowNS" "m/s" F1004 F[1] AimmsData(A1000, 4)
"WindFlowEW" "m/s" F1005 F[1] AimmsData(A1000, 5)
"WindSpeed" "m/s" F1006 F[1] AimmsData(A1000, 6)
"WindDir" "deg" F1007 F[1] AimmsData(A1000, 7)
"WindSolutionFlag" "" F1008 I[1] AimmsData(A1000, 8)
;
"BaroPress" "mbar" F1053 F[1] Units(F1003, "mbar", "pa")
"WindSpeed" "knots" F1056 F[1] Units(F1006, "knots", "m/s")
"WindDir" "rad" F1057 F[1] Units(F1007, "rad", "deg")
"WindSolutionFlag" "" F1058 S[10] EvtStr(F1008, 0, "Invalid", "  Valid")
;
"Altitude" "ft" F1093 F[1] PAlt(F1053)
;
"ID0Aimms20Count" "" F1099 L[1] F1099 ++
;
; Id=1, Aircraft State Data Packet
;
Trigger = "AIMMS" 10 Aimms20 F1198 "Ignore" Never None
;
"Time" "" F1100 S[12] AimmsData(A1000, 0)
"Latitude" "deg" F1101 F[1] AimmsData(A1000, 1)
"Longitude" "deg" F1102 F[1] AimmsData(A1000, 2)
"Altitude" "m" F1103 F[1] AimmsData(A1000, 3)
"VelocityNS" "m/s" F1104 F[1] AimmsData(A1000, 4)
"VelocityEW" "m/s" F1105 F[1] AimmsData(A1000, 5)
"VelocityUD" "m/s" F1106 F[1] AimmsData(A1000, 6)
```

```
"Roll" "deg" F1107 F[1] AimmsData(A1000, 7)
"Pitch" "deg" F1108 F[1] AimmsData(A1000, 8)
"Yaw" "deg" F1109 F[1] AimmsData(A1000, 9)
"TAS" "m/s" F1110 F[1] AimmsData(A1000, 10)
"VerticalWind" "m/s" F1111 F[1] AimmsData(A1000, 11)
"Sideslip" "deg" F1112 F[1] AimmsData(A1000, 12)
"AOAPressDiff" "" F1113 F[1] AimmsData(A1000, 13)
"SideslipDiff" "" F1114 F[1] AimmsData(A1000, 14)
;
"Latitude" "rad" F1151 F[1] Units(F1101, "rad", "deg")
"Longitude" "rad" F1152 F[1] Units(F1102, "rad", "deg")
"Altitude" "ft" F1153 F[1] Units(F1103, "ft", "m")
;
"LatitudeStr" "" F1191 S[12] LatStr(F1151)
"LongitudeStr" "" F1192 S[12] LonStr(F1152)
;
"ID1Aimms20Count" "" F1199 L[1] F1099 ++
;
; Id=2, GPS Navigation Packet
;
Trigger = "AIMMS" 10 Aimms20 F1298 "Ignore" Never None
;
"Time" "h" F1200 F[1] AimmsData(A1000, 0)
"Latitude" "rad" F1201 D[1] AimmsData(A1000, 1)
"Longitude" "rad" F1202 D[1] AimmsData(A1000, 2)
"Altitude" "m" F1203 F[1] AimmsData(A1000, 3)
"GroundSpeed" "m/s" F1204 F[1] AimmsData(A1000, 4)
"GroundTrack" "rad" F1205 F[1] AimmsData(A1000, 5)
"HFOM" "m" F1206 F[1] AimmsData(A1000, 6)
"VFOM" "m" F1207 F[1] AimmsData(A1000, 7)
"NavigationMode" "" F1208 L[1] AimmsData(A1000, 8)
"Satellites" "" F1209 F[1] AimmsData(A1000, 9)
"DatumNumber" "" F1210 F[1] AimmsData(A1000, 10)
;
"Latitude" "deg" F1251 D[1] Units(F1201, "deg", "rad")
"Longitude" "deg" F1252 D[1] Units(F1202, "deg", "rad")
"Altitude" "ft" F1253 F[1] Units(F1203, "ft", "m")
"GroundSpeed" "knots" F1254 F[1] Units(F1204, "knots", "m/s")
"GroundTrack" "deg" F1255 F[1] Units(F1205, "deg", "rad")
;
"NavModeStatusByte" "" F1260 I[1] F1208 0x1F &
"InitRequired" "" F1261 S[20] StrSel(F1260, 0, "Init Required    ")
"Initialized" "" F1261 S[20] StrSel(F1260, 1, "Initialized      ")
"3DNavSolution" "" F1261 S[20] StrSel(F1260, 2, "3D Nav Solution  ")
"2DNavSolution" "" F1261 S[20] StrSel(F1260, 3, "2D Nav Solution  ")
"3DDiffCorrSol" "" F1261 S[20] StrSel(F1260, 4, "3D Diff Corrected")
"2DDiffCorrSol" "" F1261 S[20] StrSel(F1260, 5, "2D Diff Corrected")
"DeadReckoning" "" F1261 S[20] StrSel(F1260, 6, "Dead Reckoning   ")
;
"SolutionConfidenceLevel" "" F1262 S[20] EvtStr(F1208, 5, "Normal", "  High")
"GPSTimeAlignmentMode" "" F1263 S[20] EvtStr(F1208, 7, "Enabled", "Disabled")
;
"LatitudeStr" "" F1291 S[12] LatStr(F1201)
```

```
"LongitudeStr" "" F1292 S[12] LonStr(F1202)
;
"ID2Aimms20Count" "" F1299 L[1] F1099 ++
```

# Alarm(), Alarm

## Synopsis

**Alarm**(HOUR, MINUTE, SECOND, OFFSET, DURATION)

| | |
|---|---|
| **HOUR**[1] | Hours to synchronize to (integer). |
| **MINUTE**[1] | Minutes to synchronize to (integer). |
| **SECOND**[1] | Seconds to synchronize to (integer). |
| **OFFSET**[1] | Offset from specified time in seconds (long). |
| **DURATION**[1] | Duration of high value in seconds (long). |

## Description

This function is used to generate true (return one) or false (return zero) events at particular time intervals with respect to the M300 time. For example, if an event is desired for every minute at 5 seconds after the minute mark and that will last for 10 seconds, then the function call would look like Alarm(0, 1, 0, 5, 10).

The formula trigger will affect this function. The user can control the trigger to determine the number of times the function gets called.

## Result Type/Space

D[1]

## Example

```
; Trigger
Trigger = Sync 1 None
; Name            Units  Number  Result  Computations
"GenerateAlarm"  ""     F100    I[1]    Alarm(0, 1, 0, 5, 10)
```

# AltP(), Inverse Pressure Altitude

## Synopsis

**AltP**(ALTITUDE)
**ALTITUDE**[*n*] Formula of an array of values containing altitude (in ft) (*n*≥1).

## Description

This function computes static pressure from altitude.

$$f[i] = 1013.25 \cdot \left(1 - \frac{ALTITUDE}{1.45458^5}\right)^{5.25486} mb$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name                    Units  Number  Result  Computations
"StaticPressure"          "mb"    F200   F[1]    AltP(F100)
```

# Areas(), Areas

## Synopsis

**Areas**(PROBE, F, CFAC, TAS)
**Areas**(PROBE, F, CFAC, TAS, FREQUENCY)
**Areas**(F, PROBE, RANGE, CFAC, TAS, INTERVAL)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **F**[*m*] | Formula of an array of sums for each channel (*m*≥1). |
| **RANGE**[1] | Range used in probe definition table (integer). |
| **CFAC**[1] | Correction factor. |
| **TAS**[1] | True air speed value. |
| **INTERVAL**[1] | Interval of summation used by sums routine (integer). |
| **FREQUENCY**[1] | Frequency of summation used by sums routine. |

☞ ***Note: For function calls without a RANGE argument, the range is passed via the probe entry argument.***

☞ ***Note: For function calls without an INTERVAL or FREQUENCY argument, a value of 1 will be used for either one.***

## Description

This function uses the summed up channel counts and the probe definition table to compute areas. The result is typically used for mean, median, mode, total area calculations as well as X vs. Y display plots. This function should be 'triggered' at the same time interval as the summation routine generates data, so as to eliminate redundant calculations on the same input data. The 'AREA' and 'SAREA' originate from the user specified channel files via the probe entry. The 'BUFLIFE' and 'SYSFREQ' refer to the values entered in the system table. The SYSFREQ is associated with the system frequency in the time data. This comes from the frequency values in the system board entry. The BUFLIFE is associated with the buffer life in the time data. In the M300 system, the buffer life and system frequency are the same (for synchronous buffers).

$$f[i] = \frac{F[i] \cdot AREA[i, RANGE]}{SAREA[i, RANGE] \cdot TAS \cdot \frac{BUFLIFE}{SYSFREQ} \cdot IVAL \cdot CFAC}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = min(*m*, [*probe channels*])

## Example

```
; Name      Units  Number  Result  Computations
"Areas"     ""     F200    F[15]   Areas(Pr.1D, F100, F54, F102)
```

# Arinc429Out, ARINC 429 Output

## Synopsis

**Arinc429Out**(BOARD, DATA, LABEL, BITS, RANGE)

| | |
|---|---|
| **BOARD** | Arinc429 board entry (board). |
| **DATA**[1] | Data to be sent out (float or long). |
| **LABEL**[1] | Arinc user label (integer). |
| **BITS**[1] | Number of bits to use (integer). |
| **RANGE**[1] | Range scale for data. |

## Description

This function is used to send ARINC429 data out to the ARINC 429 board. The user must specify the ARINC label to use. The function returns one when successful and zero for failure.

Valid values for bits are 0 (to send raw integer data) or 2 - 20 (to send scaled data).

## Result Type/Space

D[1]

## Example

```
; Name          Units  Number  Result  Computations
"Arinc429Out"  ""     F300    I[1]    Arinc429Out(Bd.Arinc429,F100,42,10,600.0)
```

# Arinc708Data(), ARINC 708 Data

## Synopsis

**Arinc708Data**(A, SELECT)

**A** Acquisition tag for Ballard ARINC 708 data (tag).

**SELECT**[1] Data field select (integer: 1, 9, 17, 30, 37, 43, 50, 52, 65).

## Description

This function is used to retrieve certain ARINC 708 data. The SELECT field chooses which data fields are returned.

| SELECT | Probe Data | Return Type |
|---|---|---|
| 1 | label | I[1] |
| 9 | control data accept | I[1] |
| 17 | faults | I[1] |
| 30 | tilt | F[1] |
| 37 | gain | F[1] |
| 43 | maximum range | F[1] |
| 50 | control data accept | I[1] |
| 52 | beam angle | F[1] |
| 65 | reflectivity data | I[512] |

**SELECT**

## Result Type/Space

For a listing of possible return type dependencies, see the previous table.

## Example

```
;
; Ballard 708
;
Trigger = "Ballard 708" 100 Ballard708 "Never" Never None
;
; Name          Units   Number  Result   Computations
;
"BeamAngle" "deg" F1500 F[1] Arinc708Data(A1000, 52)
"Tilt" "deg" F1510 F[1] Arinc708Data(A1000, 30)
"MaxRange" "nmi" F1520 F[1] Arinc708Data(A1000, 43)
"Label" "" F1530 I[1] Arinc708Data(A1000, 1)
"Gain" "db" F1540 F[1] Arinc708Data(A1000, 37)
"Faults" "" F1550 I[1] Arinc708Data(A1000, 17)
"DataAccept" "" F1560 I[1] Arinc708Data(A1000, 50)
"ControlDataAccept" "" F1570 I[1] Arinc708Data(A1000, 9)
;
"NumberOfBins" "" F1581 I[1] 512
"FakedGateWidth" "" F1582 l[1] 500
"FakedHits" "" F1583 l[1] 10
"FakedClock" "" F1584 F[1] F1520 F1581 / 1.26753e-5 *
;
"ReflectivityData" "" F1600 I[512] Arinc708Data(A1000, 65)
;
"Range" "m" F2000 F[512] PqRange(1, 512, F1584)
;
"CPower" "dbm" F2010 F[512] 0
;
"Reflectivity" "dbz" F2030 F[512] F1600
;
"MinusAngle" "deg" F2040 F[1] F1500 360.0 -
"RelAngle" "deg" F2042 F[1] Gt(F1500, 180.0, F2040, F1500)
"RelAngle" "deg" F2042 F[1] F2042 -1.0 *
;
"PPIAngle" "rad" F2045 F[1] F2042 F5 *
```

# Array(), Array

## Synopsis

**Array**(F, INDEX, VALUE)

| | |
|---|---|
| **F**[*n*] | Formula of an array of ($n \geq 1$). |
| **INDEX**[1] | Index of element to be changed (integer ≥ 0). |
| **VALUE**[1] | New value for array element. |

## Description

Used to construct arrays of elements or just change a few elements in an array.

$$F[INDEX] = VALUE$$

## Result Type/Space

D[*n*]

## Example

```
; Name     Units  Number  Result  Computations
"Array"   ""     F100    F[15]   Array(F100, 5, 3.14159)
```

# AsyncData(), Asynchronous Data

## Synopsis

**AsyncData**(A, OFFSET)

| | |
|---|---|
| **A** | Acquisition tag for asynchronous data (tag). |
| **OFFSET**[1] | Byte offset into data block (integer). |

☞ ***Note: Deprecated (M300 uses other means to retrieve needed asynchronous data). Access tag with the desired trigger.***

## Description

This function is used to retrieve data from the asynchronous buffer into a formula. This allows calculations and display to work with this asynchronous data. In the M200 system, the user only had to specify an asynchronous tag number to get the desired data. In the M300 system, the user must specify the correct trigger to be able to get the desired anachronism data.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Trigger
Trigger = Never Never None "2D Mono" 1 2DMono
; Name             Units  Number  Result  Formulas
"ElapsedTime"     ""     F7012   L[1]    AsyncData(A7007, 0)
```

# Average(), Average

## Synopsis

**Average**(F, CYCLES, STATE)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **CYCLES**[1] | Period of time for average (integer). |
| **STATE**[1] | State (integer: 0 or 1). |

## Description

This function computes the average of each element of the given formula for the last period of cycles. The values returned are the sum of all the values divided by the number of cycles. When the number of samples equals or exceeds the number of cycles, the oldest sample is discarded and the new sample is included in the average.

If the state is zero, the average is not changed. If the state changes between zero and one, the average is cleared and a new average is started. If the state is a one, the average is computed and returned every time.

The number of cycles can be affected by the current trigger.

## Result Type/Space

D[*n*]

## Example

```
; Name         Units  Number  Result  Computations
"Average"     ""     F200    F[5]    Average(F100, 60, 1)
```

# Avg(), Average Array

## Synopsis

**Avg**(F)

**F**[*n*] Formula of an array of values to be averaged (*n*≥1).

## Description

This function computes the average of the given formula (array of values). The value returned is the sum of all the values divided by the number of values in the array. The following formula summarizes the computation.

$$f = \left[ \frac{\sum_{i=0}^{n-1} F[i]}{n} \right]$$

## Result

D[1]

## Example

```
; Name         Units  Number  Result  Computations
"Average"     "m/s"  F200    F[1]    Avg(F100)
```

# Bearing(), Aircraft Bearing

## Synopsis

**Bearing**(LATFROM, LONFROM, LATTO, LONTO)

| | |
|---|---|
| **LATFROM**[*m*] | Latitude of reference point from (in rad) (*m*≥1). |
| **LONFROM**[*p*] | Longitude of reference point from (in rad) (*p*≥1). |
| **LATTO**[*q*] | Latitude of target point to (in rad) (*q*≥1). |
| **LONTO**[*r*] | Longitude of target point to (in rad) (*r*≥1). |

## Description

This function returns the bearing from the reference point to the target point in radians. Typically, the reference point is the aircraft's current position and the target point is a fixed point on the ground.

$$f[i] = \left[\text{atan}\left(\frac{(LONTO[i] - LONFROM[i]) \cdot \cos(LATTO[i])}{LATTO[i] - LATFROM[i]}\right)\right]$$

$$\text{if } (f < 0) \text{ then } f = f + 2\pi$$

## Result Type/Space

D[*n*], *n* = min(*m, p, q, r*)

## Example

```
; Name       Units  Number  Result  Computations
"Bearing"   "rad"  F300    F[1]    Bearing(F100, F101, F200, F201)
```

# BufferTime(), Buffer Time

## Synopsis

**BufferTime**(SELECT)
**SELECT**[1] Select value (integer: 0...3).

## Description

This function is used to return the buffer time information.

We can return start time and end time in seconds. The resulting seconds are since January 1st 1970 (floating point value).

We can also return the delta time for the buffer (end time - start time) in seconds.

We can return the delta time between buffers (end time current - end time previous).

The following table shows the valid SELECT values for the function.

| SELECT | Name |
|---|---|
| 0 | Start time |
| 1 | End time |
| 2 | Delta time for buffer |
| 3 | Delta time between buffers |

**SELECT**

## Result Type/Space

D[1], F[1]

## Example

```
; Name        Units  Number  Result  Computations
"DeltaTime"    "s"    F300    D[1]   BufferTime(2)
```

# CArray(), Character Array Element Access

## Synopsis

**CArray**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula of an array of characters (string) (*n*≥1). |
| **INDEX**[1] | Index of desired element in character array (integer ≥ 0). |

*Note: Deprecated [M300 Replacement function - See "CIndex(), Character Element Access"]*

## Description

This function allows access to individual elements of a character array. Returns a character referenced by **INDEX** in the string referenced by the formula **F**.

$$f = F[INDEX]$$

## Result Type/Space

D[1]

## Example

```
; Name              Units   Number  Result   Formulas
"CharacterArray"    ""      F200    C[1]     Carray(F2012, 5)
```

# CASData(), CAS Data Access

## Synopsis

**CASData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for CAS data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..40). |

## Description

This function allows access to individual items of the CAS data block, including house data. The following table shows the different **SELECT** values for the different CAS data fields. Please check the CAS manual for further information. To get the size spectrum for the CAS probe, use OdSums(), 1D Sums Data fields with hd[*m*] next to the description indicate house keeping data, where *m* is it's index in the house keeping data.

| Data Field | SELECT |
|---|---|
| Byte Count | 0 |
| Sum of Transit | 1 |
| Sum of Particles | 2 |
| Fifo Full | 3 |
| Reset Flag | 4 |
| Forward Overflow | 5 |
| Backward Overflow | 6 |
| Dynamic Pressure, hd[0] | 10 |
| Static Pressure, hd[1] | 11 |
| Ambient Temperature, hd[2] | 12 |
| Forward Heat Sink Temp, hd[3] | 13 |
| Backward Heat Sink Temp, hd[4] | 14 |
| Forward Block Temp, hd[5] | 15 |
| Back Block Temp, hd[6] | 16 |

**CAS Data SELECT Options**

| Data Field | SELECT |
|---|---|
| Photodiode 1, hd[7] | 17 |
| Photodiode 2, hd[8] | 18 |
| Photodiode 3, hd[9] | 19 |
| Photodiode 4, hd[10] | 20 |
| Qualifier TEC Temp, hd[11] | 21 |
| Forward TEC Temp, hd[12] | 22 |
| Backward TEC Temp, hd[13] | 23 |
| Qualifier Heat Sink Temp, hd[14] | 24 |
| Qualifier High Gain Vol, hd[15] | 25 |
| Qualifier Mid Gain Vol, hd[16] | 26 |
| Qualifier Low Gain Vol, hd[17] | 27 |
| Forward High Gain Vol, hd[18] | 28 |
| Forward Mid Gain Vol, hd[19] | 29 |
| Forward Low Gain Vol, hd[20] | 30 |
| Backward High Gain Vol, hd[21] | 31 |
| Backward Mid Gain Vol, hd[22] | 32 |
| Backward Low Gain Vol, hd[23] | 33 |
| Internal Temp, hd[24] | 34 |
| Spare Analog 1, hd[25] | 35 |
| Spage Analog 2, hd[26] | 36 |
| LWC Hot Wire Signal, hd[27] | 37 |
| LWC Slave Monitor, hd[28] | 38 |
| Laser Current Monitor, hd[29] | 39 |
| Laser Power Monitor, hd[30] | 40 |

**CAS Data SELECT Options (Continued)**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name       Units     Number  Result  Computations
"CasData"   "°F"      F300    F[1]    CasData(Aq.CasData, 12)
```

# CASDPOL(), CAS DPOL PBP

## Synopsis

**CASDPOL**(A, SELECT, BOARD)

| | |
|---|---|
| **A** | Acquisition tag for CAS DPOL data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..24). |
| **BOARD** | Board entry for CAS DPOL. |

## Description

This function allows access to individual items of the CAS DPOL PBP data block. The following table shows the different **SELECT** values for the different CAS DPOL data fields. Please check the CAS DPOL manual for further information.

Result space for the formula is important as far as allowing access to multiple PBP records as well as possible multiple samples per second. The sample included bellow to access the IPT allows up to 700 longs.

| Data Field | SELECT |
|---|---|
| IPT | 0 |
| Sizer High | 1 |
| Sizer Low | 2 |
| Qual High | 3 |
| Qual Low | 4 |
| S High | 5 |
| S Low | 6 |
| P High | 7 |
| P Low | 8 |
| Sizer High Transit Time | 9 |
| Sizer High Peak Time | 10 |
| Qual High Transit Time | 11 |
| Qual High Peak Time | 12 |

**CAS DPOL SELECT Options**

| Data Field | SELECT |
|---|---|
| S High Transit Time | 13 |
| S High Peak Time | 14 |
| P High Transit Time | 15 |
| P High Peak Time | 16 |
| Sizer Low Transit Time | 17 |
| Sizer Low Peak Time | 18 |
| Qual Low Transit Time | 19 |
| Qual Low Peak Time | 20 |
| S Low Transit Time | 21 |
| S Low Peak Time | 22 |
| P Low Transit Time | 23 |
| P Low Peak Time | 24 |

**CAS DPOL SELECT Options (Continued)**

## Result Type/Space

L[*n*], *n* = number of data samples

## Example

```
; Name       Units     Number  Result  Computations
"IPT"       ""      F2610  L[700]  CasDPOL(Aq.casdpoldata, 0, Bd.casdpol)
```

# CASDPOLData(), CAS DPOL Data Access

## Synopsis

**CASDPOLData**(A, SELECT, BOARD)

| | |
|---|---|
| **A** | Acquisition tag for CAS DPOL data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..3). |
| **BOARD** | The CAS DPOL board entry. |

## Description

This function allows access to individual items of the CAS DPOL data block. The following table shows the different **SELECT** values for the different CAS DPOL data fields.

When trying to get fields such as the digital channels please note that you get all 26 channel with one call and must provide a result space of 26 long elements as you can see from the sample bellow. This way to get individual digital channels such as laser current you must use the LIndex function and specify the correct channel desired.

Please see the /test/casdpol for a complete sample that you may use as a starting project to your project.

| Data Field | SELECT |
|---|---|
| Number of PBP Packets | 0 |
| Housekeeping Channel (32) | 1 |
| Digital Channels (26) | 2 |
| Header Time (samples) | 3 |

**CAS DPOL Data SELECT Options**

## Result Type/Space

L[*n*], *n* = number of data samples

## Example

```
; Name       Units  Number Result Computations
"Digital"    ""   F2310    L[26]   CasDpolData(Aq.casdpoldata, 2, Aq.casdpol)
"LaserCurrent"   ""   F2000  F[1]  LIndex(F2310, 10)
"LaserCurrent"   "ma"  F2100  F[1] F2000 0.05 *
```

# CASDPOLSums(), CAS DPOL Sums

## Synopsis

**CASDPOLSums**(A, INTERVAL, STATE, BOARD)

| | |
|---|---|
| **A** | Acquisition tag for CAS DPOL data (tag). |
| **INTERVAL**[1] | Time interval value (integer). |
| STATE[1] | State variable (integer). |
| BOARD | The CAS DPOL board entry. |

## Description

This function sums up all CAS DPOL samples (i.e. similar to the OdSums function) from a data buffer. This summation is accrued for the specified frequency. At the end of the time interval the sums are returned through the result space and the internal summation values are cleared for the next summation period.

The **STATE** control variable is used to control the function operation mode. If the **STATE** control variable is a '0', then the summation is done every interval. If the **STATE** control variable is a '1', then the sums are accumulated. If the **STATE** control variable is a '2', this causes the last summation value to be held. Any other transition in the control variable, clears the internal summation and starts the accumulation process all over again.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name   Units Number Result Computations
"Counts" "" F300 L[30] CasDpolSums(Aq.casdpoldata, 1, 0, Bd.casdppol)
```

# CASPBPData(), CAS PBP Data Access

## Synopsis

**CASPPBData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for CAS PBP data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..3). |

## Description

This function allows access to individual items of the CAS PBP data block, including house data. The following table shows the different **SELECT** values for the different CAS PBP data fields. Please check the CAS PBP manual for further information. If you have more than 1 hz data, only the first sample is used.

| Data Field | SELECT |
|---|---|
| Data Count | 0 |
| Forward Counts | 1 |
| Backward Counts | 2 |
| Inter Particle Timer | 3 |

**CAS PBP Data SELECT Options**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name              Units   Number  Result  Computations
"CasPbpDataCount"   ""      F300    L[1]    CasPbpData(Aq.CasPbpData, 0)
```

# CIndex(), Character Element Access

## Synopsis

**CIndex**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula of an array of characters (char or string) (*n*≥1). |
| **INDEX**[1] | Index number of element being referenced (integer ≥ 0). |

## Description

This function allows access to individual elements of a character array. Returns a character referenced by **INDEX** in the string referenced by the formula **F**.

$$f = F[INDEX]$$

## Result Type/Space

D[1]

## Example

```
; Name              Units   Number  Result   Formulas
"CharacterIndex"    ""      F200    C[1]     CIndex(F2012, 5)
```

# CIPData(), CIP Data Access

## Synopsis

**CIPData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for CIP data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..25). |

## Description

This function allows access to individual items of the CIP data block, including house data. The following table shows the different **SELECT** values for the different CIP data fields. The function will returns the value of a user specified item from a CIP buffer. Please check the CIP manual for further information. To get the size spectrum for the CIP probe, use OdSums(), 1D Sums Data fields with the notation a[*m*] indicate analog data, where *m* is its position in the analog data array.

| Data Field | SELECT |
|---|---|
| Byte Count | 0 |
| Oversize Reject Count | 1 |
| DOF Reject Count | 2 |
| End Reject Count | 3 |
| Particle Counter | 4 |
| Seconds/Milliseconds | 5 |
| Hours/Minutes | 6 |
| Host Sync Counter | 7 |
| Reset Flag | 8 |
| Diode 1 Vol, a[0] | 10 |
| Diode 64 Vol, a[1] | 11 |
| Diode 32 Vol, a[2] | 12 |
| Pitot Press, a[3] | 13 |
| Static Press, a[4] | 14 |
| LWC How Wire Signal, a[5] | 15 |

**CIP Data SELECT Options**

| Data Field | SELECT |
|---|---|
| LWC Slave Monitor, a[6] | 16 |
| CIP DSP Board Temp, a[7] | 17 |
| Spare, a[8] | 18 |
| Spare, a[9] | 19 |
| Optional Temp 1, a[10] | 20 |
| Optional Temp 2, a[11] | 21 |
| Optional Temp 3, a[12] | 22 |
| Ambient Temp, a[13] | 23 |
| Laser Current, a[14] | 24 |
| Spare, a[15] | 25 |

**CIP Data SELECT Options (Continued)**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name       Units     Number  Result  Computations
"CipData"    "°C"      F300    F[1]    CipData(Aq.CipData, 21)
```

# CIPGSData(), CIPGS Data Access

## Synopsis

**CIPGSData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for CIPGS data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..25). |

## Description

This function allows access to individual items of the CIPGS data block, including house data. The following table shows the different **SELECT** values for the different CIPGS data fields. The function will returns the value of a user specified item from a CIPGS buffer. Please check the CIPGS manual for further information. To get the size spectrum for the CIPGS probe, use OdSums(), 1D Sums Data fields with the notation a[*m*] indicate analog data, where *m* is its position in the analog data array.

| Data Field | SELECT |
|---|---|
| Byte Count | 0 |
| Oversize Reject Count | 1 |
| DOF Reject Count | 2 |
| End Reject Count | 3 |
| Particle Counter | 4 |
| Seconds/Milliseconds | 5 |
| Hours/Minutes | 6 |
| Host Sync Counter | 7 |
| Reset Flag | 8 |
| a[0] | 10 |
| a[1] | 11 |
| a[2] | 12 |
| a[3] | 13 |
| a[4] | 14 |

**CIPGS Data SELECT Options**

| Data Field | SELECT |
|---|---|
| a[5] | 15 |
| a[6] | 16 |
| a[7] | 17 |
| a[8] | 18 |
| a[9] | 19 |
| a[10] | 20 |
| a[11] | 21 |
| a[12] | 22 |
| a[13] | 23 |
| a[14] | 24 |
| a[15] | 25 |

**CIPGS Data SELECT Options (Continued)**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name        Units      Number  Result  Computations
"LaserTemp"     ""       F2100    F[1]   CIPGSData(Aq.CIPGSData, 10)
```

# CIPGSInfo(), CIPGS Info Data Access

## Synopsis

**CIPGSInfo**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for CIPGS data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..2). |

## Description

This function allows access to individual items of the CIPGS info data block. The following table shows the different **SELECT** values for the different CIPGS info data fields. The function will returns the value of a user specified item from a CIPGS info buffer. Please check the CIPGS manual for further information.

The CIPGS setup reply data is provided via this function. This data changes for the first 10 seconds following the setup command. Then these values never change after this, unless the setup command is re-sent to the probe.

| Data Field | SELECT |
|---|---|
| Minimum Current | 0 |
| Maximum Current | 1 |
| Current | 2 |

**CIPGS Info Data SELECT Options**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name        Units     Number  Result  Computations
"MinimumCurrent" ""     F2500    F[64]  CIPGSInfo(Aq.CIPInfo, 0)
"MaximumCurrent" ""     F2501    F[64]  CIPGSInfo(Aq.CIPInfo, 1)
"Current"        ""     F2502    F[64]  CIPGSInfo(Aq.CIPInfo, 2)
```

# CIPGSMonitor(), CIPGS Monitor

## Synopsis

**CIPGSMonitor**(A, SHADOW)

| | |
|---|---|
| **A** | Acquisition tag for CIPGS image data (tag). |
| **SHADOW**[1] | Shadow level (integer: 0x01, 0x02, 0x03). |

## Description

This function allows us to monitor set (spike up values in the data) or cleared (spike down values in the data) for all vertical lines from the image data. There are 64 diodes across the entire beam and hence the fixed result of 64 values.

The SHADOW parameter can be used to select minimum, middle and maximum shadow levels for the Monitor function to look for in the data.

The primary trigger for this function should 1 hz synchronous trigger and the secondary trigger should be “CIPGS Image”, buffer frequency, board for appropriate CIP GS probe.

## Result Type/Space

D[64], number of diodes/pixels for the probe.

## Example

```
; Name       Units     Number  Result  Computations
Trigger = "Sync" 1 None "CIPGS Image" 10 cipgs
"CIPGSMonitor" ""    F2003    F[64]  CIPGSMonitor(Aq.CIPGSImage, 0x02)
"CIPGSBin"     ""    F2004    F[64]  Set(1, 1, 64)
```

# CIPGSSums(), CIPGS Sums

## Synopsis

**CIPGSSums**(PROBE, A, MODE, FREQUENCY)
**CIPGSSums**(A, MODE, PROBE, INTERVAL, STATE, PIXELSIZE)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **A** | Acquisition tag for CIPGS image data (tag). |
| **STATE**[1] | State (integer). |
| **PIXELSIZE**[1] | Pixel size (integer). |
| **FREQUENCY**[1] | Integration frequency (integer). |
| **INTERVAL**[1] | Integration interval (integer). |
| **MODE**[1] | Summation mode (integer). |

## Description

This function builds up an approximation of the CIPGS spectrum using the image data. These particles are summed up and normalized using the delta particle count divided by the delta time from the actual time bars times the FREQUENCY.

The method used for summation is the area of the particle. Other methods are possible but not currently implemented.

The MODE, STATE and PIXEL SIZE variables are unused and reserved for future upgrades of the summation techniques.

The primary trigger should be 1 hz synchronous data and secondary trigger for “CIPGS Image”, buffer frequency, CIPGS board name.

## Result Type/Space

D[*62*], There are 62 number of channels in probe entry (**PROBE**)

## Example

```
; Name    Units  Number  Result   Computations
Trigger = "Sync" 1 None "CIPGS Image" 10 cipgs
"CIPGSCounts" "" F2002 F[62] CipGSSums(Pr.cipgs, Aq.CIPGSImage, 1, 1)
```

# CIPMonitor(), CIP Monitor

## Synopsis

**CIPMonitor**(A)

**A** Acquisition tag for CIP image data (tag).

## Description

This function allows us to monitor set (spike up values in the data) or cleared (spike down values in the data) for all vertical lines from the image data. There are 64 diodes across the entire beam and hence the fixed result of 64 values.

The primary trigger for this function should 1 hz synchronous trigger and the secondary trigger should be “CIP Image”, buffer frequency, board for appropriate CIP probe.

## Result Type/Space

D[64], number of diodes/pixels for the probe.

## Example

```
; Name       Units     Number  Result  Computations
Trigger = "Sync" 1 None "CIP Image" 10 cip
"CIPMonitor" ""    F2003    F[64]  CIPMonitor(Aq.CIPImage)
"CIPBin"     ""    F2004    F[64]  Set(1, 1, 64)
```

# CIPSums(), CIP Sums

## Synopsis

**CIPSums**(PROBE, A, MODE, FREQUENCY)
**CIPSums**(A, MODE, PROBE, INTERVAL, STATE, PIXELSIZE)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **A** | Acquisition tag for CIP image data (tag). |
| **STATE**[1] | State (integer). |
| **PIXELSIZE**[1] | Pixel size (integer). |
| **FREQUENCY**[1] | Integration frequency (integer). |
| **INTERVAL**[1] | Integration interval (integer). |
| **MODE**[1] | Summation mode (integer). |

## Description

This function builds up an approximation of the CIP spectrum using the image data. These particles are summed up and normalized using the delta particle count divided by the delta time from the actual time bars.

The method used for summation is the area of the particle. Other methods are possible but not currently implemented.

The MODE, STATE and PIXEL SIZE variables are unused and reserved for future upgrades of the summation techniques.

The primary trigger should be 1 hz synchronous data and secondary trigger for “CIP Image”, buffer frequency, CIP board name.

## Result Type/Space

D[*62*], There are 62 number of channels in probe entry (**PROBE**)

## Example

```
; Name    Units  Number  Result   Computations
Trigger = "Sync" 1 None "CIP Image" 10 cip
"CIPCounts" "" F2002 F[62] CipSums(Pr.cip, Aq.CIPImage, 1, 1)
```

# Cmd1D(), Command 1D

## Synopsis

**Cmd1D**(A)
**A** Acquisition tag for 1D data (tag).
*Note: Deprecated [M300 Replacement function - See "OdCmd(), 1D Command"]*

## Description

This function retrieves and returns the command byte from 1D data.

## Result Type/Space

I[1]

## Example

```
; Name      Units  Number  Result  Computations
"Cmd1D"     ""     F100    I[1]    Cmd1D(Aq.fssp)
```

# Co1DCmd(), Control 1D Command

## Synopsis

**Co1DCmd**(BOARD, COMMAND)

| | |
|---|---|
| **BOARD** | Board name for of 1D Interface board (board) |
| **COMMAND**[1] | 1D command byte to be sent to board (integer). |

## Description

This function sets up the command byte for the specified 1D type board. For example, this is used to change FSSP range or the PCASP pump on/off.

This function supports 1D, 1D256, CAMAC1D, CAMAC1D256, SPP board type.

CDP doesn't allow for range change, so it's not supported!

## Result Type/Space

I[1]

## Example

```
; Name                Units  Number  Result  Computations
"FSSPRangeControl"    ""     F200    I[1]    Co1DCmd(Bd.FSSP, 0x01)
```

# Co2DTAS(), Control 2D TAS

## Synopsis

**Co2DTAS**(BOARD, FREQUENCY)

| | |
|---|---|
| **BOARD** | Board name for 2D Mono Interface (board). |
| **FREQUENCY**[1] | Board frequency in MHz. |

## Description

Control 2D TAS. This control function computes and updates the multiply and divide factors used by digital frequency space generator on the 2D Mono interface [this control is necessary to keep the 2D Mono images from being distorted]. This frequency generates the TAS clock that strobes the 2D Mono data. Upon successful completion, this function returns the board frequency specified in the function, otherwise it will return zero.

## Result Type/Space

D[1]

## Example

```
; Name          Units     Number  Result  Computations
"Control2DTAS" "MHz"     F101    F[1]    Co2DTas(Bd.2dc, 0.800)
```

# Co2GCmd(), Control 2D Grey Command

## Synopsis

**Co2GCmd**(BOARD, COMMAND)

| | |
|---|---|
| **BOARD** | Board name for 2D Grey interface (board). |
| **COMMAND**[1] | 2D Grey command (integer). |

## Description

This function sends the 2D Grey Command byte for the user specified board. Check the 2D Grey probe manual for the specified values that can be use for the commands out to the probe. Upon successful completion, this function returns the user specified **COMMAND**, otherwise it returns zero.

## Result Type/Space

I[1]

## Example

```
; Name      Units  Number  Result  Computations
"Co2GCmd"   ""     F2001   I[1]    Co2GCmd(Bd.2dg, 0x01)
```

# Co2GTAS(), Control 2D Grey TAS

## Synopsis

**Co2GTAS**(BOARD, FREQUENCY)

| | |
|---|---|
| **BOARD** | Board name for 2D Grey Interface (board). |
| **FREQUENCY**[1] | Board frequency in MHz. |

## Description

Control 2G TAS. This control function computes and updates the multiply and divide factors used by digital frequency space generator on the 2D Grey interface [this control is necessary to keep the 2D Grey images from being distorted]. This frequency generates the TAS clock that strobes the 2D Grey data. Upon successful completion, this function returns the board frequency specified in the function, otherwise it will return zero.

## Result Type/Space

D[1]

## Example

```
; Name           Units    Number  Result  Computations
"Control2GTAS"   "MHz"    F101    F[1]    Co2GTas(Bd.2dg, 0.800)
```

# CoATDAQ141X(), Control ATDAQ141X

## Synopsis

**CoATDAQ141X**(BOARD, VOLTAGE, CHANNEL)

| | |
|---|---|
| **BOARD** | Board name for ATDAQ141X interface (board). |
| **VOLTAGE**[1] | Analog output voltage (in volts). |
| **CHANNEL**[1] | Output channel (integer: 0,1). |

## Description

Controls the output voltages for the ATDAQ141X Board. This function is used to control the analog voltage value output for a specific channel.

The function returns an integer containing the calculated analog output voltage. Check the ATDAQ141X manual for further details.

## Result Type/Space

I[1]

## Example

```
; Name              Units  Number  Result  Computations
"ControlATDAQ141X"   ""     F501    I[1]    CoATDAQ141X(Bd.ATDAQ141X, 2, 1)
```

# CoCIPGSTAS(), Control CIPGS TAS

## Synopsis

**CoCIPGSTAS**(BOARD, FREQUENCY)

| | |
|---|---|
| **BOARD** | Board name for CIPGS Interface (board). |
| **FREQUENCY**[1] | Board frequency in MHz. |

## Description

Control CIPGS TAS. This control function computes and updates the frequency used by digital frequency space generator on the CIPGS interface (this control is necessary to keep the CIPGS images from being distorted). This frequency generates the TAS clock that strobes the CIPGS data. Upon successful completion, this function returns the board frequency specified in the function, otherwise it will return zero.

## Result Type/Space

D[1]

## Example

```
; Name            Units    Number  Result  Computations
"CIPGSProbeTAS"   "MHz"    F101    F[1]    CoCIPGSTAS(Bd.cipgs, 0.800)
```

# CoCIPTAS(), Control CIP TAS

## Synopsis

**CoCIPTAS**(BOARD, FREQUENCY)

| | |
|---|---|
| **BOARD** | Board name for CIP Interface (board). |
| **FREQUENCY**[1] | Board frequency in MHz. |

## Description

Control CIP TAS. This control function computes and updates the frequency used by digital frequency space generator on the CIP interface (this control is necessary to keep the CIP images from being distorted). This frequency generates the TAS clock that strobes the CIP data. Upon successful completion, this function returns the board frequency specified in the function, otherwise it will return zero.

## Result Type/Space

D[1]

## Example

```
; Name            Units    Number  Result  Computations
"CIPProbeTAS"    "MHz"    F101    F[1]    CoCIPTAS(Bd.cip, 0.800)
```

# CoCYDDA(), Control CYDDA

## Synopsis

**CoCYDDA**(BOARD, VOLTAGE, CHANNEL, MODE)

| | |
|---|---|
| **BOARD** | Board name for CYDDA interface (board). |
| **VOLTAGE**[1] | Analog output voltage (in volts). |
| **CHANNEL**[1] | Output channel (integer: 0..15). |
| **MODE**[1] | Mode. |

## Description

Controls the output voltages for the CYDDA Board. This function is used to control the analog voltage value output for a specific channel. The **MODE** parameter can be used to select the appropriate voltage range. Valid mode values are ±10, ±5, and ±2.5. If **MODE** is set to zero, the board will perform a simultaneous update specified by the CYDDA user manual.

| Type | Mode | Output Range (Volts) |
|---|---|---|
| BIPOLAR | -10 | ±10 |
| | -5 | ±5 |
| | -2.5 | ±2,5 |
| UNIPOLAR | 10 | 10 |
| | 5 | 5 |
| | 2.5 | 2.5 |
| | 0 | Simultaneous read and update. |

**Mode Selection Options**

The function returns an integer containing the calculated analog output voltage. If mode is set to zero, the function will return a one. Check the CYDDA manual for further details.

## Result Type/Space

I[1]

## Example

```
; Name          Units  Number  Result   Computations
"ControlCYDDA"   ""     F501    I[1]    CoCYDDA(Bd.CYDDA, 10, 2, 5)
```

# CoDo(), Control Digital Output

## Synopsis

**CoDo**(BOARD, PORT, BIT, VALUE)

| | |
|---|---|
| **BOARD** | Board name for digital output interface (board). |
| **PORT**[1] | Port number (integer) |
| **BIT**[1] | Bit position of event bit to be set (integer: 0..7) |
| **VALUE**[1] | Set mode (on/off) (integer: 0 or 1) |

## Description

This function reads in the current state from the digital board. Based on the value of **BIT**, the function will replace that bit position of the control byte to the value of **VALUE**. Once that is complete, the function will output the new byte to the digital board. Function will return the new byte being output to the digital board. This function can be used for the ATDQ141X, CYPDISO, CYDIO24, PCIDAC, PMF and DT2817 boards.

## Result Type/Spaces

L[1]

## Example

```
; Name            Units  Number  Result  Computations
"DigitalOutput"   ""     F101    L[1]    CoDo(Bd.DT2817, 1, 5, 1)
```

# CoDT2817(), Control DT2817

## Synopsis

**CoDT2817**(BOARD, PORT, BIT, VALUE)

| | |
|---|---|
| **BOARD** | Board name for DT2817interface (board). |
| **PORT**[1] | Port number (integer) |
| **BIT**[1] | Bit position of event bit to be set (integer: 0..7) |
| **VALUE**[1] | Set mode (on/off) (integer: 0 or 1) |

*Note: Deprecated [M300 Replacement function - See "CoDo(), Control Digital Output"]*

## Description

This function reads in the current state from the digital board. Based on the value of **BIT**, the function will replace that bit position of the control byte to the value of **VALUE**. Once that is complete, the function will output the new byte to the digital board. Function will return the new byte being output to the digital board. This function can be used for the DT2817 board.

## Result Type/Spaces

L[1]

## Example

```
; Name          Units  Number  Result   Computations
"DigitalOutput"  ""     F101    L[1]     CoDT2817(Bd.DT2817, 1, 5, 1)
```

# CoFile(), Control File

## Synopsis

**CoFile**(STATE)
**STATE**[1] State option (on/off) (integer: 0 or 1).

## Description

This function is used to automatically turn M300 recording on/off. A one in the **STATE** argument forces the M300 into record mode, while a zero forces the M300 to stop recording.

## Result Type/Space

D[1]

## Example

```
; Name              Units  Number  Result   Computations
"RecordControl"      ""     F200    F[1]    CoFile(F100)
```

# Color(), Color

## Synopsis

**Color**(COLORNAME)
**COLORNAME**[*n*] Valid color name (*n*>0) (string).

## Description

This function is used to return the M300 color value based on the string passed. Each M300 color listed in Color System has an associated long integer value. This function allows the user to reference colors based on their names, rather than their long integer values. If **COLORNAME** does not match any M300 color strings, the M300 will assume the closest string matched.

## Result Type/Space

L[1]

## Example

```
; Name         Units  Number  Result  Computations
"AlarmColor"   ""     F201    L[1]    Color("red")
```

# Comb(), Combine Arrays

## Synopsis

**Comb**(A, AINDEX, AELEMENTS, B, BINDEX, BELEMENTS)

| | |
|---|---|
| **A**[*m*] | Formula of an array of values (*m*≥1). |
| **AINDEX**[1] | First element of **A** to be combined (integer). |
| **AELEMENTS**[1] | Number of elements in **A** to be combined (integer). |
| **B**[*p*] | Formula of an array of values (*p*≥1). |
| **BINDEX**[1] | First element of **B** to be combined (integer). |
| **BELEMENTS**[1] | Number of elements in **B** to be combined (integer). |

## Description

This function build a new array from segments of two different arrays. For each of the arrays specified, the user may pick an index and number of elements to use.

## Result Type/Space

D[*n*], *n* = [**AELEMENTS**] + [**BELEMENTS**]

## Example

```
; Name             Units  Number  Result  Computations
"CombineArrays"    ""     F101    F[15]   Comb(F99, 2, 5, F100, 0, 10)
```

# Concs(), Concentrations

## Synopsis

**Concs**(PROBE, F, CFAC, TAS, MODE)
**Concs**(F, PROBE, RANGE, CFAC, TAS, INTERVAL, MODE)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **F**[*n*] | Formula for the sums array, channel samples ($n \geq 1$). |
| **CFAC**[1] | Correction factor. |
| **TAS**[1] | True air speed. |
| **MODE**[1] | Mode value used to specify divide correction value (integer). |
| **RANGE**[1] | Selected range value. |
| **INTERVAL**[1] | Interval of summation used by sums routine (integer). |

## Description

This function uses the summed up channel counts and the probe definition table to compute concentrations. The result is typically used for mean, median, mode and total concentration calculations as well as X vs. Y display plots. This function should be 'refreshed' at the same time interval as the summation routine generates data, so as to eliminate redundant calculations on the same input data.

Please note that if the TAS is in (m/s) and the sample area is (mm2) the resulting concentrations will be in (#/cc). The correction factor can be used to return the resulting concentrations in different units as well as using the velocity acceptance ratio. For example, a correction factor of 1000 will give the result in (#/mm3), a correction factor of 0.001 will give the result in (#/l) and a correction factor of 1.0e-6 will give the result in (#/m3).

The 'SAREA', 'dD' and 'dLOGD' originate from the user specified channel files via the probe number/name. The 'BUFLIFE' and 'SYSFREQ' refer to the values entered in the system table. The SYSFREQ is associated with the system frequency in the time data. This comes from the frequency values in the system board entry. The BUFLIFE is associated with the buffer life in the time data. In the M300 system, the buffer life and system frequency are the same (for synchronous buffers). The following formula summarizes the computation.

$$f[i] = \frac{F[i]}{SAREA[i, RANGE] \cdot VALUE \cdot TAS \cdot \frac{BUFLIFE}{SYSFREQ} \cdot CFAC}$$

where i = 0 ... n, channel number
if (MODE==0) then VALUE = 1
if (MODE==1) then VALUE = dD[i, RANGE]
if (MODE==2) then VALUE = dLOGD[i, RANGE]

## Result

D[*n*], *n* = min(*n, probe channels*)

## Example

```
; Name          Units  Number  Result  Computations
Concentrations ""     F300    F[15]   CONCS(F101, P1, F200, 1.0, F10, 1, 1)
```

# CoPCIDACDA(), Control PCIDAC D/A Voltages

## Synopsis

**CoPCIDACDA**(BOARD, VOLTAGE, CHANNEL)

| | |
|---|---|
| **BOARD** | Board name for PCIDAC interface (board). |
| **VOLTAGE**[1] | Analog output voltage (in volts). |
| **CHANNEL**[1] | Output channel (integer: 0,1). |

## Description

Controls the output voltages for the PCIDAC Board. This function is used to control the analog voltage value output for a specific channel.

The function returns an integer containing the calculated analog output voltage. Check the PCIDAC manual for further details.

## Result Type/Space

I[1]

## Example

```
; Name       Units  Number  Result   Computations
"ControlV0"   ""     F501    i[1]    CoPCIDACDA(Bd.pcidac16, 1.0, 0)
```

# CoPMFDA(), Control PMF D/A Voltages

## Synopsis

**CoPMFDA**(BOARD, VOLTAGE, CHANNEL)

| | |
|---|---|
| **BOARD** | Board name for PMF interface (board). |
| **VOLTAGE**[1] | Analog output voltage (in volts). |
| **CHANNEL**[1] | Output channel (integer: 0,1). |

## Description

Controls the output voltages for the PMF Board. This function is used to control the analog voltage value output for a specific channel.

The function returns an integer containing the calculated analog output voltage. Check the PMF manual for further details.

## Result Type/Space

I[1]

## Example

```
; Name        Units  Number  Result   Computations
"ControlV0"    ""     F501    i[1]    CoPMFDA(Bd.pmf, 1.0, 0)
```

# Copy(), Copy Arrays

## Synopsis

**Copy**(F, INDEX, ELEMENTS)

| | |
|---|---|
| **F**[*m*] | Formula for source array (*m*≥1). |
| **INDEX**[1] | Index of starting element (integer > 0). |
| **ELEMENTS**[1] | Number of elements to copy (integer). |

## Description

This function can be used to copy data from one formula array to another. The starting and ending position can be specified via the starting index and the number of elements to copy.

$$f[i] = F[INDEX + i]$$

where $i = 0 \ldots (ELEMENTS - 1)$

## Result Type/Space

D[*n*], *n* = **ELEMENTS**

## Example

```
; Name        Units  Number  Result   Computations
"CopyArray"    ""     F201    F[10]   Copy(F200, 5, 10)
```

# CoQuit(), Control Quit

## Synopsis

**CoQuit**(STATE)
**STATE**[1] State option (integer: 0 or 1).

## Description

This function is used to automate the termination of the M300 software. If the **STATE** is zero this function will do nothing. If the **STATE** is non-zero (1), the M300 will close all open files, stop the acquisition process and close.

0- Do nothing.
1- Quit.

## Result Type/Space

I[1]

## Example

```
; Name      Units  Number  Result   Computations
"QuitM300"   ""     F1000    I[1]   CoQuit(F2005)
```

# CoRTI802(), Control RTI802

## Synopsis

**CoRTI802**(BOARD, VOLTAGE, CHANNEL, MODE)

| | |
|---|---|
| **BOARD** | Board name for RT1802 interface (board). |
| **VOLTAGE**[1] | Analog output voltage. |
| **CHANNEL**[1] | Output channel (integer: 0..7). |
| **MODE**[1] | Specified voltage range (integer: 0 or 1). |

## Description

This function is used to control the analog voltage value output for a specific channel. The mode parameter can be used to select the appropriate voltage range. For the RTI802 board, valid values are zero for -10 to +10 range and one for 0 to +10 range. Upon successful completion, function returns an integer containing the computed analog output voltage, otherwise -1.

## Result Type/Space

D[1]

## Example

```
; Name       Units  Number  Result  Computations
"Channel03"   ""     F203    F[1]    CoRTI802(Bd.RTI802, F103, 3, 0)
```

# CoSeaDA(), Control Sea Voltage

## Synopsis

**CoSeaDA**(BOARD, VOLTAGE)

| | |
|---|---|
| **BOARD** | Board name for SEADA board (board). |
| **VOLTAGE**[1] | Analog output voltage. |

## Description

This function is used to control the analog output voltage for the SEA D/A board.

## Result Type/Space

D[1]

## Example

```
; Name    Units  Number  Result   Computations
"SeaDA"    ""     F4000   F[1]    CoSeaDA(Bd.SeaDA, 1.35)
```

# CoShutdown(), Control Shutdown

## Synopsis

**CoShutdown**(STATE)
**STATE**[1] State option (integer) (0-1).

## Description

This function is used to automate the shutdown of the M300 DAS. If **STATE** is non-zero, the M300 will terminate and force the entire system to shutdown, including the Photon GUI and QNX.

0- Do nothing.

1- Shutdown QNX4/Photon.

## Result Type/Space

I[1]

## Example

```
; Name       Units  Number   Result  Computations
"Shutdown"    ""     F4000    I[1]   CoShutdown(F101)
```

# CountBy(), Count by

## Synopsis

**CountBy**(DATA, BINS)

| | |
|---|---|
| **DATA**[m] | Formula of an array for data values ($m \geq 1$). |
| **BINS**[p] | Formula of an array for bins values ($p \geq 1$). |

## Description

This function is used to count the data values into bins (by count).
The bins must be sorted since we use a binary search to place the data into the appropriate bin.
Typical use of this function is for PBP data for some of the probes that support it.

## Result Type/Space

D[p]

## Example

```
; Name       Units  Number  Result  Computations
"CDPPBPIPTHist" ""  F3811   F[28]   CountBy(F3804, F3810)
```

# CountEdges(), Count Edges

## Synopsis

**CountEdges**(F)
**CountEdges**(F, DIR)

| | |
|---|---|
| **F**[m] | Formula array for integer data values ($m \geq 1$). |
| **DIR**[1] | Direction value. |

## Description

This function is count the rising or falling edges for an array of values (F).

The formula values must be an integer array of zeros and ones.

Each second, the function returns the number of edges and it also remembers the previous state so it can determine the correct number of edges over time.

Default is to count rising edges, DIR of 0. To count falling edges use DIR of 1.

## Result Type/Space

D[1], F[1], L[1], l[1], I[1], i[1]

## Example

```
; Name          Units  Number  Result  Computations
"FlareEventCount" ""  F3811     L[1]   CountEdges(F3804)
```

# Cumulative(), Cumulative

## Synopsis

**Cumulative**(A)
**A**[*n*] Formula of an array of values (*n*≥2).

## Description

The resulting array of values is explained via the following equation.

$$f[0] = A[0]$$

$$f[i] = A[i-1] + A[i]$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result

D[*n*]

## Example

```
; Name                      Units  Number  Result  Computations
"OAP260XLWCCumulative"    "g/m3"   F9200   F[62]  Cumulative(F9050)
```

# Date(), Date String

## Synopsis

**Date**(A0)

**A0** Acquisition tag for date/time data (tag).

***Note: Because the M300 now uses the M300 buffer time for data retrieval, the Acquisition tag argument is no longer used. It is still needed to maintain backward compatibility with the M200.***

## Description

This function converts the M300 buffer date/time into an ASCII string for display purposes. The string is in the form of '**[YY]YY-MM-DD**', where '**YYYY**' is the year, '**MM**' is the month and '**DD**' is the days. Function will return a character array (string) of size *n* containing the characters that make up the date string that was retrieved from the data. The size of *n* is dependant on how many characters are required to represent that particular date. Usually this size is 10 or 12.

## Result Type/Space

S[*n*], *n* = 10 or 12

## Example

```
; Name         Units  Number  Result   Computations
"DateString"    ""     F1      S[12]   Date(A0)
```

# DateTime(), Date Time String

## Synopsis

**DateTime**(A0)

**A0** Acquisition tag for date/time data (tag).

***Note: Because the M300 now uses the M300 buffer time for data retrieval, the Acquisition tag argument (A0) is no longer used. It is still needed to maintain backward compatibility with the M200.***

## Description

This function converts the date/time M300 buffer time into an ASCII string for display purposes. The string is in the form of '**[YY]YY-MM-DD HH:MM:SS**', where '**YYYY**' is the year, '**MM**' is the month, '**DD**' is the days, **HH** is the hours, **MM** is the minutes and **SS** is the seconds. Function will return a character array of size *n* containing the characters that make up the date and time string that was retrieved from the data. The size of *n* is dependant on how many characters are required to represent that particular date and time. Usually this size is 18 or 20.

## Result Type/Space

S[*n*], *n* = 18 or 20

## Example

```
; Name         Units  Number  Result   Computations
"DateTime"      ""     F1      S[20]   DateTime(A0)
```

# DayOfYear(), Day of Year

## Synopsis

**DayOfYear**(A0)
**DayOfYear**(YEAR, MONTH, DAY)

| | |
|---|---|
| **A0** | Acquisition tag for date/time data (tag). |
| **YEAR**[**1**] | Year (integer). |
| **MONTH**[*1*] | Month (integer). |
| **DAY**[*1*] | Day (integer). |

## Description

This function is used to return day of year. There are two formats used for this function.

If you specify a tag number the system uses the time from the buffer to compute day of year.

The other option to computing day of year is to pass YEAR, MONTH and DAY as parameters to the function.

## Result Type/Space

L[1], l[1], I[1], i[1]

## Example

```
; Name        Units  Number  Result  Computations
"DayOfYear"     ""    F300    I[1]   DayOfYear(A0)
```

# Delay(), Delay

## Synopsis

**Delay**(F, CYCLES)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **CYCLES**[1] | Number of seconds (integer ≥ 1). |

## Description

This function is used to delay the value of a formula by the specified number of cycles. The system keeps the desired number of values in memory and gives the delayed value as the return value. The function returns zero until the number of **CYCLES** have elapsed by. The current trigger will affect the meaning of **CYCLES**.

## Result Type/Space

D[*n*]

## Example

```
; Name         Units   Number  Result   Computations
"1MinDelay"     ""      F200    F[10]   Delay(F200, 60)
```

# Delta(), Delta

## Synopsis

**Delta**(F, CYCLES)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **CYCLES**[1] | Number of seconds (integer ≥ 1). |

## Description

This function computes the difference of the current formula value and the formula value observed **CYCLES** previously. The function returns zero until the number of **CYCLES** have elapsed. The function returns zero until the number of **CYCLES** have elapsed. The current trigger will affect the meaning of cycles.

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result   Computations
"Delta"      ""     F200    F[10]   Delta(F100, 60)
```

# DewPointToRH(), Dew Point to Relative Humidity

## Synopsis

**DewPointToRH**(DP, TEMP)
**DP**[*n*] Formula for dewpoint (*n*>0).
**TEMP**[1] Formula for outside air temperature value.

## Description

This function calculates an approximation of the relative humidity based on the **DP** (dewpoint) and **TEMP** (outside air temperature) arguments passed. The following formulas demonstrate the calculations performed.

$$f[i] = 100 \cdot \frac{\left(6.11 \cdot 10^{\frac{7.567 DP[i]}{239.7 + DP[i]}}\right)}{\left(6.11 \cdot 10^{\frac{7.567 \cdot TEMP}{239.7 + TEMP}}\right)}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name      Units   Number  Result  Computations
"RH"         "%"     F4008   F[1]   DewPointToRH(F3005, F3006)
```

# Dfault(), Default Value

## Synopsis

**Dfault**(F, LOW, HIGH, DEFAULT)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **LOW**[1] | Lower limit value. |
| **HIGH**[1] | Upper limit value. |
| **DEFAULT**[1] | Default value. |

## Description

This function is used to check limits on an array of values and return the default value if the input values are outside specified limits.

$$\text{if } ((F[i] < LOW) \wedge (F[i] > HIGH)) \text{ then } f[i] = DEFAULT$$
$$\text{else } f[i] = F[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name         Units  Number  Result  Computations
"Default"     ""     F100    I[10]   Dfault(F100, 1, 100, 0)
```

# DIndex(), Double Element Access

## Synopsis

**DIndex**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (double) (*n*≥1). |
| **INDEX**[1] | Index number of element being referenced (integer ≥ 0). |

## Description

Uses the element **INDEX** to reference that particular value in an array of (double) data.

This function returns a single double precision floating point value referenced by **INDEX** in the array **F**.

$$f = F[INDEX]$$

## Result Type/Space

D[1]

## Example

```
; Name          Units  Number  Result  Computations
"DoubleIndex"    ""     F300    D[1]    DIndex(F100, 24)
```

# DirData(), Directory Data

## Synopsis

**DirData**(A, SELECT)
**A** Acquisition tag for data (tag).
**SELECT**[1] Directory field select (integer: 0...8).

## Description

This function is used to retrieve specific data from a directory structure referenced by the acquisition tag **A**. The **SELECT** argument is used to select which data parameter in the directory structure is retrieved.

| SELECT | Directory Data |
|---|---|
| 0 | Offset |
| 1 | Number of bytes |
| 2 | Samples |
| 3 | Bytes Per Sample |
| 4 | Type |
| 5 | Para1 |
| 6 | Para2 |
| 7 | Para3 |
| 8 | Address |

**SELECT**

## Result Type/Space

L[1]

## Example

```
; Name          Units   Number  Result   Computations
"DirectoryData"  ""      F205    L[1]     DirData(A100, 8)
```

# Div(), Divide

## Synopsis

**Div**(A, B)

**A**[*m*] Formula of an array of values (*m*≥1).

**B**[*p*] Formula of an array of values (B[*i*] ≠ 0) (*p*≥1).

***Note: Deprecated [M300 Replacement function - See "/, Divide"]***

## Description

This function returns an array of values representing the division of the two given arrays, element by element. This function uses Interpolation [**See Interpolation**]. Although the M300 is designed to trap most errors including user errors, care should be exercised to avoid division by zero. The following formula summarizes the calculations.

$$f[i] = \frac{A[i]}{B[i]}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*,*p*)

## Example

```
; Name      Units  Number  Result   Computations
"Divide"     ""     F300    F[15]   Div(F200, F201)
```

# DToF(), Double to Float

## Synopsis

**DToF**(A)

**A** Acquisition tag for double data (tag).

## Description

This function takes eight bytes (double) and converts them to a float number. It will return an array of size *n* containing the converted double to float values, where *n* is the number of samples from the directory referenced by **A** (Acquisition tag name/number).

The M200 system did not have support for double types. This function was necessary to get the double data into a float format. With the M300 system, this function is not necessary. You should be able to just use the tag number and get the number in double format onto the stack.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name           Units  Number  Result   Computations
"DoubleToFloat"  ""      F100    F[45]   DToF(A100)
```

# Eq(), Equal

## Synopsis

**Eq**(A, B, FTRUE, FFALSE)

| | |
|---|---|
| **A**[1] | First value used in comparison. |
| **B**[1] | Second value used in comparison. |
| **FTRUE**[*m*] | Formula for true value ($m \geq 1$). |
| **FFALSE**[*p*] | Formula for false value ($p \geq 1$). |

## Description

This function compares two values and it returns the value of true formula **FTRUE**, if the first value **A** is equal to the second value **B,** otherwise the value of false formula **FFALSE** is returned. This function uses Interpolation [**See Interpolation**].

$$\text{if } A = B \text{ then } f[i] = FTRUE[i]$$
$$\text{else } f[i] = FFALSE[i]$$

## Result

D[*n*], if A = B, $n = m$, else $n = p$

## Example

```
; Name    Units  Number  Result    Computations
"Equal"    ""     F130    F[15]    Eq(F199, F200, F400, F401)
```

# Esi(), Vapor Pressure of Water with Respect to Ice

## Synopsis

**Esi**(T)

**T**[*n*] Formula for temperature in degrees celsius (*n*≥1).

## Description

This function returns vapor pressure of water with respect to ice in mb from temperature in degrees celsius.

The following computation is performed for each temperature passed.

esi =10^(-9.09718*(273.16/(T+273.16)-1)-3.56654*LOG10(273.16/(T+273.16))+0.876793*(1-(T+273.16)/273.16)+LOG10(6.1071))

## Result

D[*n*]

## Example

```
; Name    Units  Number  Result    Computations
"ESI"      "mb"   F130    F[1]    Esi(F199)
```

# Esw(), Vapor Pressure of Water with Respect to Water

## Synopsis

**Esi**(T)
**T**[*n*] Formula for temperature in degrees celsius (*n*≥1).

## Description

This function returns vapor pressure of water with respect to water in mb from temperature in degrees celsius.

The following computation is performed for each temperature passed.

esw =10 ^ (-7.90298*(373.16/(T+273.15)-1)+5.02808*LOG10(373.16/(T+273.15))-0.00000013816 * (10 ^ (11.344*(1-(T+273.15)/373.16))-1)+0.0081328 * (10 ^ (-3.49149*(373.16/(T+273.15)))-1) + LOG10(1013.246))

## Result

D[*n*]

## Example

```
; Name    Units  Number  Result    Computations
"ESW"      "mb"   F130    F[1]    Esw(F199)
```

# EvtStr(), Event String

## Synopsis

**EvtStr**(EVENT, BIT, STR0, STR1)

| | |
|---|---|
| **EVENT** | Acquisition tag or formula number for event (tag). |
| **BIT**[1] | Event bit number (integer: 0..31). |
| **STR0**[*m*] | Zero (0, false) value string (*m*≥1) (string). |
| **STR1**[*p*] | One (1, true) value string (*p*≥1) (string). |

## Description

This function converts the state of one digital event bit into one of two string values. This is normally done to obtain a string that can be displayed in the text table. If the specified bit is a zero (false), **STR0** is returned. If the specified bit is a one (true), **STR1** is returned. The returned character array is of size *n* containing either **STR0** or **STR1**, where s[*n*] is the maximum size between **STR0** and **STR1**.

## Result Type/Space

S[*n*], $n = \max(m, p)$

## Example

```
; Name          Units  Number  Result   Computations
"EventString"    ""     F122    C[9]    EvtStr(F121, 1, "ARMED", "DISARMED")
```

# EvtVal(), Event Value

## Synopsis

**EvtVal**(EVENT, BIT, STATE)

| | |
|---|---|
| **EVENT**[1] | Acquisition tag for event (tag). |
| **BIT**[1] | Bit number of event bit (integer: 0..31). |
| **STATE**[1] | State option (integer: 0 or 1). |

## Description

This function provides the ability to obtain the value (0 or 1) of an event bit. The resulting state can be toggled if the **STATE** value is a zero. The returned integer value (1 or 0) contains the value of the event bit given by **BIT**.

## Result Type/Space

D[1]

## Example

```
; Name         Units  Number  Result   Computations
"EventValue"    ""     F300    L[1]    EvtVal(A100, 4, 1)
```

# FalconData(), Falcon Data

## Synopsis

**FalconData**(A, TITLE)

| | |
|---|---|
| **A** | Acquisition tag for Falcon data (tag). |
| **TITLE**[1] | Title indicating which falcon data field to retrieve (string). |

## Description

This function is used to allow access to individual elements of the falcon data buffer. The data returned is dependent on the source specified by the **TITLE** and the information entered in the

falcon data table. These values can then be used by different tables for further computations and display.

## Result Type/Space

D[1]

## Example

```
; Name         Units  Number  Result   Computations
"FalconData"    ""     F100    I[1]    FalconData(A100, "RAD-TEMP")
```

# FalconDay(), Falcon Day

## Synopsis

**FalconDay**(A, OFFSET)

| | |
|---|---|
| **A** | Acquisition tag for Falcon data (tag). |
| **OFFSET**[1] | Offset for the time data in the falcon buffer block (integer). |

## Description

This function is used to decode the time data from the falcon data buffer and return the current day. The falcon day returned by this function can then be used by the text table for display of the falcon day value.

## Result Type/Space

D[1]

## Example

```
; Name        Units  Number  Result   Computations
"FalconDay"    ""     F100    I[1]    FalconDay(A100, 5)
```

# FalconTime(), Falcon Time

## Synopsis

**FalconTime**(A, OFFSET)

| | |
|---|---|
| **A** | Acquisition tag for Falcon data (tag). |
| **OFFSET**[1] | Offset for the time data in the falcon buffer block (integer). |

## Description

This function is used to get the falcon time into a string format. The time string has the following format '**HH:MM:SS.FFF**' where **HH** represents hours, **MM** minutes, **SS** seconds and **FFF** fractions of second. This time string data can be used in the text table to display falcon time.

## Result Type/Space

S[14]

## Example

```
; Name          Units  Number  Result   Computations
"FalconTime"     ""     F100    S[14]   FalconTime(A100, 5)
```

# FArray(), Float Array Element Access

## Synopsis

FArray(F, INDEX)

**F**[*n*] Formula number of an array of values (*n*≥1).

**INDEX**[1] Index of desired float in array (integer ≥ 0).

*Note: Deprecated [M300 function replacement - See "FIndex(), Float Element Access"]*

## Description

This function is used to access individual elements in a float array.

$$f = F[INDEX]$$

## Result Type/Space

D[1]

## Example

```
; Name          Number  Formula Result   Computations
"FloatArray"     ""      F200    F[1]     Farray(F100, 3)
```

# FIndex(), Float Element Access

## Synopsis

**FIndex**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula number of an array of values (float) (*n*≥1). |
| **INDEX**[1] | Index number of element being referenced (integer ≥ 0). |

## Description

Uses the element **INDEX** to reference that particular value in an array of (**float**) data.

$$f = F[INDEX]$$

## Result

D[1]

## Example

```
; Name          Units  Number  Result   Computations
"FloatIndex"     ""     F300    F[1]    FIndex(F100, 24)
```

# Ge(), Greater Than Equal

## Synopsis

**Ge**(A, B, FTRUE, FFALSE)

| | |
|---|---|
| **A**[1] | First value used in comparison. |
| **B**[1] | Second value used in comparison. |
| **FTRUE**[*m*] | Formula for true value ($m \geq 1$). |
| **FFALSE**[*p*] | Formula for false value ($p \geq 1$). |

## Description

This function compares two values and it returns the value of true formula ('**FTRUE**'), if the first value **A** is greater than or equal to the second value **B,** otherwise the value of false formula ('**FFALSE**') is returned. This function uses Interpolation [**See Interpolation**].

$$\text{if } (A \geq B) \text{ then } f[i] = FTRUE[i]$$
$$\text{else } f[i] = FFALSE[i]$$

## Result Type/Space

D[*n*], if A ≥ B, $n = m$, else $n = p$

## Example

```
; Name              Units  Number  Result   Computations
"GreaterThanEqual"  ""      F130    F[1]    GE(F100, F101, F200, F201)
```

# GetData(), Get Data

## Synopsis

**GetData**(A, OFFSET, COUNT, TYPE)
**GetData**(A, OFFSET, COUNT, TYPE, SWAP)

| | |
|---|---|
| **A** | Acquisition tag for data (tag). |
| **OFFSET**[1] | Byte offset into data (integer). |
| **COUNT**[1] | Number of data values to be returned (integer). |
| **TYPE**[1] | Type of data (integer) |
| **SWAP**[1] | Swap data (integer: 0 or 1) |

## Description

This function is used to get data from a raw acquisition tag. Data offset and count are user selectable. The data can be byte swapped if necessary. The type of data used is based on the following table.

| Size (Bytes) | TYPE | Data Type | Result |
|---|---|---|---|
| 1 | 1 | String | S[n] |
| 1 | 2 | Signed Char | C[n] |
| 1 | 3 | Unsigned Char | c[n] |
| 2 | 4 | Signed Integer | I[n] |
| 2 | 5 | Unsigned Integer | i[n] |
| 4 | 6 | Signed Long | L[n] |
| 4 | 7 | Unsigned Long | l[n] |
| 4 | 8 | Float | F[n] |
| 8 | 9 | Double | D[n] |

**Type**

## Result Type/Space

See type table (n = COUNT).

## Example

```
; Name          Units  Number  Result   Computations
"GPSLatitude  "rad"    F1104    D[1]   GetData(A1200, 12, 1, 9)
```

# GrData(), Grey Data Access

## Synopsis

**GrData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for 2D Grey Advanced data (tag). |
| **SELECT**[**1**] | Selector for desired data (integer: 0..14). |

## Description

This function allows access to individual items of the 2D Grey Advanced data header. The following table shows the different **SELECT** values for the different 2D Grey Advanced header data fields.

| Data Field | SELECT |
|---|---|
| 2DG or 2DGADV Samples | 0 |
| Particle Count | 1 |
| Forward Link | 2 |
| Reserved 1 | 3 |
| Elapsed Time (SOI-Since Start of Image) | 4 |
| Elapsed Time (SOB-Since Start of Buffer) | 5 |
| Image Slice Count | 6 |
| Multiply TAS Factor | 7 |
| Divide TAS Factor | 8 |
| Minimum Pixels | 9 |
| Middle Pixels | 10 |
| Maximum Pixels | 11 |
| Reserved 2 | 12 |
| Reserved 3 | 14 |

**2DGREY Data SELECT Options**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name           Units    Number  Result   Computations
"ElapsedTime"    "ms"      F300    F[1]    GrData(A2000, 4)
```

# GrSums(), 2D Grey Sums

## Synopsis

**GrSums**(PROBE, A, MODE, FREQUENCY)
**GrSums**(A, MODE, PROBE, INTERVAL, STATE, PIXELSIZE)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **A** | Acquisition tag for 2D Grey Advanced data (tag). |
| **MODE**[1] | Function sizing mode (integer). |
| **STATE**[1] | Function control variable (integer). |
| **FREQUENCY**[1] | Frequency of integration. |
| **INTERVAL**[1] | Integration interval (integer). |
| **PIXELSIZE**[1] | Pixel dimension. |

## Description

This function builds up an approximation for the 2D Grey advanced spectrum using the particle dimensions and enclosed time of the 2D Grey scaled images. These particles can be summed up (using several different modes) and normalized using the elapsed time value (if desired). The output of the function is an array and may be processed like the arrays from the 1D and 2D data.

Using the **MODE** parameter, it is possible to control the sizing method. The lower nibble for the **MODE** parameter controls the sizing method while the upper nibble for the **MODE** parameter has some additional control bits (shadow levels and uncorrected counts). the **MODE** parameter is best specified in hexadecimal notation. In order to come up with the correct value for the **MODE** parameter it is necessary to first find the desired sizing methods and then use the decimal and binary values from the tables in the next page to come up with the final value (in hexadecimal) to pass to the summation function. For example, to specify the area (with edge reject) sizing method using only the middle and maximum shadows. You would pick 4 for the nibble from the first table. Then pick 0110 (binary) for the upper nibble (maximum and middle shadows). The hexadecimal value for 0110 (binary) is 6. Therefore the desired value for the **MODE** parameter is 0x64.

| Mode (low nibble) | Description |
|---|---|
| 0 | (X + Y) / 2 |
| 1 | X (TAS independent) |
| 2 | Y (TAS dependant) |
| 3 | Area |
| 4 | Area(reject particles that touch edge |

**MODE, Lower-nibble**

| Mode (low nibble) | Description |
|---|---|
| 5 | X(reject particles that touch edge) |
| 6 | Y(reject particles that touch edge) |
| 7 | (X + Y) / 2 (reject particles that touch edge) |

**MODE, Lower-nibble (Continued)**

The following table shows the valid **MODE** bits (bits 7, 6, 5, and 4 of the **MODE**) for the upper nibble (binary values, x means don't care) and a description of what they do to control the sizing methods.

| Mode (upper nibble) | Description |
|---|---|
| xxx1 | Minimum shadow bit selector |
| xx1x | Middle shadow bit selector |
| x1xx | Maximum shadow bit selector |
| 1xxx | Raw uncorrected counts bit selector |

**MODE, Upper-nibble**

As you can see, bit 7 of the **MODE** parameter can be set to return raw uncorrected counts (or in 0x8 hexadecimal), for all sizing methods (no normalization using elapsed time).

For both area sizing methods (lower nibble 3 and 4 for the **MODE**), you can use the upper nibble for the 'MODE' parameter to control which shadow levels are added (minimum, middle, maximum). A shadow level is added to the total area calculation by setting the corresponding bit. Bit 4 is used for minimum shadow, bit 5 for middle shadow and bit 6 for maximum shadow.

If the minimum, middle, and maximum bits of the **MODE** parameter are all zero, this indicates an invalid mode and all bits are assumed on (default mode).

A particle is found to touch the edge by having either the first of the last pixel set in any color (minimum, middle or maximum). This test is done by this function for all particle slices. For the edge reject modes, the elapsed times for all particles (rejected or not) are counted up and used in the final correction.

The X size of a particle is computed by adding all the set bits in a particular slice. From slice to slice, in a given particle, the X size only changes, if it was greater than the largest X size found so far.

The X size computation may be modified by the minimum, middle and maximum bits of the **MODE** parameter. If the minimum bit is set, all three shadow levels are counted (minimum, middle and maximum). If the middle bit is set, two shadow levels are counted (middle and maximum). Similarly, if the maximum bit is set, only the maximum shadow is counted. For the X size method, these are the only three valid modifiers. One of these methods will be picked depending on which bits are set (minimum is checked first, then middle and finally maximum).

## Result Type/Space

D[n], n = number of data samples

## Example

```
; Name     Units  Number  Result    Computations
"GrSums"     ""     F100    F[64]   GrSums(Pr.2dg, A100, 0x01, 1.0)
```

# Gt(), Greater Than

## Synopsis

**Gt**(A, B, FTRUE, FFALSE)

| | |
|---|---|
| **A**[1] | First value used in comparison. |
| **B**[1] | Second value used in comparison. |
| **FTRUE**[*m*] | Formula for true value ($m \geq 1$). |
| **FFALSE**[*p*] | Formula for false value ($p \geq 1$). |

## Description

This function compares two values and it returns the value of true formula ('**FTRUE**'), if the first value **A** is greater than the second value **B,** otherwise the value of false formula ('**FFALSE**') is returned.

$$\text{if } (A > B) \text{ then } f[i] = FTRUE[i]$$
$$\text{else } f[i] = FFALSE[i]$$

## Result Type/Space

D[*n*], if A > B, $n = m$, else $n = p$

## Example

```
; Name          Units  Number  Result   Computations
"GreaterThan"    ""     F130    F[5]    Gt(F100, F101, F201, F202)
```

# HSAnalog(), High Speed Analog Scaling

## Synopsis

**HSAnalog**(A, X, B)

| | |
|---|---|
| **A** | Acquisition tag for High Speed Analog data (tag) [2 byte, integer]. |
| **X**[1] | Formula for scaling value. |
| **B**[1] | Formula for offset value. |

## Description

This function converts analog counts data to floating point units in volts.

$$f[i] = A[i] \cdot X + B$$
$$\text{for } i = 0 \ldots [n-1] \text{ samples}$$

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name              Units     Number  Result   Computations
"HighSpeedAnalog"    "v"       F100    F[20]   HsAnalog(A100, 3.0517578E-4, 0)
```

# HvMask(), High Volume Precipitation Spectrometer Mask

## Synopsis

**HvMask**(A)
**A** Acquisition tag for HVPS data (tag).

## Description

This function will look at all existing HVPS data and try to find a diagnostics buffer with the current mask information. The current HVPS mask is retrieved as an array of 16 integer words (16 bits each word), for a total of a 256 bit mask.

## Result Type/Space

I[16]

## Example

```
; Name          Units  Number  Result   Computations
"HVPSMask"       ""     F100    I[16]   HvMask(A100)
```

# HvpsMask(), High Volume Precipitation Spectrometer Mask

## Synopsis

**HvpsMask**(A)

**A** Acquisition tag for HVPS data (tag).

*Note: Deprecated [M300 function replacement - See "HvMask(), High Volume Precipitation Spectrometer Mask"]*

## Description

This function will look at all existing HVPS data and try to find a diagnostics buffer with the current mask information. The current HVPS mask is retrieved as an array of 16 integer words (16 bits each word), for a total of a 256 bit mask.

## Result Type/Space

I[16]

## Example

```
; Name        Units  Number  Result   Computations
"HVPSMask"     ""     F100    I[16]   HvpsMask(A100)
```

# HvpsTiming(), High Volume Precipitation Spectrometer Timing

## Synopsis

**HvpsTiming**(A, TASFACTORTAG)

| | |
|---|---|
| **A** | Acquisition tag for HVPS data (tag). |
| **TASFACTORTAG**[1] | Tag for HVPS TAS factors data (tag). |

***Note: Deprecated [M300 Replacement function - See "HvTiming(), High Volume Precipitation Spectrometer Timing"]***

## Description

This function is used to retrieve the HVPS timing data (first float element) and the HVPS overflow data (second float element) from an HVPS data block..

## Result Type/Space

D[2]

## Example

```
; Name          Units  Number  Result   Computations
"HVPSTiming"     ""     F100    D[2]    HVPSTiming(A100, A101)
```

# HvSums(), High Volume Precipitation Spectrometer Sums

## Synopsis

**HvSums**(PROBE, A, MODE, FREQUENCY)
**HvSums**(A, PROBE, INTERVAL)

| | |
|---|---|
| **A** | Acquisition tag for HVPS data (tag). |
| **PROBE** | Probe name/number (probe). |
| **MODE**[1] | Summation options (integer). |
| **FREQUENCY**[1] | Integration frequency. |
| **INTERVAL**[1] | Integration interval (integer). |

## Description

This function builds up an approximation of the HVPS spectrum using the image data and the time information.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name         Units  Number  Result   Computations
"HVPSSums"      ""     F100    F[64]   HvSums(P3, A100, 1, 1.0)
```

# HvTiming(), High Volume Precipitation Spectrometer Timing

## Synopsis

**HvpsTiming**(A, TASFACTORTAG)

| | |
|---|---|
| **A** | Acquisition tag for HVPS data (tag). |
| **TASFACTORTAG**[1] | Tag for HVPS TAS factors data (tag). |

## Description

This function is used to retrieve the HVPS timing data (first float element) and the HVPS overflow data (second float element) from an HVPS data block..

## Result Type/Space

D[2]

## Example

```
; Name         Units  Number  Result   Computations
"HVPSTiming"    ""     F100    D[2]    HvTiming(A100, A101)
```

# IArray(), Integer Array Element Access

## Synopsis

**IArray**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula for array of integers (integer) (*n*≥1). |
| **INDEX**[1] | Index of desired element from integer array (integer Š 0). |

*Note: Deprecated [M300 Replacement function - See "IIndex(), Integer Element Access"]*

## Description

This function is used to access an individual element from an integer array. It returns a integer value referenced by **INDEX** in the integer array referenced by **F**.

$$f = F[INDEX]$$

## Result Type/Space

I[1]

## Example

```
; Name           Units  Number  Result   Computations
"Integer Array"  ""      F200    I[1]     IArray(F100, 3)
```

# IasP(), Inverse Pressure Indicated Airspeed

## Synopsis

**IasP**(IAS)
**IAS**[*n*] Formula of an array of indicated airspeed (in knots) (*n*>0).

## Description

This function computes the pitot pressure (differential pressure) from indicated air speed.

$$f[i] = 1013.25\left[\left(\frac{IAS^2}{2.1837E6} + 1\right)^{3.49813} - 1\right]\text{mb}$$
$$\text{for } i = 0\ldots[n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Namne        Units     Number  Result   Computations
"PitotPress"   "mb"      F200    F[1]     IasP(F100)
```

# IIndex(), Integer Element Access

## Synopsis

**IIndex**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula for array of integers (integer) (*n*>0). |
| **INDEX**[1] | Index of desired element from integer array (integer > 0). |

## Description

This function is used to access an individual element from an integer array. It returns a integer value referenced by **INDEX** in the integer array referenced by **F**

$$f = F[INDEX]$$

## Result Type/Space

I[1]

## Example

```
; Name          Units  Number  Result   Computations
"Integer Index"  ""     F200    I[1]    IIndex(F100, 3)
```

# Incloud(), In Cloud Prediction

## Synopsis

**Incloud**(STDEV, STDEVTHRES, SPOWER, PPOWER, POWERTHRES, TIME)

| | |
|---|---|
| **STDEV**[*1*] | Standard deviation for power (float). |
| **STDEVTHRES**[*1*] | Standard deviation for power threshold (float). |
| **SPOWER**[*1*] | Sense power (float). |
| **PPOWER**[*1*] | Predicted power (float). |
| **POWERTHRES**[*1*] | Power threshold (float). |
| **TIME**[*1*] | Time in seconds (integer). |

## Description

This function is used for in cloud prediction from the WCM-2000 system.

## Result Type/Space

D[1], F[1], L[1], l[1], I[1], i[1]

## Example

```
; Namne    Units Number Result Computations
"TWCIncloud"  "" F13204  I[1] Incloud(F10110, F11046, F12035, F13102, 1.1, 30)
```

# Ins429Bin(), INS 429 Data

## Synopsis

**Ins429Bin**(A, BITS, RANGE)

| | |
|---|---|
| **A** | Acquisition tag for INS 429 BIN data (tag). |
| **BITS**[1] | Number of bits to used from data (integer: 1..21). |
| **RANGE**[1] | Full scale range. |

## Description

This function unpacks INS 429 BIN data.

$$f[i] = (A[i] \gg (20 - BITS)) \cdot \frac{RANGE}{2^{BITS}}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name         Units  Number  Result    Computations
"INS429BIN"     ""     F100    F[35]    Ins429Bin(A100, 11, 0.25)
```

# InsBCD(), INS BCD Data

## Synopsis

**InsBcd**(A)

**A** Acquisition tag for INS BCD data (tag).

## Description

This function unpacks INS BCD data and returns a normalized value between -1.0 and 1.0.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name     Units  Number  Result    Computations
"InsBCD"    ""     F100    F[20]    InsBCD(A100)
```

# InsBin(), INS Binary Data

## Synopsis

**InsBin**(A)

**A** Acquisition tag for INS Binary data (tag).

## Description

This function unpacks INS binary data and returns a normalized value between -1.0 and 1.0.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name      Units  Number  Result   Computations
"InsBin"     ""     F100    F[20]   InsBin(A100)
```

# InsBin2(), INS Binary 2 Data

## Synopsis

**InsBin2**(A)

**A** Acquisition tag for INS Binary data (tag).

## Description

This function unpacks INS binary data format on the NOAA P3. This format contains only one sign bit as opposed the to the standard two sign bits. This function returns a normalized value between -1.0 and 1.0.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name     Units  Number  Result   Computations
"InsBin"    ""     F100    F[20]   InsBin2(A100)
```

# InsPos(), INS BCD Position

## Synopsis

**InsPos**(A)

**A** Acquisition tag for INS BCD latitude or longitude data (tag).

## Description

This function unpacks INS BCD data for the latitude or longitude labels and returns radians.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name      Units    Number   Result   Computations
"INSPos"    "rad"    F100     F[1]     InsPos(A100)
```

# IntegerData(), Integer Data

## Synopsis

**IntegerData**(A, INDEX, SCALE, OFFSET, SWAP)

| | |
|---|---|
| **A** | Acquisition tag for integer data (tag). |
| **INDEX**[1] | Index to desired integer data (integer). |
| **X**[1] | Scale multiplier value. |
| **B**[1] | Offset value. |
| **SWAP**[1] | Perform byte swap on data (integer: 0 or 1). |

## Description

This function will retrieve integer data from the data buffer and perform scaling on the data. If swap is set to 1 the function will change the high byte to the low byte and the low byte to the high byte for each index.

$$f[i] = A[INDEX + i] \cdot X + B$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name          Units  Number  Result   Computations
"IntegerData"    ""     F100    F[10]   IntegerData(A100, 5, PI, 0, 1)
```

# Intercept(), Calculate Intersect Point at Y-Axis

## Synopsis

**Intercept**(KNOWNYS, KNOWNXS, STATE)
**KNOWNYS**[n] Known Y values.
**KNOWNXS**[n] Known X values.
**STATE**[1] Function control variable (integer).

## Description

This function calculates the point at which a line intersects the y-axis by using the best fit regression line plotted through the known x and known y values.

The **STATE** control variable is used to control the function operation mode. If the **STATE** control variable is a '0', then the function performs a reset and computes a new value. If the **STATE** control variable is a '1', then new data is processed and a intercept value returned. If the **STATE** control variable is a '2', this cause the last intercept value to be held.

## Result Type/Space

D[*1*]

## Example

```
; Name          Units  Number  Result   Computations
"Intercept"       ""     F100    F[1]   Intercept(F1000, F2000, F200)
```

# IR(), In Range

## Synopsis

**IR**(F, LOW, HIGH)

| | |
|---|---|
| **F**[*n*] | Formula value for an array of values (*n*≥1). |
| **LOW**[1] | Formula for lower limit. |
| **HIGH**[1] | Formula for upper limit. |

## Description

This function is short for In Range and will return a one if the formula value is within the low and high limits (inclusive). Otherwise, if the formula value is outside the range, a zero is returned.

$$\text{if } ((F[i] \geq LOW) \wedge (F[i] \leq HIGH)) \text{ then } f[i] = 1$$
$$\text{else } f[i] = 0$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

I[*n*]

## Example

```
; Name       Units  Number  Result   Computations
"InRange"     ""     F100    I[10]   IR(F101, 5, 100)
```

# IVar1D(), Inverse Velocity Acceptance Ratio 1D

## Synopsis

**IVar1D**(A, STROBEINDEX, TOTALSTROBEINDEX, CFAC, INTERVAL)

| | |
|---|---|
| **A** | Acquisition tag for 1D data (tag). |
| **STROBEINDEX**[1] | Strobe count channel index (integer). |
| **TOTALSTROBEINDEX**[1] | Total strobes count index (integer). |
| **CFAC**[1] | Correction factor. |
| **INTERVAL**[1] | Integration interval (integer). |

*Note: Deprecated [M300 Replacement function - See "OdIVar(), 1D Inverse Velocity Acceptance Ratio"]*

## Description

This function computes the inverse velocity acceptance ratio from 1D data. This value can be used to correct concentrations, volumes and masses calculations.

$$f = \frac{\Sigma A[TOTALSTROBEINDEX]}{\Sigma A[STROBEINDEX]} \cdot CFAC$$

## Result Type/Space

D[1]

## Example

```
; Name                Units   Result     Computations
"InverseVelAccRatio"  ""       F[10]     IVar1D(A100, 15, 16, 1, 1)
```

# IVar1DAdv(), Advanced Inverse Velocity Acceptance Ratio 1D

## Synopsis

**IVar1DAdv**(STROBE, TOTALSTROBES, CFAC, INTERVAL)

| | |
|---|---|
| **STROBE** | Tag for strobe count (tag). |
| **TOTALSTROBES** | Tag for total strobes (tag). |
| **CFAC**[1] | Correction factor. |
| **INTERVAL**[1] | Integration interval (integer). |

*Note: Deprecated [M300 Replacement function - See "OdIVarAdv(), 1D Advanced Inverse Velocity Acceptance Ratio"]*

## Description

This function computes the inverse velocity acceptance ratio from 1D advanced data. This value can be used to correct concentrations, volumes and masses calculations.

$$f = \frac{TOTALSTROBES}{STROBE} \cdot CFAC$$

## Result Type/Space

D[1]

## Example

```
; Name                    Units   Number   Result   Computations
"AdvInverseVelAccRatio"   ""       F100     F[1]    Ivar1dAdv(A100, A101, 1, 1)
```

# KeyIndex(), Sorted Array Indexing

## Synopsis

**KeyIndex**(KEY)
**KeyIndex**(KEY, COUNT)

| | |
|---|---|
| **KEY**[1] | Key value to be inserted/indexed (long integer). |
| **COUNT**[1] | Size of array required (long integer). |

## Description

This function is used to insert or search a sorted array for a particular value. If the **KEY** is already present in the array, the index (position within the array) is returned. If the **KEY** value is not already present, it will be inserted into the array based on how it compares with the other values already in the array. It's new index will then be returned. Note that the index is zero based, meaning the first value in the array as an index position of zero.

The **COUNT** value is to be used only when creating the array for the first time (initialization). After that, **COUNT** will be ignored. Not also that if **COUNT** is omitted during initialization, a default value of 1024 will be used.

## Result Type/Space

L[1]

## Example

```
; Name      Units  Number  Result  Computations
"DropID"    ""     F200    L[1]    KeyIndex(F1001, 512)
```

# LArray(), Long Array Element Access

## Synopsis

**LArray**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula for an array of long integers (*n*≥1). |
| **INDEX**[1] | Index of desired long integer in array (integer ≥ 0). |

*Note: Deprecated [M300 Replacement function -See "LIndex(), Long Element Access"]*

## Description

This function is used to access individual elements in a (long integer) array.

$$f = F[INDEX]$$

## Result Type/Space

D[1]

## Example

```
; Name          Units  Number  Result  Computations
"LongArray"     ""     F300    L[1]    LArray(F100, 25)
```

# LatStr(), Latitude String

## Synopsis

**LatStr**(LATITUDE)
**LATITUDE**[1] Formula for latitude value (in radians).

## Description

This function converts the latitude in radians value to an ASCII latitude string for display purposes. The return string is in the form of '**N DD MM.HH**' where '**N**' stands for north/south, '**DD**' stands for degrees, '**MM**' stands for minutes and '**HH**' for fraction of minutes.

## Result Type/Space

S[12]

## Example

```
; Name        Units  Number  Result   Computations
"LatString"    ""     F300    S[12]   LatStr(F100)
```

# Le(), Less Than Equal

## Synopsis

**Le**(A, B, FTRUE, FFALSE)

| | |
|---|---|
| **A**[1] | First value used in comparison. |
| **B**[1] | Second value used in comparison. |
| **FTRUE**[*m*] | Formula for true value ($m \geq 1$). |
| **FFALSE**[*p*] | Formula for false value ($p \geq 1$). |

## Description

This function compares two values and it returns the value of true formula **FTRUE**, if the first value **A** is less than or equal to the second value **B**, otherwise the value of false formula **FFALSE** is returned. This function uses Interpolation [**See Interpolation**].

$$\text{if } (A \leq B) \text{ then } f[i] = FTRUE[i]$$
$$\text{else } f[i] = FFALSE[i]$$

## Result Type/Space

D[*n*], if A ≤ B, $n = m$, else $n = p$

## Example

```
; Name           Units   Number  Result   Computations
"LessThanEqual"   ""      F300    F[1]    Le(F100, F102, F200, F100)
```

# LeastSqReg(), Least Square Regression

## Synopsis

**LeastSqReg**(KNOWNYS, KNOWNXS, STATE, COUNT)

| | |
|---|---|
| **KNOWNYS**[n] | The known y values. |
| **KNOWNXS**[n] | The know x values. |
| **STATE**[*1*] | The control STATE variable (Integer). |
| **COUNT**[*1*] | Number of points (Integer). |

## Description

This function uses the least square regression method to obtain a, b, c and r values from a set of x and y values.

When the state is zero, the function resets the computation variables for the least square regression. New values are used for the computation while the state is a one. The final results for a, b, c and r values can be obtained when the state is a two.

The count is used to setup the number of internal memory points for the x and y values. This function need to keep the x and y variables in memory so it can do the r computation.

## Result Type/Space

D[*4*], The result for a is index 0, the result b is at index 1, the result for c is at index 2 and the result for r is at index 3.

## Example

```
; Name           Units   Number  Result   Computations
"LeastSqReg" "" F3600 F[4] LeastSqReg(F3400, F3520, F3002, 10000)
"a" "" F3610 F[1] FIndex(F3600, 0)
"b" "" F3611 F[1] FIndex(F3600, 1)
"c" "" F3612 F[1] FIndex(F3600, 2)
"r" "" F3613 F[1] FIndex(F3600, 3)
```

# Limit(), Limit Value

## Synopsis

**Limit**(F, LOW, HIGH)

| | |
|---|---|
| **F**[*n*] | Formula value to be limited (*n*≥1*)*. |
| **LOW**[1] | Lower limit value. |
| **HIGH**[1] | Upper limit value. |

## Description

This function checks limits on a formula value and hard limits the result between a low and a high limit.

$$if(F[i] \le LOW) \text{ then } f[i] = LOW$$
$$\text{else } if(F[i] \le HIGH) \text{ then } f[i] = HIGH$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name    Units  Number  Result   Computations
"Limit"    ""     F100    F[1]    Limit(F100, 0, 100)
```

# LIndex(), Long Element Access

## Synopsis

**LIndex**(F, INDEX)

| | |
|---|---|
| **F**[*n*] | Formula representing an array of values (long) (n≥1). |
| **INDEX**[1] | Index number of element being referenced (integer ≥ 0). |

## Description

Uses the element INDEX to reference that particular value in an array of (long integer) data.

$$f = F[INDEX]$$

## Result Type/Space

D[1]

## Example

```
; Name              Units   Number  Result   Computations
"LongIntegerIndex"  ""       F300    F[15]   LIndex(F100, 24)
```

# LonStr(), Longitude String

## Synopsis

**LonStr**(LONGITUDE)
**LONGITUDE**[1] Formula for longitude value (in radians).

## Description

This function converts the longitude in radians value to an ASCII longitude string for display purposes. The return string is in the form of '**E DDD MM.HH**' where '**E**' stands for east/west, '**DDD**' stands for degrees, '**MM**' stands for minutes and '**HH**' for fraction of minutes.

## Result Type/Space

S[12]

## Example

```
; Name            Units Number  Result   Computations
"LongitudeString"  ""    F200    S[12]   Lonstr(F100)
```

# Lookup(), Lookup Interpolation

## Synopsis

**Lookup**(F, LOOKUP)

| | |
|---|---|
| **F**[*n*] | Formula value (float) (*n*≥1). |
| **LOOKUP** | Lookup table (lookup). |

## Description

This function computes the linear interpolation of the value(s), given an X and Y array of data provided via the lookup table. See "Lookup Table, (lup.300)"

If the X value is less than the domain of the X values in the table, then lowest Y value will be returned. On the other hand, if the X value is greater than the domain of X values, then the highest Y value will be returned.

The formula passed to this function must be of the float type and can represent a single value or an array of values.

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result   Computations
"Lookup"     ""     F200    F[10]   Lookup(F100, Lo.Temp)
```

# LookupGet(), Lookup Entry Get Value

## Synopsis

**LookupGet**(LOOKUP, ROW, COLUMN)

| | |
|---|---|
| **LOOKUP** | Lookup table (lookup). |
| **ROW**[1] | Row index to return (integer). |
| **COLUMN**[1] | Column index to return (integer). |

## Description

This function returns a value or an array of values depending on the index values passed to the function, from the lookup data entry. See "Lookup Table, (lup.300)"

The row and column indexes are base 0, this means the first value has an index of zero (not one). An index of minus one indicates that the full row or column is to be return. We can't do two dimensional arrays in the M300. One index can be minus one but not both simultaneously.

When both row and column indexes are non negative then we are getting a single value from the lookup data.

To return a row you set column to minus one.

To return a column you set row to minus one.

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result   Computations
"LookupGet"  ""      F200   F[10]   LookupGet(Lo.Temp, -1, 4)
```

# LookupSet(), Lookup Set Entry Value

## Synopsis

**LookupSet**(LOOKUP, ROW, COLUMN, F)

| | |
|---|---|
| **LOOKUP** | Lookup table (lookup). |
| **ROW**[1] | Row index to modify (integer). |
| **COLUMN**[1] | Column index to modify (integer). |
| **F**[*n*] | Formula value to modify in lookup (float) (*n*≥1). |

## Description

This function modifies a value or an array of values depending on the index values passed to the function, from the lookup data entry. See "Lookup Table, (lup.300)"

The row and column indexes are base 0, this means the first value has an index of zero (not one). An index of minus one indicates that the full row or column is to be return. We can't do two dimensional arrays in the M300. One index can be minus one but not both simultaneously.

When both row and column indexes are non negative then we are setting a single value from the lookup data.

To set a row you set column to minus one and specify the values to set in the formula value.

To set a column you set row to minus one and specify the values to set in the formula value.

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result   Computations
"LookupSet"  ""     F200    F[10]   LookupSet(Lo.Temp, -1, 4, F100)
```

# LrnPos(), Loran/GPS Position

## Synopsis

**LrnPos**(A)

**A** Acquisition tag for Loran/GPS latitude or longitude data (tag).

## Description

This function converts Loran/GPS latitude or longitude raw data into radians. This function upon completion returns an array of size *n* floating point values containing the Loran/GPS latitude and longitude data in radians. The size of *n* is the number of samples field from the directory referenced by **A** (Acquisition tag name/number).

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name            Units   Number  Result  Computations
"LoranPosition"   "rad"   F300    F[15]   LrnPos(A100)
```

# Lt(), Less Than

## Synopsis

**Lt**(A, B, FTRUE, FFALSE)

| | |
|---|---|
| **A**[1] | First value used in comparison. |
| **B**[1] | Second value used in comparison. |
| **FTRUE**[*m*] | Formula for true value ($m \geq 1$). |
| **FFALSE**[*p*] | Formula for false value ($p \geq 1$). |

## Description

This function compares two values and it returns the value of true formula **FTRUE** if the first value is less than the second value, or the value of false formula **FFALSE** otherwise. This function uses Interpolation [**See Interpolation**].

$$if(A < B) \text{ then } f[i] = FTRUE[i] \quad .$$
$$\text{else } f[i] = FFALSE[i]$$

## Result Type/Space

D[*n*], if A < B, $n = m$, else $n = p$

## Example

```
; Name         Units  Number  Result   Computations
"LessThan"      ""     F300    F[5]    Lt(F100, F101, F200, F100)
```

# LToF(), Long to Float

## Synopsis

**LToF**(A)

**A** Acquisition tag for long data (tag).

## Description

This function takes four bytes (long integer) and converts them to a float number. Upon completion this function returns an array of size *n* containing the converted long integer to float

values, where *n* is the number samples field from the directory referenced by **A** (Acquisition tag name/ number).

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name           Units  Number  Result   Computations
"LongToFloat"     ""     F100    F[20]   LToF(A100)
```

# Masses(), Masses

## Synopsis

**Masses**(PROBE, F, CFAC, TAS, FREQUENCY, LINEAR, EXP)
**Masses**(F, P, RANGE, CFAC, TAS, INTERVAL, LINEAR, EXP)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **F**[*m*] | Formula for the array of sums (*m*≥1). |
| **CFAC**[1] | Correction factor. |
| **TAS**[1] | True air speed value. |
| **RANGE**[1] | Range used in probe definition table (integer). |
| **INTEVAL**[1] | Interval of summation (integer) |
| **LINEAR**[1] | Linear scaling coefficient. |
| **EXP**[1] | Exponential scaling coefficient. |
| **FREQUENCY**[1] | Integration frequency. |

## Description

This function uses the summed up channel counts and the probe definition table to compute masses. The result is typically used for mean, median, mode, total mass calculations as well as X vs. Y display plots. This function should be 'refreshed' at the same time interval as the summation routine generates data, so as to eliminate redundant calculations on the same input data.

The 'SAREA' and 'MIDSIZE' originate from the user specified channel files via the probe name/number. The following formula summarizes the computations.

$$f[i] = \frac{F[i] \cdot LINEAR \cdot MIDSIZE[i, RANGE]^{EXP}}{SAREA[i, RANGE] \cdot TAS \cdot \frac{BUFLIFE}{SYSFREQ} \cdot INTERVAL \cdot CFAC}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result

D[*n*], *n* = min(*m, probe channels*)

## Example

```
; Name   Units  Number  Result Computations
Masses    ""     F200    F[15] Masses(F100, Pr.2D, F102, F1, 2.0, 1.4, 1.0)
```

# Max(), Maximum

## Synopsis

**Max**(F)
**F**[*n*] Formula of an array of values (*n*≥1).

## Description

This function is used to find an return the largest value in the formula (array of values).

## Result Type/Space

D[1]

## Example

```
; Name          Units  Number  Result   Computations
"MaximumVal"     ""     F300    F[1]     Max(F100)
```

# MaxSiz(), Maximum Size

## Synopsis

**MaxSiz**(X, Y)
**MaxSiz**(X, Y, MODE)

| | |
|---|---|
| **X**[*m*] | Formula of an array for X data values (*m*≥1). |
| **Y** [*p*] | Formula of an array for Y data values (*p*≥1). |
| **MODE**[1] | Which computation mode to use (0, 1). |

## Description

The MaxSiz function has two modes of operation.

Mode 0
This function is used to find the largest value in the X array corresponding to a non zero value in Y array. This is done by searching the Y array from the end to the beginning until a non zero value is found. It then uses the same index to return the corresponding value from the X array. This function is typically used to return the largest particle size found using the sizes data as the X array and the sums data as the Y array.

Mode 1
In this mode the function uses the Y array to search for the largest value. Then it returns the corresponding value from the X array.

## Result Type/Space

D[1]

## Example

```
; Name          Units  Number  Result  Computations
"MaximumSize"    ""     F300   F[1]    MaxSiz(F100, F200)
```

# MaxTim(), Maximum Time

## Synopsis

**MaxTim**(F, STATE)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **STATE**[1] | State option variable (integer: 0 or 1). |

## Description

This function returns the time string for which the maximum value of a formula occurred. The **STATE** variable is used to control the function operation. If the **STATE** is zero, the return value is unchanged. If the state changes from zero to one (rising edge), the last time is cleared and a new maximum time is started. If the **STATE** is one, the time will change when the current value is a maximum. The return string is in the form of **HH:MM:SS**', **HH** is the hours, **MM** is the minutes and **SS** is the seconds.

## Result Type/Space

S[10]

## Example

```
; Name           Units  Number  Result  Computations
"MaximumTime"     ""     F200    S[10]  MaxTim(F100, F99)
```

# MaxVal(), Maximum Value

## Synopsis

**MaxVal**(F, STATE)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **STATE**[1] | State option variable (integer: 0 or 1). |

## Description

This function returns the maximum value observed. The **STATE** variable is used to control the function operation. If the **STATE** is zero, the maximum value stays unchanged. If the state changes from zero to one (rising edge), the maximum value is cleared and a new maximum is started. If the **STATE** is one, the maximum value will change when the current value is a maximum.

## Result Type/Space

D[*n*]

## Example

```
; Name           Units  Number  Result  Computations
"MaximumValue"    ""     F200    F[1]   MaxVal(F100, 0)
```

# Mean(), Mean

## Synopsis

**Mean**(X, Y)

| | |
|---|---|
| **X**[*m*] | Formula of an array for X data values ($m \geq 1$). |
| **Y** [*p*] | Formula of an array for Y data values ($p \geq 1$). |

## Description

This function computes the expected X value using Y array values as weighing coefficients for the X array values.

## Result Type/Space

D[1]

## Example

```
; Name    Units   Number  Result  Computations
"Mean"    ""    F300    F[1]    Mean(F100, F200)
```

# Median(), Median

## Synopsis

**Median**(X, Y)
**Median**(X, Y, MODE)

| | |
|---|---|
| **X**[*m*] | Formula of an array for X data values ($m \geq 1$). |
| **Y** [*p*] | Formula of an array for Y data values ($p \geq 1$). |
| **MODE**[1] | Computation mode (integer: 0 or 1) |

## Description

Computes the median X value using the Y array values as the weighting coefficients for the X array values. Uses linear interpolation. This function has been upgraded to allow a mode parameter to be passed in. If **MODE** is set to zero, the area (old style) is used or if it is set to one, counts is used for the median computation

## Result Type/Space

D[1]

## Example

```
; Name       Units  Number  Result   Computations
"Median"      "mm"    F300    F[1]    Median(F100, F200)
```

# Min(), Minimum

## Synopsis

**Min**(F)

**F**[*n*] Formula of an array of values (*n*≥1).

## Description

This function return the minimum value from the formula (array of values).

## Result Type/Space

D[1]

## Example

```
; Name       Units    Number   Result    Computations
"Minimum"     "kt"     F901     F[1]      Min(F100)
```

# MinSiz(), Minimum Size

## Synopsis

**MinSiz**(X, Y)
**MinSiz**(X, Y, MODE)

| | |
|---|---|
| **X**[*m*] | Formula of an array for X data values (*m*≥1). |
| **Y** [*p*] | Formula of an array for Y data values (*p*≥1). |
| **MODE**[1] | Which computation mode to use (0, 1). |

## Description

The MinSiz function has two modes of operation.

Mode 0
This function is used to find the smallest value in the X array corresponding to a non zero value in Y array. This is done by searching the Y array from the beginning to the end until a non zero value is found. It then uses the same index to return the corresponding value from the X array. This function is typically used to return the smallest particle size found using the sizes data as the X array and the sums data as the Y array.

Mode 1
In this mode the function uses the Y array to search for the smallest value. Then it returns the corresponding value from the X array.

## Result Type/Space

D[1]

## Example

```
; Name          Units  Number  Result  Computations
"MinnimumSize"   ""     F300   F[1]    MinSiz(F100, F200)
```

# MinTim(), Minimum Time

## Synopsis

**MinTim**(F, STATE)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values/single value (*n*≥1). |
| **STATE**[1] | State option variable (integer: 0 or 1). |

## Description

This function returns the time string for which the minimum value of a formula occurred. The **STATE** variable is used to control the function operation. If the **STATE** is zero, the return value is unchanged. If the state changes from zero to one (rising edge), the last time is cleared and a new minimum time is started. If the **STATE** is one, the time will change when the current value is a minimum. The return string is in the form of **HH:MM:SS**', **HH** is the hours, **MM** is the minutes and **SS** is the seconds.

## Result Type/Space

S[10]

## Example

```
; Name           Units  Number  Result  Computations
"MinimumTime"     ""     F200    S[10]  MinTim(F100, F151)
```

# MinVal(), Minimum Value

## Synopsis

**MinVal**(F, STATE)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values (*n*≥1). |
| **STATE**[1] | State option variable (integer: 0 or 1). |

## Description

This function returns the minimum value observed. The **STATE** variable is used to control the function operation. If the **STATE** is zero, the minimum value is unchanged. If the state changes from zero to one (rising edge), the minimum value is cleared and a new minimum is started. If the **STATE** is one, the minimum value will change when the current values is a minimum.

## Result

D[*n*]

## Example

```
; Name           Units  Number  Result  Computations
"MinimumValue"    ""     F200    F[1]   MinVal(F100, 0)
```

# Mode(), Mode

## Synopsis

**Mode**(X, Y)

| | |
|---|---|
| **X**[*m*] | Formula of an array for X data values (*m*≥1). |
| **Y** [*p*] | Formula of an array for Y data values (*p*≥1). |

## Description

This function computes the mode X value using the Y array values as the weighing coefficients for the X array values. Uses quadratic interpolation.

## Result Type/Space

D[1]

## Example

```
; Name   Units  Number  Result  Computations
"Mode"   "µm"    F300    F[1]    Mode(F100, F200)
```

# MoSums(), 2D Mono Sums

## Synopsis

**MoSums**(PROBE, A, MODE, FREQUENCY)
**MoSums**(A, ELAPSED, MODE, PROBE, INTERVAL)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **A** | Acquisition tag for 2D Mono data (tag). |
| **ELAPSED** | 2D Elapsed time tag (tag). |
| **MASK**[1] | Time slice mask value (integer). |
| **FREQUENCY**[1] | Integration frequency. |
| **INTERVAL**[1] | Integration interval (integer). |

## Description

This function builds up an approximation of the 2D spectrum using the image data and the time slice mask. These images are summed up using either the slice count or the slice width and normalized using the elapsed time value. The output of the function is a sums array and may be processed like the sums array from 1D data.

The end of a particle is detected by finding one or more blank slices (all ones, 0xFFFFFFFF). In addition, the upper byte of the MASK parameter (16 bit integer) can be used to control the following options

| MODE | Description |
|---|---|
| xxxxxxx1xxxxxxxx | Add zero area particles to first bin |
| xxxxxxx0xxxxxxxx | Don't add zero area particles to first bin |
| xxxxxx1xxxxxxxxx | Use x-dimension method (across slice) |
| xxxxxx0xxxxxxxxx | Use y-dimension method (TAS dependent) |

**MODE**

## Result Type/Space

D[*n*], *n* = number of channels in probe entry (**PROBE**)

## Example

```
; Name    Units  Number  Result   Computations
"MoSums"   ""     F100    F[64]   MoSums(Pr.2dc, A100, 0x100, 1.0)
```

# Mul(), Multiply

## Synopsis

**Mul**(A, B)

**A**[*m*] Formula of an array of values (*m*≥1).

**B**[*p*] Formula of an array of values (*p*≥1).

*Note: Deprecated [M300 Replacement function - See "*, Multiply"]*

## Description

This function returns an array of values representing the multiplication of the two given arrays, element by element. This function uses interpolation. [**See Interpolation**].

$$f[i] = A[i]B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*, *p*)

## Example

```
; Name        Units  Number  Result   Computations
"Multiply"     ""     F300    F[15]   Mul(F100, F101)
```

# Nmea(), NMEA Sentence

## Synopsis

**Nmea**(F, IDSTR, SELSTR)

| | |
|---|---|
| **F**[*m*] | Formula string to tag for NMEA data (*m*≥1). |
| **IDSTR**[*p*] | String used for sentence ID (*p*≥1) (string). |
| **SELSTR**[*q*] | String used to pick data from NMEA Sentence (*q*≥1) (string). |

## Description

This function retrieves data from a GPS NMEA Sentence. The **IDSTR** can be "", or "NULL" to have the function search for data in ALL sentences (less efficient), however this might generate jitter in certain data, such as latitude. This is caused by the M300 looking for data in all sentences and the data having different values/precision in certain sentences. This fix is to use the **IDSTR** to pick the data from the correct NMEA sentence.

The **SELSTR** value is responsible for picking the type of data desired.

For an alternate function for getting NMEA data, *See "SrNmea(), NMEA Sentence"*

| SELECTOR | Type | Return Type | SENTENCEID |
|---|---|---|---|
| "DAT" | GPS Date | S[#] | GPZDA, GPRMC |
| "LAT" | Latitude (rad) | D[1] | GPGLL, GPRMC, GPGGA |
| "LON" | Longitude (rad) | D[1] | GPGLL, GPRMC, GPGGA |
| "GTK" | Ground Track (deg) | F[1] | GPVTG, GPRMC |
| "GSP" | Ground Speed (kts) | F[1] | GPVTG, GPRMC |
| "TIM" | GPS Time | G[#] | GPRMC, GPGGA |
| "STA" | Status | L[1] | GPGGA |
| "MGV" | Magnetic Var. (deg) | F[1] | GPRMC |
| "STC" | Satellite Count | F[1] | GPGGA |
| "ALTM" | Altitude (m) | F[1] | GPGGA |
| "ALT" | Altitude (ft) | F[1] | PGRMZ |
| "ROLL" | Roll (deg) | F[1] | SBG01 |
| "PITCH" | Pitch (deg) | F[1] | SBG01 |

**Table 8: SELSTR and IDSTR values**

| SELECTOR | Type | Return Type | SENTENCEID |
|---|---|---|---|
| "YAW" | Yaw (deg) | F[1] | SBG01 |

**Table 8: SELSTR and IDSTR values**

***Note: Data for this function must be in serial NMEA format. You cannot use this function to unpack data for the SEA GPS Interface. You must use the traditional way for SEA GPS interface data. The data can come from ANY available serial port in the system.***

## Result Type/Space

D[n], F[n], L[n], l[n], I[n], i[n], C[n], c[n], S[n].

## Example

```
; Name        Units   Number   Result   Computations
"Latitude"    "rad"   F1000    D[1]     Nmea(F205, "GPGLL","LAT")
```

# OdCmd(), 1D Command

## Synopsis

**OdCmd**(PROBE, A)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **A** | Acquisition tag for 1D data (tag). |

## Description

This function retrieves and returns the command byte from 1D data.

## Result Type/Space

I[1]

## Example

```
; Name         Units  Number  Result   Computations
"FSSPRange"     ""     F100    I[1]    OdCmd(Pr.fssp, Aq.fssp)
```

# OdIVar(), 1D Inverse Velocity Acceptance Ratio

## Synopsis

**OdIVar**(A, STROBEINDEX, TOTALSTROBEINDEX, CFAC, INTERVAL)

| | |
|---|---|
| **A** | Acquisition tag for 1D data (tag). |
| **STROBEINDEX**[1] | Strobe count channel index (integer) |
| **TOTALSTROBEINDEX**[1] | Total strobes count index (integer). |
| **CFAC**[1] | Correction factor. |
| **INTERVAL**[1] | Integration interval (integer). |

## Description

This function computes the inverse velocity acceptance ratio from 1D data. This value can be used to correct concentrations, volumes and mass calculations. The following formula summarizes the computations.

$$f = \frac{A[TOTALSTROBEINDEX]}{A[STROBEINDEX]} \cdot CFAC$$

## Result Type/Space

D[1]

## Example

```
; Name                 Units  Number   Result   Computations
"InverseVelAccRatio"    ""     F200    F[100]   OdIVar(A100, 15, 16, 1, 1)
```

# OdIVarAdv(), 1D Advanced Inverse Velocity Acceptance Ratio

## Synopsis

**OdIVarAdv**(STROBETAG, TOTALSTROBETAG, CFAC, INTERVAL)

| | |
|---|---|
| **STROBETAG** | Total valid strobe counts tag (tag). |
| **TOTALSTROBETAG** | Total strobe counts tag (tag). |
| **CFAC**[1] | Correction factor. |
| **INTERVAL**[1] | Integration interval (integer). |

## Description

This function computes the inverse velocity acceptance ratio from 1D advanced data. This value can be used to correct concentrations, volumes and masses calculations. The following formula summarizes the computations.

$$f = \frac{TOTALSTROBETAG}{STROBETAG} \cdot CFAC$$

## Result Type/Space

D[1]

## Example

```
; Name  Units  Number  Result   Computations
"IVAR"   ""     F100    F[1]    OdIVarAdv(A1006, A1005, 1, 1)
```

# OdRef(), 1D Reference Voltage

## Synopsis

**OdRef**(A)
**A** Acquisition tag for 1D data (tag).

## Description

This function retrieves the reference voltage from 1D data and converts it to volts. This function can be used with 1D data, CAMAC 1D data, SPP100, SPP200, and SPP300 data types

## Result Type/Space

D[1]

## Example

```
; Name                 Units   Number  Result   Computations
"1DReferenceVoltage"    ""      F100    F[1]     OdRef(A100)
```

# OdSums(), 1D Sums

## Synopsis

**OdSums**(A, FIRST, FREQUENCY)
**OdSums**(A, INTERVAL, STATE, FIRST)

| | |
|---|---|
| **A** | 1D acquisition tag number (tag). |
| **FIRST**[1] | Use first bin or skip (integer: 0 or 1). |
| **FREQUENCY**[1] | Frequency of summation. |
| **INTERVAL**[1] | Interval of summation (in display cycles) (integer). |
| **STATE**[1] | Function control variable (integer). |

## Description

This function sums up all 1D samples (i.e. FSSP, ASASP) from a data buffer. This summation is accrued for the specified frequency. At the end of the time interval the sums are returned through the result space and the internal summation values are cleared for the next summation period.

The **STATE** control variable is used to control the function operation mode. If the **STATE** control variable is a '0', then the summation is done every interval. If the **STATE** control variable is a '1', then the sums are accumulated. If the **STATE** control variable is a '2', this causes the last summation value to be held. Any other transition in the control variable, clears the internal summation and starts the accumulation process all over again.

The **FIRST** parameter can be used to avoid returning the summation value for channel zero (this may be desired with the advanced 1D256 interface). Use a value of '0' to return all counts. Use a value of '1' to skip the counts for channel zero.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name          Units  Number  Result   Computations
"FSSPCounts"     ""     F100    F[15]   OdSums(A1001, 0, 1)
```

# PAlt(), Pressure Altitude

## Synopsis

**PAlt**(SPRES)
**SPRES**[*n*] Formula of an array of values containing static pressures (in mb) ($n \geq 1$).

## Description

This function computes pressure altitude from static pressure.

$$f[i] = 1.4545 \times 10^5 \times \left(1 - \left(\frac{SPRES[i]}{1013.25}\right)^{0.1903}\right) ft$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name              Units  Number  Result  Computations
"PressureAltitude"  "ft"   F200    F[1]    PAlt(F100)
```

# PIas(), Pressure Indicated Airspeed

## Synopsis

**PIas**(PPRES)
**PPRES**[*n*] Formula of an array of pitot pressure values (in mb) (*n*>0).

## Description

This function computes pressure indicated air speed from pitot pressure (differential pressure).

$$f[i] = \sqrt{(2.1837\times10^{6})\left(\left(1+\frac{PPRES[i]}{1013.25}\right)^{0.285867}-1\right)}\,\text{knots}$$

$$\text{for } i = 0\ldots[n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units    Number  Result  Computations
"PressureIAS"   "kt"     F200    F[1]    PIas(F100)
```

# Poly(), Polynomial

## Synopsis

**Poly**(X, $A_0$, $A_1$, ..., $A_n$)

| | |
|---|---|
| **X**[*m*] | Formula for X value (*m*≥1). |
| $\mathbf{A_0}$[*p*] | First order coefficient (*p*≥1). |
| $\mathbf{A_1}$[*r*] | Second order coefficient (*r*≥1). |
| $\mathbf{A_n}$[*s*] | *n*th order coefficient (*s*≥1). |

☞ ***Note: The Poly() function can accept a variable number of parameters. However, the function needs at least 3 parameters to work correctly. Specifically, it needs the 'X' parameter and at least two coefficient parameters as specified above. For example,*** `Poly(F100, F200, F300)`.

## Description

This function is used to evaluate an *n*th order polynomial. The function can only compute polynomials of the first or higher order. The polynomial computations are done via factorization for better efficiency. The following formula summarizes the computation.

$$f[i] = \sum_{i=n}^{0} A_i X^i$$

## Result Type/Space

D[*n*], *n* is based on interpolation of the number of elements for each parameter used in the function. The interpolation is done on the size of each parameter used in the factorization process.

## Example

```
; Name          Units Number Result  Computations
"RosTempTotal" ""    F300   F[15]   Poly(F2430, -51.0738, 20.64947, -0.0637105)
```

# PosAvData(), POSAV Data Access

## Synopsis

**PosAvData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for POSAV data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..23). |

## Description

This function allows access to individual items of the POSAV data block. The following table shows the different **SELECT** values for the different POSAV data fields. The function will returns the value of a user specified item from a POSAV buffer. Please check the POSAV manual for further information.

| Data Field | SELECT |
|---|---|
| Time 1 (s) | 0 |
| Time 2 (s) | 1 |
| Distance Tag | 2 |
| Time Type | 3 |
| Distance Type | 4 |
| Latitude (deg) | 5 |
| Longitude (deg) | 6 |
| Altitude (m) | 7 |
| North/South Velocity (m/s) | 8 |
| East/West Velocity (m/s) | 9 |
| Up/Down Velocity (m/s) | 10 |
| Roll (deg) | 11 |
| Pitch (deg) | 12 |
| Heading (deg) | 13 |
| Wander Angle (deg) | 14 |
| Track Angle (deg) | 15 |

**POSAV Data Select**

| Data Field | SELECT |
|---|---|
| Speed (m/s) | 16 |
| Angular Rate Long (deg/s) | 17 |
| Angular Rate Trans (deg/s) | 18 |
| Angular Rate Down (deg/s) | 19 |
| Long Acceleration (deg/s/s) | 20 |
| Trans Acceleration (deg/s/s) | 21 |
| Down Acceleration (deg/s/s) | 22 |
| Status | 23 |

**POSAV Data Select**

## Result Type/Space

F[*1*]

## Example

```
; Name        Units      Number  Result  Computations
"NVelocity"  "m/s"       F1008    F[1]  PosAvData(Aq.PosPrimary, 8)
```

# Power(), Power

## Synopsis

**Power**(A, B)

**A**[*m*] Formula of an array of BASE numbers (*m*>0).

**B**[*p*] Formula of an array of exponents (*p*>0).

*Note: Deprecated [M300 Replacement function - See "pow, Power Function"]*

## Description

This function is used to compute 'A' raised to the 'B' power element by element. The number of values returned and type are dependent on the formula's result/type space. This function uses interpolation [**See Interpolation**]. The following formula summarizes the calculations.

$$f[i] = A[i]^{B[i]}$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*, *p*)

## Example

```
; Name    Units  Number  Result    Computations
"Power"    ""     F300    F[15]    Power(F100, F200)
```

# PqConfig(), Piraq Configuration Data

## Synopsis

**PqConfig**(A, SELECT)

| | |
|---|---|
| **A** | Piraq acquisition tag number (tag). |
| **SELECT**[1] | Selector value (integer). |

## Description

This function allows access to the Piraq configuration data. The configuration data is specified in the board table for the Piraq board entry. The M300 stores this data under the Piraq Config acquisition type.

| SELECT | Data Description | SELECT | Data Description |
|---|---|---|---|
| 0 | Timing Mode | 9 | Delay |
| 1 | Gates | 10 | Hits |
| 2 | Gate Width | 11 | Pulsed Width |
| 3 | Pulse Repetition Time | 12 | Watchdog |
| 4 | First Gate Mode | 13 | Phase Correct Mode |
| 5 | Clutter Filter Mode | 14 | Time Series Mode |
| 6 | Time Series Gate | 15 | Scan Rate |
| 7 | Pulse Rate | 16 | Index of Refraction |
| 8 | Beam Width | 17 | ISP file |

**SELECTOR**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name   Units  Number  Result   Computations
"Gates"   ""     F2101   I[1]    PqConfig(A1005, 2)
```

# PqPower(), Piraq Power

## Synopsis

**PqPower**(A, GATEWIDTH, HITS, SCALE, OFFSET, MODE)

| | |
|---|---|
| **A** | Acquisition tag for Piraq data (tag). |
| **GATEWIDTH**[1] | Gate width (integer). |
| **HITS**[1] | Hits count from Piraq config. (integer). |
| **SCALE**[1] | Scale factor. |
| **OFFSET**[1] | Offset value |
| **MODE**[1] | Operational mode (integer: 0 or 1). |

## Description

This function is used to compute the coherent and incoherent power for the Piraq I, Q, and A acquisition type. The power is returned in dbm units.

If **MODE** is set to 0 (Coherent power), the function uses the '**a**' and '**b**' Piraq data with the following formula.

$$f[i] = SCALE \cdot (5\log_{10}(a_i^2 + b_i^2) - 10\log_{10}(HITS) - 20\log_{10}(GATEWIDTH)) + OFFSET$$

$$\text{for } i = 0 \ldots [n-1]$$

If **MODE** is set to 1 (Incoherent power), the function uses the '**p**' Piraq data with the following formula.

$$f[i] = SCALE \cdot (10\log_{10}(p_i) - 10\log_{10}(HITS) - 20\log_{10}(GATEWIDTH)) + OFFSET$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = number of data samples.

## Example

```
; Name    Units   Number  Result  Computations
"CPower"  "dbm"   F2010   F[200]  PqPower(F2000,F2104,F2103,1.01278,-140.92,0)
```

# PqRange(), Piraq Range

## Synopsis

**PqRange**(GATEWIDTH, GATES)
**PqRange**(GATEWIDTH, GATES, CLOCK)

| | |
|---|---|
| **GATEWIDTH**[1] | Gate width from Piraq config (integer). |
| **GATES**[1] | Number of gates from Piraq config (integer). |
| **CLOCK**[1] | Clock period. |

## Description

This function computes the range, in meters, from the given Piraq configuration data. Note that the **CLOCK** value must be the period (1/frequency), not the frequency.

$$f = CLOCK \cdot (2.99792458 \times 10^8) \cdot (0.5) \cdot (i + 0.5) \cdot GATEWIDTH$$
$$\text{for } i = 0 \ldots [n - 1]$$

***Note: If the CLOCK argument is omitted, a period constant of $1.25 \times 10^{-7}$ (8 MHz) is used. Also the constant $2.99792458 \times 10^8$ is for the speed of light.***

## Result Type/Space

D[*n*], *n* = **GATES**

## Example

```
; Name         Units   Number  Result  Computations
"PiraqRange"    "m"    F2000   F[200]  PqRange(F2102, F2102) F1310 *
```

# PqRaw(), Piraq Raw Data

## Synopsis

**PqRaw**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for Piraq data (tag). |
| **SELECT**[1] | Data select (integer) |

## Description

This function is used to access the Piraq I, Q&A raw data (See "Type 100 (PIRAQ I, Q and P)") The function will return either a, b, or p data based on the **SELECT** argument.

| SELECT | Type |
|---|---|
| 0 | a |
| 1 | b |
| 2 | p |

**SELECT**

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name  Units  Number   Result  Computations
"a"      ""     F2001   F[200]  PqRaw(A2000, 0)
"b"      ""     F2002   F[200]  PqRaw(A2000, 1)
"c"      ""     F2003   F[200]  PqRaw(A2000, 2)
```

# PqReflectivity(), Piraq Reflectivity

## Synopsis

**PqReflectivity**(POWER, RANGE, RADARCONST)

| | |
|---|---|
| **POWER**[*m*] | Formula for Piraq power (*m*≥1). |
| **RANGE**[*p*] | Formula for Piraq range (*p*≥1). |
| **RADARCONST**[1] | Radar constant. |

## Description

This function computes Piraq reflectivity in dBz from power, range and the radar constant. The following formula summarizes the computation.

$$f[i] = POWER[i] + 20 \cdot \log_{10}(RANGE[i]) + RADARCONST$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \min(m, p)$

## Example

```
; Name            Units   Number  Result  Computations
"Reflectivity"   "dbz"    F2030   F[200]  PqReflectivity(F2020, F2000, -38.0)
```

# PqStatus(), Piraq Status

## Synopsis

**PqStatus**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for Piraq status data (tag). |
| **SELECT**[1] | Data select (integer: 0..18) |

## Description

This function allows access to the Piraq status data. This data is read from the Piraq board dual port RAM and stored in the M300 via the Piraq status acquisition type. The following table shows which data 15 are retrieved based on the **SELECT** value.

| SELECT | Data Type | SELECT | Data Type |
|---|---|---|---|
| 0 | New buffer | 10 | Denominator discriminator |
| 1 | Gates | 11 | First gate inverse magnitude |
| 2 | Hits | 12 | Spare 1 |
| 3 | Gate width | 13 | Spare 2 |
| 4 | First gate mode | 14 | Spare 3 |
| 5 | Phase correct mode | 15 | Spare 4 |
| 6 | Clutter filter mode | 16 | Status |
| 7 | Time series mode | 17 | Pulse width |
| 8 | Time series gate | 18 | Data format |
| 9 | Numerator discriminator | | |

**SELECT**

## Result Type/Space

I[*n*], *n* = number of data samples

## Example

```
; Name         Units    Number  Result    Computations
"GateWidth"     ""       F2203    I[1]     PqStatus(A2002, 3)
```

# PrData(), Probe Data

## Synopsis

**PrData**(PROBE, SELECT)
**PrData**(PROBE, RANGE, SELECT)

| | |
|---|---|
| **PROBE** | Probe from probe table (probe). |
| **SELECT**[1] | Probe field select (integer: 0...7). |
| **RANGE**[1] | Range for probe (integer: 0...15). |

## Description

This function is used to retrieve certain probe data from the probe channel files. This data may be used during run time for computations or displays. The user does not have to specify the range. The range value is retrieved from the probe entry and the selector value determines which type of probe data will be returned. (See "Probe Table, (prb.300)" and See "Probe Channel File, (*.prb)".)

| SELECT | Probe Data |
|---|---|
| 0 | min |
| 1 | max |
| 2 | middle |
| 3 | dD |
| 4 | dlogD |
| 5 | area |
| 6 | volume |
| 7 | Sample Area |
| 8 | Pixel Size |

**SELECT**

## Result Type/Space

D[*n*], *n* = number of channels in probe entry (**PROBE**)

## Example

```
; Name          Units   Number  Result   Computations
"MaximumSize"    ""      F105    F[15]   PrData(Pr.Fssp, 1)
```

# ProbeData() Probe Data

## Synopsis

**ProbeData**(PROBE, RANGE, SELECT)

| | |
|---|---|
| **PROBE** | Probe name from probe table (probe). |
| **RANGE**[1] | Probe range (integer). |
| **SELECT**[1] | Probe field select (integer: 0..7). |

☞ ***Note: Deprecated [M300 Function replacement - See "PrData(), Probe Data"]***

## Description

This function is used to retrieve certain probe data from the probe channel files. This data may be used during run time for computations or displays. The **RANGE** value is used to index which channel values will be used. The **SELECT** parameter determines which type of probe data will be returned as seen in the table below.

| SELECT | Probe Data |
|---|---|
| 0 | min |
| 1 | max |
| 2 | middle |
| 3 | dD |
| 4 | dlogD |
| 5 | area |
| 6 | volume |
| 7 | Sample Area |

**Table 8: SELECT**

## Result Type/Space

D[*n*], *n* = number of channels in Probe entry (**PROBE**)

## Example

```
; Name          Units  Number  Result  Computations
"MaximumSize"     ""     F105   F[15]    ProbeData(100, 0, 1)
```

# PromoBins() Promo Bins

## Synopsis

**PromoBins**(AMP, TTIME, AMPBINS, POINTS, TTIMEMIN, TTIMEMAX)
**PromoBins**(TTIME, TTIMEBINS, POINTS, TTIMEMIN, TTIMEMAX)

| | |
|---|---|
| **AMP**[POINTS] | Amplitude data (float). |
| **TTIME**[POINTS] | Transit Time data (float). |
| **AMPBINS**[n] | Amplitude bin data (float). |
| **TTIMEBINS**[m] | Transit Time bin data (float). |
| **POINTS**[1] | Data points (integer). |
| **TTIMEMIN**[1] | Transit Time Minimum (float). |
| **TTIMEMAX**[1] | Transit Time Maximum (float). |

## Description

This function can be used to compute the number of samples by bins for amplitude (AMP) and transit time (TTIME).

The POINTS are the total number of data samples from the Promo2000 data (see PromoData function).

## Result Type/Space

D[*n*], *n* = number of bins for amplitude.
D[*m*], *m* = number of bins for transit time.

## Example

```
; Name   Units Number Result Computations
"AmpCounts" "" F1000  F[51]  PromoBins(F20100, F20101, F20010, F20102, 0, 100)
```

# PromoData() Promo Data

## Synopsis

**PromoData**(A, SELECT)
**A** Acquisition tag for promo data (tag).
**SELECT**[1] Promo data field select (integer: 0..2).

## Description

This function is used to get the Promo2000 data from the recorded data tag (A).
The following table shows the valid SELECT values for the Promo2000 data.

| SELECT | Name | Result |
|---|---|---|
| 0 | Amplitude | p >= data points |
| 1 | Transit Time | p >= data points |
| 2 | Data Points | 1 |

**Table 9: SELECT**

## Result Type/Space

D[*n*] F[n], L[n], l[n], I[n], i[n], *n* = see select table above.

## Example

```
; Name          Units  Number  Result  Computations
"AmplitudeData"  ""    F1000 F[7000]  PromoData(Aq.Promo2000Data, 0)
```

# Protect(), Protect Values

## Synopsis

**Protect**(A, B)

| | |
|---|---|
| **A**[*n*] | Formula of an array of values (*n*≥1). |
| **B**[*1*] | Comparison value. |

## Description

This function is intended to protect the value of a formula from going near zero so that the formula can be used as the denominator in divisions.

The **A**[*n*] parameter represents the values to be protected.

The **B** parameter is the value near zero for the comparison (this is typically 1.0e-6). Any values in the formula **A** that are bellow **B**, will be set to **B**.

## Result Type/Space

D[n], F[n], L[n], l[n], I[n], i[n]

## Example

```
; Name     Units  Number  Result  Computations
"Protect"     ""     F300   F[1]  Protect(F300, 1.0e-6)
```

# PrTasClockIn(), Probe TAS Clock In

## Synopsis

**PrTASClockIn**(A)

**A** Acquisition tag for the TAS factors data (tag).

## Description

This function retrieves the multiply and divide factor from the data based on the user specified tag. It then uses the retrieved multiply and divide factors to calculate and return the TAS clock frequency (in MHz). The following formula summarizes the computation.

$$f = \frac{(MULFAC \cdot 0.05)}{DIVFAC}$$

## Result Type/Space

D[1]

## Example

```
; Name           Units    Number   Result   Computations
"TASClockFreq"   "MHz"    F100     F[1]     PrTasClockIn(A100)
```

# PrTasClockOut(), Probe TAS Clock Out

## Synopsis

**PrTASClockOut**(PROBE, TAS)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **TAS**[1] | True air speed. |

## Description

This function calculates the TAS clock frequency based on the user specified TAS. Upon completion this function returns the calculated TAS clock frequency in MHz. This function updates the internal probe value for y-size, which will be used by other functions. The TAS values is first limited based on the TAS limit value from the probe entry. The frequency value, generated by this function, should be used as an input to the Control TAS functions, such as Co2DTAS(). The following formulas summarize the computation.

$$FREQUENCY = \frac{TAS}{SIZE}$$

$$ySize = \frac{TAS}{FREQUENCY}$$

$$f = FREQUENCY$$

## Result Type/Space

D[1]

## Example

```
; Name           Units    Number  Result  Computations
"TASClockOut"    "MHz"    F101    F[1]    PrTasClockOut(Pr.2dc, F200)
```

# PTas(), Pressure Airspeed

## Synopsis

**PTAS**(STEMP, PPRES, SPRES)

| | |
|---|---|
| **STEMP**[*m*] | Static temperature (in C) (*m*≥1). |
| **PPRES**[*p*] | Pitot pressure (in mb) (*p*≥1). |
| **SPRES**[*r*] | Static pressure (in mb) (*r* ≥1). |

## Description

This function computes pressure true air speed (in meters/second) from static temperature, pitot pressure, and static pressure. Function uses interpolation [**See Interpolation**]**.** The following formula summarizes the computation.

$$f[i] = \sqrt{2009.6 \times (STEMP[i] + 273.15) \times \left(\left(1 + \frac{PPRES[i]}{SPRES[i]}\right)^{0.285867} - 1\right)} \text{ (m/s)}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*,*p*,*r*)

## Example

```
; Name           Units   Number  Result  Computations
"PressureTAS"    "m/s"   F200    F[1]    PTas(F100, F101, F102)
```

# RaConstant(), Radar Constant

## Synopsis

**RaConstant**(RADAR, WAVEGUIDELOSS, K2)

| | |
|---|---|
| **RADAR** | Radar entry (radar). |
| **WAVEGUIDELOSS**[1] | Waveguide loss value (in db). |
| **K2**[1] | K2 value. |

## Description

This function computes and returns the radar constant based on the parameters **WAVEGUDELOSS**, **K2** and data stored in the radar entry specified by **RADAR**. The data used from the radar entry are the following: waveguide loss (in db), wavelength (in meters), receiver gain (in db), transmit power (in kw), antenna gain (in db), horizontal beam width (in degrees), vertical beam width (in degrees), and pulse rate (in μsec). The following formula summarizes the computation.

$$f_1 = \left[2 \cdot WAVEGUIDELOSS + \left(20 \cdot \log_{10}\left(\frac{2.99792458\times10^8}{FREQUENCY}\right)\right) + (10 \cdot \log_{10}22.092) + 100\right] \cdot RECEIVERGAIN$$

$$f_2 = f_1 + \left[(10 \cdot \log_{10}(TRANSMITPOWER) + 60) + (2 \cdot RECEIVERGAIN) + \left(10\log_{10}\left(HORIZONTALBEAMWIDTH \cdot \frac{\pi}{180}\right)\right)\right]$$

$$f = f_2 + \left[\left(10 \cdot \log_{10}\left(VERTICALBEAMWIDTH \cdot \frac{\pi}{180}\right)\right) + (10 \cdot \log 10(PULSERATE \cdot 2.99792458\times10^8)) + (10 \cdot \log_{10}K2)\right]$$

☞ ***Note: The constant 2.99792458x10$^8$ is for the speed of light***

## Result Type/Space

D[1]

## Example

```
; Name           Units  Number  Result   Computations
"RadarConstant"   ""     F2000   F[1]    RaConstant(F2102, F2103, 20.4)
```

# Rand(), Random

## Synopsis

**Rand**(SELECT)
**SELECT**[1] Select for return type (integer).

## Description

This function returns random values. If **SELECT** is zero, it will return a random integer in the range [0, $2^{32}$]. If **SELECT** is non-zero, it will return a random normalized floating point value (normalized value has a range [0.0, 1.0]).

## Result Type/Space

D[*n*], *n* = size of the result space entry

## Example

```
; Name      Units  Number  Result   Computations
"Rand"       ""     F200    I[20]   Rand(0)
```

# RandData(), Random Data

## Synopsis

**RandData**(SCALE, OFFSET, MINIMUM, MAXIMUM )

| | |
|---|---|
| **SCALE**[1] | Scaling value. |
| **OFFSET**[1] | Offset value. |
| **MINIMUM**[1] | Lower limit. |
| **MAXIMUM**[1] | Upper limit |

## Description

This function calls Rand() (See "Rand(), Random") and uses the random value with **SCALE** and **OFFSET** to generate a new value that will always lie between **MINIMUM** and **MAXIMUM**. The following formula summarize the computation.

$$f[i] = Rand(1) \cdot SCALE + OFFSET$$
$$\text{if } (f[i] < MINIMUM) \text{ then } f[i] = MINIMUM$$
$$\text{else if } (f[i] > MAXIMUM) \text{ then } f[i] = MAXIMUM$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = size of the result space entry

## Example

```
; Name         Units  Number  Result    Computations
"RandomTemp"    ""     F201    F[20]    RandData(2.25, 32, 0, 100)
```

# RandSeed(), Random Seed

## Synopsis

**RandSeed**(SEED)
**SEED**[1] Random number generator seeding value (unsigned integer).

## Description

This function uses **SEED** as the new seeding value for any subsequent calls to Rand() and RandData(). Upon successful completion the function will return the **SEED** value.

## Result Type/Space

I[1]

## Example

```
; Name      Units  Number  Result   Computations
"RandSeed"   ""     F400    I[1]    RandSeed(F399)
```

# Range(), Range

## Synopsis

**Range**(REFLAT, REFLON, LAT, LON)

| | |
|---|---|
| **REFLAT**[*m*] | Latitudes of reference points (from) (in radians) ($m \geq 1$). |
| **REFLON**[*p*] | Longitudes of reference points (from) (in radians) ($p \geq 1$). |
| **LAT**[*r*] | Latitudes of target points (to) (in radians) ($r \geq 1$). |
| **LON**[*s*] | Longitudes of target points (to) (in radians) ($s \geq 1$). |

## Description

This function computes the distance in nautical miles between two points, usually a reference position and an aircraft's current position.

$$\Delta LATITUDE[i] = (LAT[i] - REFLAT[i]) \cdot 3.4377467 \times 10^3$$

$$\Delta LONGITUDE[i] = (LON[i] - REFLON) \cdot 3.4377467 \times 10^3 \cdot \cos(LAT)$$

$$f[i] = \sqrt{\Delta LATITUDE^2 + \Delta LONGITUDE^2}(\text{nmi})$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \min(m, p, r, s)$

## Example

```
; Name      Units   Number  Result  Computations
"Range"     "nmi"   F300    F[1]    Range(F100, F101, F200, F201)
```

# Ref1D(), 1D Reference Voltage

## Synopsis

**Ref1D**(A)
**A** Acquisition tag for 1D data (tag).
*Note: Deprecated [M300 Replacement function - See "OdRef(), 1D Reference Voltage"]*

## Description

This function retrieves the reference voltage from 1D data and converts it to volts. This function can be used with 1D, CAMAC 1D, SPP100, SPP200, and SPP300 data types

## Result Type/Space

D[1]

## Example

```
; Name                 Units   Number  Result   Computations
"1DReferenceVoltage"    ""      F100    F[1]     Ref1D(A100)
```

# RHToDewPoint(), Relative Humidity to Dew Point

## Synopsis

**RHToDewPoint**(RH, TEMP)

| | |
|---|---|
| **RH**[*n*] | Formula for relative humidity (*n*>0). |
| **TEMP**[1] | Formula for outside air temperature value. |

## Description

This function calculates an approximation of the dewpoint based on the **RH** (relative humidity) and **TEMP** (outside air temperature) arguments passed. The following formulas demonstrate the calculations performed.

$$f_1[i] = 1 - (0.01 + RH[i])$$

$$f_2[i] = TEMP - (14.55 + 0.114 \cdot TEMP) \cdot f_1[i] + [(2.5 + 0.007 \cdot TEMP) \cdot f_1[i]]^3 - ((15.9 + 0.117 \cdot TEMP) \cdot f_1[i]^{14})$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name      Units   Number  Result  Computations
"DewPoint" " °c"     F4007   F[1]   RHToDewPoint(F3007, F3006)
```

# Scale(), First Order Scaling

## Synopsis

**Scale**(X, A, B)

| | |
|---|---|
| **X**[*m*] | Formula of an array of elements to be scaled (*m*≥1). |
| **A**[*p*] | Formula of an array of gain values (*p*≥1). |
| **B**[*r*] | Formula of an array of offset value (*r*≥1). |

## Description

This function returns an array of values or single value representing the linear scale. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i] \cdot X[i] + B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m, p, r*)

## Example

```
; Name   Units  Number  Result  Computations
"Scale"   ""     F200    F[10]   Scale(F100, F101, F102)
```

# Scale2(), Second Order Scaling

## Synopsis

**Scale2**(X, A, B, C)

| | |
|---|---|
| **X**[*m*] | Formula of an array of elements to be scaled (*m*≥1). |
| **A**[*p*] | Formula of an array of second order coefficients (*p*≥1). |
| **B**[*r*] | Formula of an array of first order coefficients (*r*≥1). |
| **C**[*s*] | Formula of an array of offset values (*s*≥1). |

## Description

This function returns an array of values representing the second order scale. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i]X[i]^2 + B[i]X[i] + C[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p, r, s)$

## Example

```
; Name      Units  Number  Result  Computations
"Scale2"    ""     F200    F[10]   Scale2(F100, F101, F102, F103)
```

# Scale3(), Third Order Scaling

## Synopsis

**Scale3**(X, A, B, C, D)

| | |
|---|---|
| **X**[*m*] | Formula of an array of elements to be scaled (*m*≥1). |
| **A**[*p*] | Formula of an array of third order coefficients (*p*≥1). |
| **B**[*r*] | Formula of an array of second order coefficients (*r*≥1). |
| **C**[*s*] | Formula of an array of first order coefficients (*s*≥1). |
| **D**[*t*] | Formula of an array of offset values (*t*≥1). |

## Description

This function returns an array of values representing the third order scale. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i]X[i]^3 + B[i]X[i]^2 + C[i]X[i] + D[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m, p, r, s, t*)

## Example

```
; Name       Units  Number  Result  Computations
"Scale3"     ""     F200    F[10]   Scale3(F100, F101, F102, F103, F104)
```

# ScaleArray(), First Order Array Scaling

## Synopsis

**ScaleArray**(X, A, B)

| | |
|---|---|
| **X**[*m*] | Formula of an array of elements to be scaled (*m*≥1). |
| **A**[*p*] | Formula of an array of gain values (*p*≥1). |
| **B**[*r*] | Formula of an array of offset values (*r*≥1). |

***Note: Deprecated [M300 Replacement function - See "Scale(), First Order Scaling"]***

## Description

This function returns an array of values or single value representing the linear scale. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i] \cdot X[i] + B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*, *p*, *r*)

## Example

```
; Name         Units  Number   Result  Computations
"ScaleArray"   ""      F200    F[10]   ScaleArray(F100, F101, F102)
```

# ScaleArray2(), Second Order Array Scaling

## Synopsis

**ScaleArray2**(X, A, B, C)

| | |
|---|---|
| **X**[*m*] | Formula of an array of elements to be scaled (*m*≥1). |
| **A**[*p*] | Formula of an array of second order coefficients (*p*≥1). |
| **B**[*r*] | Formula of an array of first order coefficients (*r*≥1). |
| **C**[*s*] | Formula of an array of offset values (*s*≥1). |

*Note: Deprecated [M300 Replacement function - See "Scale2(), Second Order Scaling"]*

## Description

This function returns an array of values representing the second order scale. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i]X[i]^2 + B[i]X[i] + C[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m, p, r, s*)

## Example

```
; Name          Units  Number  Result   Computations
"ScaleArray2"    ""     F200    F[10]   ScaleArray2(F100, F101, F102, F103)
```

# ScaleArray3(), Third Order Array Scaling

## Synopsis

**ScaleArray3**(X, A, B, C, D)

| | |
|---|---|
| **X**[*m*] | Formula of an array of elements to be scaled (*m*≥1). |
| **A**[*p*] | Formula of an array of third order coefficients (*p*≥1). |
| **B**[*r*] | Formula of an array of second order coefficients (*r*≥1). |
| **C**[*s*] | Formula of an array of first order coefficients (*s*≥1). |
| **D**[*t*] | Formula of an array of offset values (*t*≥1). |

*Note: Deprecated [M300 Replacement function - See "Scale3(), Third Order Scaling"]*

## Description

This function returns an array of values representing the third order scale. This function uses interpolation [**See Interpolation**]. The following formula summarizes the computations.

$$f[i] = A[i]X[i]^3 + B[i]X[i]^2 + C[i]X[i] + D[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*, *p*, *r*, *s*, *t*)

## Example

```
; Name         Units  Number  Result  Computations
"ScaleArray3"  ""     F200    F[10]   ScaleArray3(F100, F101, F102, F103, F104)
```

# Seconds(), Seconds

## Synopsis

**Seconds**(A0)

**A0** Date time acquisition tag number (0 always) (tag).

## Description

This function is used to get seconds since midnight.

## Result Type/Space

D[1]

## Example

```
; Name      Units   Number  Result  Computations
"Seconds"   ""      F100    F[1]    Seconds(A0)
```

# SerialASCII(), Serial ASCII

## Synopsis

**SerialASCII**(A, INDEX, DELIMITER, COUNT, MODE)

| | |
|---|---|
| **A** | Acquisition tag for Serial ASCII data (tag). |
| **INDEX**[1] | Index of value in serial data (integer, 1, 2, 3, 4, ...). |
| **DELIMITER**[1] | ASCII byte value used as data delimiter between ASCII data (integer). |
| **COUNT**[1] | Number of data values to be returned starting from this index (integer). |
| **MODE**[1] | Mode option for data type (integer: 0 or 1) |

*Note: Deprecated [M300 Replacement function - See "SrASCII(), Serial ASCII"]*

## Description

This function gets the data at the index specified in the ASCII data block. The **MODE** parameter is used to specify decimal (zero) or hexadecimal (one) data for integer and long types.

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name           Units  Number  Result  Computations
"SerialASCII"    ""     F100    I[5]    SerialASCII(A100, 10, 44, 5, 0)
```

# SerialDADS(), Serial DADS

## Synopsis

**SerialDADS**(A, INDEX, IDENTIFIER)

| | |
|---|---|
| **A** | Acquisition tag for Serial DC 8 DADS ASCII data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **IDENTIFIER**[1] | ASCII byte value for identifier between ASCII data blocks (integer). |

***Note: Deprecated [M300 function replacement - See "SrDADS(), Serial DADS"]***

## Description

This function is used to retrieve specific data fields from a block of DC 8 DADS ASCII data. Different data fields are separated with spaces. The identifier for each block needs to be specified, in order to retrieve data from the appropriate data block.

## Result Type/Space

D[*n*], *n* = starting at **INDEX** until the byte value **IDENTIFIER** is reached

## Example

```
; Name          Units  Number  Result   Computations
"SerialDADS"     ""     F100    F[5]    SerialDADS(A100, 10, 65)
```

# SerialIEEE(), Serial IEEE

## Synopsis

**SerialIEEE**(A, INDEX, COUNT)

| | |
|---|---|
| **A** | Acquisition tag for Serial IEEE data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **COUNT**[1] | Number of data values for this index (integer). |

*Note: Deprecated [M300 Replacement function - See "SrIEEE(), Serial IEEE"]*

## Description

This function gets IEEE data at the index specified in the data block. It works with data from either serial IEEE data type or the DRV11 data type. Make sure that parameter one in the acquisition table for these types indicates the appropriate data swap option for different machine types.

An index value of 1 represents the first value in the data (index base 1, not base 0).

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name          Units  Number  Result   Computations
"SerialIEEE"     ""     F100    F[5]    SerialIEEE(A100, 10, 5)
```

# SerialInteger(), Serial Integer

## Synopsis

**SerialInteger**(A, INDEX, COUNT)

| | |
|---|---|
| **A** | Acquisition tag for Serial integer data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **COUNT**[1] | Number of data values for this index (integer). |

*Note: Deprecated [M300 Replacement Function - See "SrInteger(), Serial Integer"]*

## Description

This function is used to retrieve specific data values from a block of integer data. This function can be used in a data block from either the serial integer data type or the DRV11 data type. Make sure that parameter one in the acquisition table for these types, indicates the appropriate integer type (16 bit or 32 bit integer) and data swap options.

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name            Units  Number  Result   Computations
"SerialInteger"    ""      F100    I[5]    SerialInteger(A100, 10, 5)
```

# SerialVAX(), Serial VAX

## Synopsis

**SerialVAX**(A, INDEX, COUNT)

| | |
|---|---|
| **A** | Acquisition tag for SerialVax data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **COUNT**[1] | Number of data values for this index (integer). |

*Note: Deprecated [M300 Function replacement - See "SrVAX(), Serial VAX"]*

## Description

This function is used to retrieve specific IEEE data values from a block of VAX float data. The data stored is unchanged, however, the data displayed is swapped and the exponent is decremented by two in order to obtain the desired value.

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name         Units  Number  Result   Computations
"SerialVAX"     ""     F100    F[5]    SerialVax(A100, 10, 5)
```

# Set(), Set

## Synopsis

**Set**(INIT)
**Set**(INIT, INC)
**Set**(INIT, INC, COUNT)

| | |
|---|---|
| **INIT**[1] | Initialization value. |
| **INC**[1] | Increment value to add to previous value. |
| **COUNT**[1] | Number of values (integer). |

## Description

This function can be used to easily initialize a formula value.

The INIT variable specifies the value to initialize the formula to. If the formula is an array of values, then all values in the array are set to the INIT value.

If the formula is an array of values, then the INC variable specifies the increment to use after the first value. For example, if INIT is 0 and INC is -5, then the array would have a sequence of values such as 0, -5, -10, etc.

Typically the entire array of values is initialized by the function. The COUNT variable can be used to limit the number of values initialized in the array.

## Result Type/Space

Result space varies and is set by the formula used.

## Example

```
; Name           Units  Number  Result   Computations
"AmplitudeCounts"  ""    F1000   F[51]   Set(0)
```

# Sizes(), Sizes

## Synopsis

**Sizes**(PROBE, RANGE)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **RANGE**[1] | Size range in table (integer). |

*Note: Deprecated [M300 Function Replacement - See "PrData(), Probe Data"]*

## Description

This function is used to retrieve the middle channel sizes (**MIDSIZE**) from a probe definition table and store in a array for other formula and display routines to use. To get the other parameters for the probe table use the PrData() function, (See "PrData(), Probe Data"). See "Probe Table, (prb.300)" in Table Reference on page 620. and See "Probe Channel File, (*.prb)" in Table Reference on page 622.

## Result Type/Space

D[*n*], *n* = number of channels in probe entry (**PROBE**)

## Example

```
; Name     Units  Number  Result  Computations
"Sizes"     ""     F100    F[15]   Sizes(P0, F99)
```

# Skip(), Skip

## Synopsis

**Skip**(VALUE, SKIPTO)
**VALUE**[1] Conditional formula/value (integer).
**SKIPTO**[1] Formula the M300 will skip to.

## Description

This function is used to skip to a formula. If **VALUE** is true (nonzero), the next formula executed will be the formula given in **SKIPTO**. Note that if the formula does not exists (**SKIPTO**), the M300 will ignore this function. Also, the function only works within a trigger block. In other words, until the next trigger command. If the formula is not found, then all formulas are skipped until the next trigger (or the end of the formula table). It should be noted that the use of the skip function can increase the complexity level of the formula table. Please use with care, and only when needed. The function will return **VALUE** upon completion.

## Result Type/Space

I[1]

## Example

```
; Name   Units  Number  Result   Computations
"Skip"    ""     F100    I[1]    Skip(F201, F600)
```

# Slope(), Return Slope of a Line

## Synopsis

**Slope**(KNOWNYS, KNOWNXS, STATE)
**KNOWNYS**[n] Known Y values.
**KNOWNXS**[n] Known X values.
**STATE**[1] Function control variable (integer).

## Description

This function returns the slope of a linear regression line through the given data points.

The **STATE** control variable is used to control the function operation mode. If the **STATE** control variable is a '0', then the function performs a reset and computes a new value. If the **STATE** control variable is a '1', then new data is processed and a intercept value returned. If the **STATE** control variable is a '2', this cause the last intercept value to be held.

## Result Type/Space

D[*1*]

## Example

```
; Name          Units  Number  Result   Computations
"Slope"           ""     F100    F[1]   Slope(F1000, F2000, F200)
```

# SpData(), SPP/CDP Data

## Synopsis

**SpData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for SPP100, SPP200, SPP300, CDP, CDPPBP (tag). |
| **SELECT**[1] | Data select (integer). |

## Description

This function is used to retrieve all samples for the selected data item from the SPP100, SPP200, SPP300, CDP and CDPPBP data structure. Only one data item may be returned at a time into a particular formula number. Specify the number of desired samples in the formula space. The OdSums(), 1D Sums , OdRef(), 1D Reference Voltage and OdCmd(), 1D Command should used to compute and retrieve the counts, reference voltage and command (range) for the SPP.

The following table shows the possible values for the **SELECT** parameter and the corresponding returned data element. Not all SPP/CDP Probes have the same data fields. Please check the SPP/CDP probe manual for further information on the fields that are supported by your probe.

| SELECT | SPP100 Data Item |
|---|---|
| 0-7 | Analog channels 0-7 (raw counts) |
| 0x80-0x87 | Analog channels 0-7 (volts) |
| 8 | Reject depth of field |
| 9 | Reject average transit |
| 10 | Average transit |
| 11 | FIFO Full |
| 12 | Reset Flag |
| 13 | ADC Overflow |
| 14 | Samples |
| 15 | Sync Error A |
| 16 | Sync Error B |
| 17 | Sync Error C |
| 18 | Time of First Detected Particle |

**SELECT**

| SELECT | SPP100 Data Item |
|---|---|
| 19 | ADC Counts |
| 20 | Timing |
| 21 | Inter Particle Time (IPT) |

**SELECT**

For a description of these data items, check the SPP/CDP manual.

The house keeping data (analog channels), varies form probe to probe. The best source for information on these is again the probe manual.

## Result Type/Space

D[*n*], *n* = number of channels in probe entry

## Example

```
; Name           Units  Number  Result   Computations
"AverageTransit"  ""     F100    F[10]   SPData(A100, 10)
```

# SPP100Data(), SPP100 Data Retrieve

## Synopsis

**SPP100Data**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for SPP100 data (tag). |
| **SELECT**[1] | Data select (integer). |

*Note: Deprecated [M300 Replacement function - See "SpData(), SPP/CDP Data"]*

## Description

This function is used to retrieve all samples for the selected data item from the SPP100 data structure. Only one data item may be returned at a time into a particular formula number. Specify the number of desired samples in the formula space. The SUMS1D(), REF1D() and CMD1D() functions should used to compute and retrieve the counts, reference voltage and command (range) for the SPP100.

The following table shows the possible values for the **SELECT** parameter and the corresponding returned data element.

| SELECT | SPP100 Data Item |
|---|---|
| 0-7 | Analog channels 0-7 (raw counts) |
| 0x80-0x87 | Analog channels 0-7 (volts) |
| 8 | Reject depth of field |
| 9 | Reject average transit |
| 10 | Average transit |
| 11 | FIFO Full |
| 12 | Reset Flag |
| 13 | ADC Overflow |

**Table 10: SELECT**

For a description of these data items, check the SPP100 manual.

The following table shows the description of each analog channel.

| Description | Channel |
|---|---|
| Signal A, A[0] | 0 |
| Mask A, A[1] | 1 |
| Aux S3, A[2] | 2 |
| Aux S4, A[3] | 3 |
| Laser Reference, A[4] | 4 |
| Aux S1, A[5] | 5 |
| Aux S2, A[6] | 6 |
| Internal Temperature, A[7] | 7 |

**Table 11: Analog Channels**

## Result Type/Space

D[*n*], *n* = number of channels in probe entry

## Example

```
; Name             Units  Number  Result   Computations
"AverageTransit"    ""     F100    F[10]   SPP100Data(A100, 10)
```

# SrASCII(), Serial ASCII

## Synopsis

**SrASCII**(DATA, INDEX, DELIMITER, COUNT, MODE)

| | |
|---|---|
| **DATA** | Acquisition tag for Serial ASCII data or formula (tag or formula). |
| **INDEX**[1] | Index of value in serial data (integer, 1, 2, 3, 4, 5, ...). |
| **DELIMITER**[1] | ASCII byte value used as data delimiter between ASCII data (integer). |
| **COUNT**[1] | Number of values to be returned starting from this index (integer). |
| **MODE**[1] | Mode option for data type (integer: 0 or 1) |

## Description

This function gets the data at the index specified in the ASCII data block. The **MODE** parameter is used to specify decimal (zero) or hexadecimal (one) data for integer and long types.

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name           Units  Number  Result  Computations
"SerialASCII"    ""     F100    I[5]    SrASCII(A100, 10, 44, 5, 0)
```

# SrDADS(), Serial DADS

## Synopsis

**SrDADS**(A, INDEX, IDENTIFIER)

| | |
|---|---|
| **A** | Acquisition tag for Serial DC 8 DADS ASCII data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **IDENTIFIER**[1] | ASCII byte value for identifier between ASCII data blocks (integer). |

## Description

This function is used to retrieve specific data fields from a block of DC 8 DADS ASCII data. Different data fields are separated with spaces. The **IDENTIFIER** for each block needs to be specified, in order to retrieve data from the appropriate data block.

## Result Type/Space

D[*n*], *n* = starting at **INDEX** until the byte value **IDENTIFIER** is reached

## Example

```
; Name      Units  Number  Result  Computations
"SrDADS"    ""     F100    F[5]    SrDADS(A100, 10, 65)
```

# SrData(), Serial Data Function

## Synopsis

**SrData**(A, OFFSET, COUNT, MODE, SWAP)

| | |
|---|---|
| **A** | Acquisition tag for Serial data (tag). |
| **OFFSET**[1] | Byte offset into data (integer). |
| **COUNT**[1] | Number of data values to be returned (integer). |
| **MODE**[1] | Mode selector (integer) |
| **SWAP**[1] | Swap buffers selector (integer: 0 or 1) |

## Description

This function is used to access specific binary data from a raw data block. The **OFFSET** parameter selects the start of the data. The **COUNT** parameter selects the number of data values to be returned. The data can be swapped in necessary. The **MODE** parameter is used to select the data type.

| MODE | Data Type | Bytes |
|---|---|---|
| 2 | Char | 1 |
| 3 | Unsigned Char | 1 |
| 4 | Integer | 2 |
| 5 | Unsigned Integer | 2 |
| 6 | Long | 4 |
| 7 | Unsigned Long | 4 |
| 8 | Float | 4 |
| 9 | Double Float | 8 |

**MODE**

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name      Units  Number  Result  Computations
"StaticPress" "mb"   F100    F[5]  SrData(A100, 100, 5, 8, 0)
```

# SrIEEE(), Serial IEEE

## Synopsis

**SrIEEE**(A, INDEX, COUNT)

| | |
|---|---|
| **A** | Acquisition tag for Serial IEEE data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **COUNT**[1] | Number of data values for this index (integer). |

## Description

This function gets IEEE data at the index specified in the data block. It works with data from either serial IEEE data type or the DRV11 data type. Make sure that parameter one in the acquisition table for these types indicates the appropriate data swap option for different machine types.

An index value of 1 represents the first value in the data (index base 1, not base 0).

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name     Units  Number  Result   Computations
"SrIEEE"    ""     F100    F[5]    SrIEEE(A100, 10, 5)
```

# SrInteger(), Serial Integer

## Synopsis

**SrInteger**(A, INDEX, COUNT)

| | |
|---|---|
| **A** | Acquisition tag for Serial Integer data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **COUNT**[1] | Number of data values for this index (integer). |

## Description

This function is used to retrieve specific data values from a block of integer data. This function can be used in a data block from either the serial integer data type or the DRV11 data type. Make sure that parameter one in the acquisition table for these types, indicates the appropriate integer type (16 bit or 32 bit integer) and data swap options.

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name          Units  Number  Result  Computations
"SrInteger"     ""     F100    F[5]    SrInteger(A100, 10, 5)
```

# SrNmea(), NMEA Sentence

## Synopsis

**SrNmea**(F, IDSTR, INDEX, COUNT, MODE)
**SrNmea**(F, IDSTR, INDEX, COUNT, MODE, HEX)

| | |
|---|---|
| **F**[*m*] | Formula string to tag for NMEA data (*m*≥1). |
| **IDSTR**[*p*] | String used for sentence ID (*p*≥1) (string). |
| **INDEX**[1] | Index into serial data (integer). |
| **COUNT**[1] | Number of data entries to get (integer). |
| **MODE**[1] | Mode selector (integer). |
| **HEX**[1] | Hexadecimal number selector (integer). |

## Description

This function retrieves data from a GPS NMEA Sentence. The **IDSTR** is used to pick the data from the correct NMEA sentence. This should match the sentence ID from the NMEA format.

The **INDEX** is used to select the desired field from the data. For the first field use 1, for the second field use 2 and so on.

The **COUNT** indicates the number of data entries to get. This allows us to get multiple data entries at a time making it more efficient and using less formulas when possible.

The **MODE** allows for different types of data. Please see the table bellow.

For hexadecimal numbers, the **HEX** selector can be used to retrieve a hexadecimal value as opposed to decimal.

Having the correct trigger is critical to getting the correct data. This means use the sentence ID to trigger on the exact NMEA sentence.

For an alternate function for getting NMEA data, *See "Nmea(), NMEA Sentence"*

| MODE | Data Type | Bytes |
|---|---|---|
| -1 | String | |
| 0 | Time | |
| 1 | Date (reserved) | |
| 2 | Char | 1 |
| 3 | Unsigned Char | 1 |
| 4 | Integer | 2 |
| 5 | Unsigned Integer | 2 |

**MODE**

| MODE | Data Type | Bytes |
|---|---|---|
| 6 | Long | 4 |
| 7 | Unsigned Long | 4 |
| 8 | Float | 4 |
| 9 | Double Float | 8 |

**MODE**

*Note: Data for this function must be in serial NMEA format. You cannot use this function to unpack data for the SEA GPS Interface. You must use the traditional way for SEA GPS interface data. The data can come from ANY available serial port in the system.*

## Result Type/Space

D[n], F[n], L[n], l[n], I[n], i[n], C[n], c[n], S[n].

## Example

```
; Name        Units    Number   Result Computations
"PIXSE SPEED_  " ""    F3150     D[3] SrNmea(F3000, "$PIXSE", 1, 3, 9)
"East"         "m/s"   F3160     D[1] DIndex(F3150, 0)
"North"        "m/s"   F3161     D[1] DIndex(F3150, 1)
"Vertical"     "m/s"   F3162     D[1] DIndex(F3150, 2)
```

# SrVAX(), Serial VAX

## Synopsis

**SrVAX**(A, INDEX, COUNT)

| | |
|---|---|
| **A** | Acquisition tag for Serial VAX data (tag). |
| **INDEX**[1] | Index of value in serial data (integer). |
| **COUNT**[1] | Number of data values for this index (integer). |

## Description

This function is used to retrieve specific IEEE data values from a block of VAX float data. The data stored is unchanged, however, the data displayed is swapped and the exponent is decremented by two in order to obtain the desired value.

## Result Type/Space

D[*n*], *n* = **COUNT**

## Example

```
; Name      Units  Number  Result   Computations
"SrVAX"      ""     F100    F[5]    SrVax(A100, 10, 5)
```

# StDev(), Standard Deviation

## Synopsis

**StDev**(X, STATE)
**X**[n] X values.
**STATE**[1] Function control variable (integer).

## Description

This function returns the standard deviation through the given data points.

The **STATE** control variable is used to control the function operation mode. If the **STATE** control variable is a '0', then the function performs a reset and computes a new value. If the **STATE** control variable is a '1', then new data is processed and a standard deviation value returned. If the **STATE** control variable is a '2', this cause the last standard deviation value to be held.

## Result Type/Space

D[*1*]

## Example

```
; Name          Units  Number  Result   Computations
"StDev"           ""     F100    F[1]   StDev(F2000, F200)
```

# STemp(), Static Temperature

## Synopsis

**STemp**(TTEMP, PPRES, SPRES, RECOVERY)

| | |
|---|---|
| **TTEMP**[*m*] | Total Temperatures (in C) (*m*≥1). |
| **PPRES**[*p*] | Pitot Pressures (in mb) (*p*≥1). |
| **SPRES**[*r*] | Static Pressures (in mb) (*r*≥1). |
| **RECOVERY**[1] | Recovery factors. |

## Description

This function computes static temperature from total temperature, pitot pressure, static pressure, and an installation specific recovery constant. The recovery constant varies between 0.0 and 1.0. This function uses interpolation [**See Interpolation**].

$$f[i] = \frac{TTEMP[i] + 273.15}{1 + RECOVERY \times \left(\left(1 + \frac{PPRES[i]}{SPRES[i]}\right)^{0.285867} - 1\right)} - 273.15$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m, p, r*)

## Example

```
; Name                Units    Number  Result  Computations
"StaticTemperature"   "°c"    F200    F[20]    STemp(F100, F101, F102, 1.0)
```

# Stormscope(), Stormscope

## Synopsis

**Stormscope**(LAT, LON, COUNT, STATE)

| | |
|---|---|
| **LAT**[*m*] | Latitude (*m*≥1). |
| **LON**[*p*] | Longitude (*p*≥1). |
| **COUNT**[1] | Strike count. |
| **STATE**[1] | State. |

## Description

This function can be used to combine Stormscope strike latitude and longitude values into a single array of latitude/longitude pairs. The count specifies the number of Stormscope strikes to join together.

The state can be used to control the function execution such as that a value of 0 doesn't return nor update the result value. If the state is a 1 the function returns the latitude and longitude pairs each second. If the state transitions between 1 to 0 it resets the internal function variables.

## Result Type/Space

D[*n*], F[n], This function modifies the formula valid count field which can be used for ASCII output.

## Example

```
; Name    Units Number Result Computations
"SLadLonD"   "" F44090  F[63] Stormscope(F1510, F1520, F1010, F44080)
```

# StrCat(), String Concatenate

## Synopsis

**StrCat**(STRING1, STRING2)
**StrCat**(STRING1, STRING2, LENGTH)

| | |
|---|---|
| **STRING1**[*m*] | String for compare (*m*≥1) (string). |
| **STRING2**[*p*] | String for compare (*p*≥1) (string). |
| **LENGTH**[1] | Number of characters to be used from string2 (integer). |

## Description

This function concatenates two string together. The **STRING1** is copied first and followed by **STRING2**. The **LENGTH** parameter can be used to select the number of characters from **STRING2** to use a limit.

## Result Type/Space

S[n]

## Example

```
; Name             Units  Number  Result   Computations
"StringConcatenate"   ""   F1000  S[32]  StrCat(F1810, F1820)
```

# StrCmp(), String Compare

## Synopsis

**StrCmp**(STRING1, STRING2)
**StrCmp**(STRING1, STRING2, LENGTH)

| | |
|---|---|
| **STRING1**[*m*] | String for compare (*m*≥1) (string). |
| **STRING2**[*p*] | String for compare (*p*≥1) (string). |
| **LENGTH**[1] | Number of characters to be compared (integer). |

## Description

This function compares the two strings passed in the parameters and returns zero if the strings are not equal. Otherwise, if the strings are the same, this function return one. The number of characters compared is determined by the number of characters in the second string parameter. If the **LENGTH** argument is given, this will determine the number of character to be compared. For example, if a function compares "ABCD" and "ABC", without the **LENGTH** argument, it will return false. However, if a value of 3 is given for **LENGTH**, the function will return true. The compare is case sensitive.

## Result Type/Space

I[1]

## Example

```
; Name             Units  Number  Result   Computations
"StringCompare"     ""      F100    I[1]    StrCmp(A65532, "fml 2499 1")
```

# StrCpy(), String Copy

## Synopsis

**StrCpy**(STRING)
**StrCpy**(STRING, LENGTH)

| | |
|---|---|
| **STRING**[*m*] | String for compare (*m*≥1) (string). |
| **LENGTH**[1] | Number of characters to be copied (integer). |

## Description

This function copy the string in the parameter provided to the formula's result space. This in fact copies the contents on one formula to another.

The **LENGTH** parameter can be used to limit the number of characters copied.

## Result Type/Space

S[n]

## Example

```
; Name            Units  Number  Result   Computations
"StringCopy"      ""     F1000    S[32]   StrCpy(F2000)
```

# StrParameters(), String Parameter Count

## Synopsis

**StrParameters**(STRING, DELIMETER)

| | |
|---|---|
| **STRING**[*n*] | String for compare (*n*≥1) (string). |
| **DELIMETER**[1] | Delimiter character between parameters (integer). |

## Description

This function returns the number of parameters from a string. The delimiter identifies the separating ASCII character between parameters.

## Result Type/Space

D[1], F[1], L[1], l[1], I[1], i[1]

## Example

```
; Name             Units  Number  Result   Computations
"StringParameters"   ""  F1000    I[1]   StrParameters(F2000, 47)
```

# StrPrt(), String Print

## Synopsis

**StrPrt**(FORMAT, VALUE)

| | |
|---|---|
| **FORMAT**[*m*] | String for format (*m*≥1) (string). |
| **VALUE**[1] | Value to print (double, float, long, integer). |

## Description

This function is used to do a basic print of a **VALUE** to a string (result space for the formula).

The **FORMAT** parameter is a string that will need to follow the guidelines for the c programing language printf function.

The **VALUE** cannot be a string type.

## Result Type/Space

S[n]

## Example

```
; Name            Units  Number  Result   Computations
"StringPrint"     ""     F1000   S[32]    StrPrt("A%05.0f", F2000)
```

# StrSel(), String Select

## Synopsis

**StrSel**(VALUE, SELECT STRING)

| | |
|---|---|
| **VALUE**[1] | Integer value for comparison. |
| **SELECT**[1] | Integer select value for comparison. |
| **STRING**[n] | String to copy to result space ($n \geq 1$) (string). |

## Description

This function compares the integer values (value and select). If the comparison is true then the string provided is copied to the result space. Otherwise nothing is done.

## Result Type/Space

S[n], *n* = length of the parsed string

## Example

```
; Name        Units  Number  Result Computations
"DLW"           ""   F9103     S[6] StrSel(F9102, 1, "DLH")
"OZR"           ""   F9103     S[6] StrSel(F9102, 2, "OZR")
"PXT"           ""   F9103     S[6] StrSel(F9102, 4, "PXT")
"SAE"           ""   F9103     S[6] StrSel(F9102, 8, "SAW")
```

# StrToD(), String to Double

## Synopsis

**StrToD**(STRING)
**StrToD**(STRING, OFFSET)
**STRING**[*m*] String to be converted (*m*≥1) (string).
**OFFSET**[1] Byte offset into string to start conversion (integer).

## Description

This function takes a string of characters given by **STRING** and converts the string into it's double representation. The function recognizes **STRING** containing an optional white space, followed by an optional sign, a sequence of digits containing an optional decimal point, and an optional 'e' or 'E' (exponent) followed by a sequence of digits. For example, the user may need to perform a computation on a value that is currently in a string form. Prior to this computation, the string must be converted into a real numerical data form for the CPU. **OFFSET** is used to "skip" a certain number of characters in the string. If **OFFSET** is not specified, it is assumed to be zero.

The function returns the converted value, or zero if the value cannot be converted.

## Result Type/Space

D[1]

## Example

```
; Name            Units  Number  Result  Computations
"StringtoDouble"  ""     F100    D[1]    StrToD(F105, 4)
```

# StrTok(), Parse String Token

## Synopsis

**StrTok**(STRING, TOKENS)
**STRING**[1] String for parsing (string).
**TOKENS**[1] Token List (string).

## Description

This function compares given **TOKENS** to values within the **STRING** and allows the user to break up the data into a separate strings that can go into different formulas. The first time the function is called you must pass the formula number of the string data. After this you must call this function with the "NULL" parameter to continue parsing the string. This is to use the data in memory and try to parse out the desired tokens to return the portion of the string before the token. It is important to note that only one string at a time can be parsed using this function.

## Result Type/Space

S[*n*], *n* = length of the parsed string

## Example

```
; Name      Units  Number  Result  Computations
"String"     ""     F300   S[300]  A1200
"Str"        ""     F301   S[300]  StrTok(F300, "\n\r")
"Str"        ""     F302   S[300]  StrTok("NULL", "\n\r")
"Str"        ""     F303   S[300]  StrTok("NULL", "\n\r")
"Str"        ""     F304   S[300]  StrTok("NULL", "\n\r")
"Str"        ""     F305   S[300]  StrTok("NULL", "\n\r")
```

# StrToL(), String to Long Integer

## Synopsis

**StrToL**(STRING)
**StrToL**(STRING, OFFSET)
**StrToL**(STRING, OFFSET, BASE)
**StrToL**(STRING, OFFSET, BASE, STRINGLEN, STRINGCOUNT)

| | |
|---|---|
| **STRING**[*m*] | String to be converted (*m*≥1) (string). |
| **OFFSET**[1] | Byte offset into string to start conversion (integer). |
| **BASE**[1] | Base to convert value to (integer: 0, 2-36) |
| **STRINGLEN**[1] | Number of characters to use (integer; 0 for auto-length) |
| **STRCOUNT**[1] | Number of samples (integer) |

## Description

This function takes a string of characters given by **STRING** and converts the string into it's long integer representation. The function recognizes **STRING** containing an optional white space, followed by an optional sign, and a sequence of digits and letters (alphanumeric). For example, the user may need to perform a computation on a value that is currently in a string form. Prior to this computation, the string must be converted into a real numerical data form for the CPU.

**OFFSET** is used to "skip" a certain number of characters in the string. If **OFFSET** is not specified, it is assumed to be zero.

If **BASE** is zero, the first alphanumeric characters encountered in the string determine it's base. If the first characters are '0x' or '0X', the digits are treated as hexadecimal. If the first character is 'o' or 'O', the digits are treated as octal, otherwise the digits will be treated as decimal (default). If **BASE** is non-zero, it must be in the range [2, 36]. The letters 'a-z' and 'A-Z' represent the values 10-35. Only those letters whose designated values are less than **BASE** are permitted. If the value of **BASE** is 16, the characters '0x' or '0X' may optionally precede the sequence of letters and digits. If **BASE** is not specified, it is assumed to be 10 (decimal).

The function returns the converted value. If the value exceeds the usable range, the maximum or minimum range value (depending on the sign in the string) is returned. If **BASE** is out or range, zero is returned.

## Result Type/Space

L[1]

## Example

```
; Name            Units  Number  Result  Computations
"StringtoLong"     ""     F100    L[1]   StrToL(F105, 4, 16)
```

# StrToUL(), String to Unsigned Long Integer

## Synopsis

**StrToUL**(STRING)
**StrToUL**(STRING, OFFSET)
**StrToUL**(STRING, OFFSET, BASE)
**StrToUL**(STRING, OFFSET, BASE, STRINGLEN, STRINGCOUNT)

| | |
|---|---|
| **STRING**[*m*] | String to be converted (*m*≥1) (string). |
| **OFFSET**[1] | Byte offset into string to start conversion (integer). |
| **BASE**[1] | Base to convert value to (integer: 0, 2-36) |
| **StrToUL**[1] | Number of characters to use (integer; 0 for auto-length) |
| **StrToUL**[1] | Number of samples (integer) |

## Description

This function takes a string of characters given by **STRING** and converts the string into it's unsigned long integer representation. The function recognizes **STRING** containing an optional white space, followed by an optional sign, and a sequence of digits and letters (alphanumeric). For example, the user may need to perform a computation on a value that is currently in a string form. Prior to this computation, the string must be converted into a real numerical data form for the CPU.

**OFFSET** is used to "skip" a certain number of characters in the string. If **OFFSET** is not specified, it is assumed to be zero.

If **BASE** is zero, the first alphanumeric characters encountered in the string determine it's base. If the first characters are '0x' or '0X', the digits are treated as hexadecimal. If the first character is 'o' or 'O', the digits are treated as octal, otherwise the digits will be treated as decimal (default). If **BASE** is non-zero, it must be in the range [2, 36]. The letters 'a-z' and 'A-Z' represent the values 10-35. Only those letters whose designated values are less than **BASE** are permitted. If the value of **BASE** is 16, the characters '0x' or '0X' may optionally precede the sequence of letters and digits. If **BASE** is not specified, it is assumed to be 10 (decimal).

The function returns the converted value. If the value exceeds the usable range, the maximum or minimum range value (depending on the sign in the string) is returned. If **BASE** is out or range, zero is returned.

## Result Type/Space

L[1]

## Example

```
; Name             Units  Number  Result  Computations
"StringToULong"     ""     F100    L[1]   StrToUL(F105, 0, 10)
```

# StrXmlProtect(), String XML Protect

## Synopsis

**StrXmlProtect**(STRING)
**STRING**[*n*] String for compare (*n*≥1) (string).

## Description

This function replaces all the '<', '>' and '/' characters from XML type data with spaces.

This simplifies, clarifies and provides an alternate way to handle the XML type data.

## Result Type/Space

S[n]

## Example

```
; Name          Units  Number  Result   Computations
"StrXmlProt"     ""     F1000  S[1024]  StrXmlProtect(F2000)
```

# Sub(), Subtract Arrays

## Synopsis

**Sub**(A, B)

**A**[*m*] Formula of an array of values (*m*≥1).

**B**[*p*] Formula of an array of values (*p*≥1).

*Note: Deprecated [M300 Replacement function - See "-, Sub"]*

## Description

This function returns an array of values representing the subtraction of the two given arrays element by element. This function uses interpolation [**See Interpolation**]**.** The following formula summarizes the computations.

$$f[i] = A[i] + B[i]$$
$$\text{for i} = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*, *p*)

## Example

```
; Name        Units  Number  Result   Computations
"Subtract"     ""     F300    F[15]   Sub(F100, F200)
```

# Sum(), Summation

## Synopsis

**Sum**(F)

**F**[*n*] Formula of an array of values ($n \geq 1$).

## Description

Computes the sum of all values in an array.

$$f = \sum_{i=0}^{n-1} F[i]$$

## Result Type/Space

D[1]

## Example

```
; Name    Units  Number  Result   Computations
"Sum"      ""     F200    F[1]    Sum(F100)
```

# Sums1D(), Sums 1D

## Synopsis

**Sums1D**(A, INTERVAL, STATE, FIRST)

| | |
|---|---|
| **A** | 1D acquisition tag number (tag). |
| **INTERVAL**[1] | Interval of summation (in display cycles) (integer). |
| **STATE**[1] | Function control variable (integer). |
| **FIRST**[1] | Skip channel zero size (integer). |

*Note: Deprecated [M300 Replacement function - See "OdSums(), 1D Sums"]*

## Description

This function sums up all 1D samples (i.e. FSSP, ASASP) from a data buffer. This summation is accrued for an interval of time as specified by the second parameter (in display cycles). At the end of the time interval the sums are returned through the result space and the internal summation values are cleared for the next summation period.

The **STATE** control variable is used to control the function operation mode. If the **STATE** control variable is a '0', then the summation is done every interval. If the **STATE** control variable is a '1', then the sums are accumulated. If the **STATE** control variable is a '2', this causes the last summation value to be held. Any other transition in the control variable, clears the internal summation and starts the accumulation process all over again.

The **FIRST** parameter can be used to avoid returning the summation value for channel zero (this may be desired with the advanced 1D256 interface). Use a value of '0' to return all counts. Use a value of '1' to skip the counts for channel zero. If the acquisition type is a CAS probe then this parameter is used to select which channels will be used. If the **FIRST** control variable is a '0' then the forward channel counts will be added, if the variable is a '1' then the backward channel counts will be added, and if the variable is a '2' then the inter arrival counts are added.

## Result Type/Space

D[*n*], *n* = number of samples

## Example

```
; Name       Units  Number  Result   Computations
"Counts"      ""     F100    F[15]   Sums1D(A100, 1, 0, 0)
```

# Sums2D(), 2D Sums

## Synopsis

**Sums2D**(2D, ELAPSED, MODE, PROBE, INTERVAL)

| | |
|---|---|
| **2D** | 2D image tag (tag). |
| **ELAPSED** | 2D elapsed time tag (tag). |
| **MODE**[1] | Mode value (integer). |
| **PROBE** | Probe name/number (probe). |
| **INTERVAL**[1] | Integration interval (integer). |

*Note: Deprecated [M300 Replacement function - See "MoSums(), 2D Mono Sums"]*

## Description

This function builds up an approximation of the 2D spectrum using the image data and the time slice mask. These images are summed up using either the slice count or the slice width and normalized using the elapsed time value. The output of the function is a sums array and may be processed like the sums array from 1D data.

Starting with version 3.01 dated (8/21/98), the SUMS2D function no longer uses the mode as the end of a particle. The end of a particle is now detected by finding one or more blank slices (all ones, 0xFFFFFFFF). In addition, the upper byte of the **MODE** parameter (16 bit integer) can be used to control the following options.

| MODE | Description |
|---|---|
| xxxxxxx1xxxxxxxx | Add zero area particles to first bin |
| xxxxxxx0xxxxxxxx | Don't add zero area particles to first bin |
| xxxxxx1xxxxxxxxx | Use x-dimension method (across slice) |
| xxxxxx0xxxxxxxxx | Use y-dimension method (TAS dependent) |

**Table 12: MODE**

## Result Type/Space

D[*n*], *n* = number of channels in probe entry (**PROBE**)

## Example

```
; Name     Units    Number   Result    Computations
"2DSums"    ""       F100     F[64]    Sums2D(A100, A101, 0x55, P2, 1)
```

# Sums2G(), 2D Grey Sums

## Synopsis

**Sums2G**(ORTAG, SLICETAG, ELAPSEDTAG, MODE, PROBE, INERVAL, STATE)
**Sums2G**(ORTAG, SLICETAG, ELAPSEDTAG, MODE, PROBE, INERVAL, STATE, YSIZE)
**Sums2G**(ORTAG, MINTAG, MIDTAG, MAXTAG, ELAPSEDTAG, MODE, PROBE, INTERVAL, STATE, YSIZE)

| | |
|---|---|
| **ORTAG** | Acquisition tag for 2D Grey OR slice data (tag). |
| **SLICETAG** | Acquisition tag for 2D Grey slice count data (tag). |
| **ELAPSEDTAG** | Acquisition tag for 2D Grey elapsed time data (tag). |
| **MINTAG** | Acquisition tag for 2D Grey minimum shadow data (tag). |
| **MIDTAG** | Acquisition tag for 2D Grey middle shadow data (tag). |
| **MAXTAG** | Acquisition tag for 2D Grey maximum shadow data (tag). |
| **MODE**[1] | Sizing mode value (integer). |
| **PROBE** | Probe name/number (probe). |
| **INTERVAL**[1] | Integration interval (integer). |
| **STATE**[1] | Function control variable (integer). |
| **YSIZE**[1] | Pixel dimension. |


*Note: Deprecated [M300 function replacement - See "GrSums(), 2D Grey Sums"]*

## Description

This function builds up an approximation of the 2D Grey spectrum using the 'X' and 'Y' dimensions and elapsed time of the 2D Grey scaled images. These images are summed up and normalized using the elapsed time value. The output of the function is a sums array and may be processed like the sums array from 1D and 2D data.

| MODE (low nibble) | Description |
|---|---|
| 0 | (X + Y) / 2 |
| 1 | X (TAS independent) |
| 2 | Y (TAS dependant) |
| 3 | Area (no edge reject) |
| 4 | Area (use edge reject) |
| 5 | X (particle reject) |

**Table 13: MODE**

| MODE (low nibble) | Description |
|---|---|
| 6 | Y (particle reject) |
| 7 | (X + Y) / 2 [Use edge reject] |

**Table 13: MODE (Continued)**

The user may select different sizing modes for the function, by providing different values for the **MODE** parameter. The following table describes possible sizing modes.

If the **MODE** is '0', the average of the 'X' and 'Y' dimensions are used for sizing. If the **MODE** is a '1', only the 'X' dimension is used for sizing. If the **MODE** is '2', the 'Y' dimension is used for sizing. If the **MODE** is a '3' or '4', the particle is assumed round and the particle dimension is obtained by finding the diameter from the total particle area. The difference between **MODE** '3' and '4' is that **MODE** '3' counts all particle (no edge reject) and **MODE** '4' only counts the particles that do not touch the edges (edge reject).

For the modes that compute particle diameter from total particle area (modes '3' and '4'), the upper nibble for the **MODE** parameter is used to control which shadow levels are added (minimum, middle, maximum). A shadow level is added to the total area calculation by setting the corresponding bit. Bit 4 is used for minimum shadow, bit 5 for middle shadow and bit 6 for maximum shadow. Normally by default all shadow levels are used to compute the total particle area (upper nibble for **MODE** parameter is zero).

The **STATE** control variable is also used to control the function operational mode. If the **STATE** control variable is a '0', then the summation is done every interval. If the **STATE** control variable is a '1', then the sums are accumulated. If the **STATE** control variable is a '2', this causes the last summation value to be held. Any other transition in the control variable, clears the internal summation and starts the accumulation process all over again.

Note that this function can have three different calling formats.

The first, is fully compatible with earlier versions, and it does the regular spectrum computations. This calling format only supports modes 0, 1 and 2.

The second, has one extra parameter, and it uses the same computation with the exception that it does not assume square sizing pixels. This later method can be useful if the TAS is greater than the maximum TAS the probe can sample at. In this case, the last parameter is used to specify the 'Y' size of the pixel. This size should be equal to the TAS divided by the probe clock frequency ('TAS/FREQ' where 'FREQ=(MULTFAC*50000)/DIVFAC'). This calling format only supports modes 0, 1 and 2.

The third, can be used to compute particle size from the total area provided by the minimum, middle and maximum counts. This method assumes that the particles are round and it works out the particle diameter from the total area of a circle (round particle). This calling format only supports modes 3 and 4.

Since this function has different calling formats, there is no syntax checking on the number of parameters passed to the function. Care should be taken to use the appropriate number of parameters, or the function will not return a spectrum array and it may affect system performance.

## Result Type/Space

D[*n*]

## Example

```
; Name     Units  Number  Result   Computations
"2GSums"    ""     F100    F[64]   SUMS2G(A100, A101, A101, 0, P3, 1, 1)
```

# Sums2GAdv(), 2D Grey Advanced Sums

## Synopsis

**Sums2GAdv**(A, MODE, PROBE, INTERVAL, STATE, YSIZE)

| | |
|---|---|
| **A** | Acquisition tag for 2D Grey data (tag). |
| **MODE**[1] | Function sizing mode (integer). |
| **PROBE** | Probe definition (probe). |
| **INTERVAL**[1] | Integration interval (integer). |
| **STATE**[1] | Function control variable (integer). |
| **YSIZE**[1] | Pixel dimension (float). |

*Note: Deprecated [M300 Function replacement - See "GrSums(), 2D Grey Sums"]*

## Description

This function builds up an approximation of the 2D Grey advanced spectrum using the particle dimensions and elapsed time of the 2D Grey scaled images. These particles can be summed up (using several different modes) and normalized using the elapsed time value (if desired). The output of the function is an array and may be processed like the arrays from 1D and 2D data.

The **STATE** control variable is also used to control the function operational mode. If the **STATE** control variable is a '0', then the summation is done every interval. If the **STATE** control variable is a '1', then the sums are accumulated. If the **STATE** control variable is a '2', this causes the last summation value to be held. Any other transition in the control variable, clears the internal summation and starts the accumulation process all over again.

The 'PIXELSIZE' should be equal to the probe pixel size for TAS less than the maximum TAS the probe can sample at. For higher TAS, the 'PIXELSIZE' should be equal to the 'TAS/FREQ' where 'FREQ=(MULTFAC*50000)/DIVFAC'.

Using the **MODE** parameter, it is possible to control the sizing method. The lower nibble for the **MODE** parameter controls the sizing method while the upper nibble for the **MODE** parameter has some additional control bits (shadow levels and uncorrected counts). The **MODE** parameter is best specified in hexadecimal notation. In order to come up with the correct value for the **MODE** parameter it is necessary to first find the desired sizing method and then use the decimal and binary values from the tables in the next page to come up with the final value (in hexadecimal) to pass to the summation function. For example, to specify the area (with edge reject) sizing method using only the middle and maximum shadows. You would pick 4 for the lower nibble from the first table. Then pick 0110 (binary) for the upper nibble (maximum and middle shadows). The hexadecimal value for 0110 (binary) is 6. Therefore the desired value for the **MODE** parameter is 0x64.

The following table shows the valid 'MODE' values for the lower nibble (decimal values) and a description of what they do to compute the particle size.

| MODE (low nibble) | Description |
|---|---|
| 0 | (X+Y) / 2 |
| 1 | X (TAS independent) |
| 2 | Y (TAS dependent) |
| 3 | Area |
| 4 | Area (reject particles that touch edge) |
| 5 | X (reject particles that touch edge) |
| 6 | Y (reject particles that touch edge) |
| 7 | (X+Y) / 2 (reject particles that touch edge) |

**Table 14: MODE (lower nibble)**

The following table shows the valid **MODE** bits (bits 7, 6, 5 and 4 of the **MODE**) for the upper nibble (binary values, x means don't care) and a description of what they do to control the sizing methods.

| MODE (upper nibble) | Description |
|---|---|
| xxx1 | Minimum shadow bit selector |
| xx1x | Middle shadow bit selector |
| x1xx | Maximum shadow bit selector |
| 1xxx | Raw uncorrected counts bit selector |

**Table 15: MODE Sizing bits**

As you can see, bit 7 of the 'MODE' parameter can be set to return raw uncorrected counts (or in 0x80 hexadecimal), for all sizing methods (no normalization using elapsed time).

For both Area sizing methods (lower nibble 3 and 4 for the **MODE**), you can use the upper nibble for the **MODE** parameter to control which shadow levels are added (minimum, middle, maximum). A shadow level is added to the total area calculation by setting the corresponding bit. Bit 4 is used for minimum shadow, bit 5 for middle shadow and bit 6 for maximum shadow.

If the minimum, middle and maximum bits of the **MODE** parameter are all zero, this indicates an invalid mode and all bits are assumed on (default mode).

A particle is found to touch the edge by having either the first or the last pixel set in any color (minimum, middle or maximum). This test is done by this function for all particle slices. For the edge

reject modes, the elapsed times for all particles (rejected or not) are counted up and used in the final correction.

The X size of a particle is computed by adding all the set bits in a particular slice. From slice to slice, in a given particle, the X size only changes, if it was greater than the largest X size found so far.

The X size computation may be modified by the minimum, middle and maximum bits of the **MODE** parameter. If the minimum bit is set, all three shadow levels are counted (minimum, middle and maximum). If the middle bit is set, two shadow levels are counted (middle and maximum). Similarly, if the maximum bits is set, only the maximum shadow is counted. For the X size method, these are the only three valid modifiers. One of these methods will be picked depending on which bits are set (minimum is checked first, then middle and finally maximum).

After April 12 2000, there is new method for correcting the total particle counts. Please note that more work space is needed.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name      Units  Number  Result  Computations
2DSums      ""     F100    F[64]   Sums2dgadv(A100, 0x02, P3, 1, 1, 25)
```

# SumsHVPS(), High Volume Precipitation Spectrometer Sums

## Synopsis

**SumsHVPS**(A, PROBE, INTERVAL)

| | |
|---|---|
| **A** | Acquisition tag for HVPS data tag (tag). |
| **PROBE** | Probe number/name (probe). |
| **INTERVAL**[1] | Integration interval (integer). |

***Note: Deprecated [M300 Function replacement - See "HvSums(), High Volume Precipitation Spectrometer Sums"]***

## Description

This function builds up an approximation of the HVPS spectrum using the image data and the time information.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name        Units   Number   Result    Computations
"HVPSSums"     ""      F100     F[64]    SumsHVPS(A100, P3, 1)
```

# System(), System Data Access

## Synopsis

**System**(SELECT)
**SELECT**[1] Selector for desired data (integer: 0).

## Description

This function allows access to individual items of the M300 system. The following table shows the different **SELECT** values for the different M300 data fields.

| Data Field | SELECT | Result |
|---|---|---|
| M300 Operational Mode | 0 | I[1] |

**M300 System Select**

| Description | Value |
|---|---|
| NONE | 0 |
| ACQUISTION | 1 |
| PLAYBACK | 2 |
| UDP | 3 |

**M300 Operational Mode**

## Result Type/Space

See M300 System Select table above.

## Example

```
; Name        Units      Number  Result  Computations
"M300Mode"      ""        F1020    I[1]  System(0)
```

# TamdarData(), Tamdar Data Access

## Synopsis

**TamdarData**(A, SELECT)

| | |
|---|---|
| **A** | Acquisition tag for Tamdar data (tag). |
| **SELECT**[1] | Selector for desired data (integer: 0..19). |

## Description

This function allows access to individual items of the Tamdar data block. The following table shows the different **SELECT** values for the different Tamdar data fields. The function will return the value of a user specified item from a Tamdar buffer. Please check the Tamdar manual for further information.

| Data Field | SELECT | Result |
|---|---|---|
| Probe Serial Number | 0 | L[1] |
| Day of Month | 1 | L[1] |
| Latitude (string) | 2 | S[12] |
| Latitude (rad) | 3 | F[1] |
| Longitude (string) | 4 | S[12] |
| Longitude (rad) | 5 | F[1] |
| GPS UTC Time | 6 | S[12] |
| Pressure Altitude (ft) | 7 | L[1] |
| Ambient Temperature (celcius) | 8 | F[1] |
| Wind Direction (deg) | 9 | F[1] |
| Wind Magnitude (knots) | 10 | L[1] |
| Wind Flag | 11 | L[1] |
| RH1 | 12 | F[1] |
| RH1 Flag | 13 | L[1] |
| Eddy Dissipation Rate | 14 | L[1] |
| Eddy Time | 15 | L[1] |

**Tamdar Data SELECT**

| Data Field | SELECT | Result |
|---|---|---|
| Indicated Airspeed (knots) | 16 | L[1] |
| GPS Altitude (ft) | 17 | L[1] |
| Icing/Heater Flag | 18 | L[1] |
| RH2 | 19 | F[1] |

**Tamdar Data SELECT**

## Result Type/Space

See Tamdar Data SELECT table above.

## Example

```
; Name       Units     Number  Result  Computations
"Latitude"   "rad"     F1020    F[1]   TamdarData(A1000, 3)
```

# TasP(), Pitot Press from TAS

## Synopsis

**TASP**(STEMP, TAS, SPRES)

| | |
|---|---|
| **STEMP**[*m*] | Static temperature (in c) (*m*≥1). |
| **TAS**[*p*] | TAS (in m/s) (*p*≥1). |
| **SPRES**[*r*] | Static pressure (in mb) (*r* ≥1). |

## Description

This function computes pitot pressure (in mb) from static temperature, tas, and static pressure. Function uses interpolation [**See Interpolation**]. The following formula summarizes the computation.

$$f[i] = SPRESS \times \left( \left( \left( \frac{TAS^2}{2009.6 \times (STEMP + 273.15)} \right) + 1 \right)^{3.498130249} - 1 \right)$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m*,*p*,*r*)

## Example

```
; Name           Units   Number  Result  Computations
"PitotPress"    "mbar"   F200    F[1]    TasP(F100, F101, F102)
```

# Test(), Test

## Synopsis

**Test**($P_1$, $P_2$, ... , $P_n$)

| | |
|---|---|
| $P_1$ | First parameter. |
| $P_2$ | Second parameter. |
| $P_n$ | *n*th Parameter. |

## Description

This function is used to create an array of test values. This array may be used as an input to other functions or displays. The resulting array is formed by the parameters passed in the function.

## Result Type/Space

D[*n*]

## Example

```
; Name   Units  Number  Result   Computations
"Test"    ""     F100    F[7]    Test(0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0)
```

# Time(), Time

## Synopsis

**Time**(A)

**A** Date time acquisition tag number (tag).

## Description

This function converts the date/time acquisition data into an ASCII string for display purposes. Only the time part is returned. The resulting string has the format **HH:MM:SS.FFF** where **HH** stands for hours, **MM** stands for minutes, **SS** stands for seconds and **FFF** is for milliseconds. The number of elements in the result formula space controls how much of the time string actually gets returned. If the formula result space is three, the time string will look like **HH**. If the result space is six, the time string will look like **HH:MM**. If the result space is 13, the time string will be **HH:MM:SS.FFF**. For the formula result space, you must add 1 extra for the terminating character.

The tag number can be "A0" for M300 time or a GPS tag number for time. These are the only two supported tags.

## Result Type/Space

S[*n*], *n* = size of result space entry

## Example

```
; Name      Units  Number  Result  Computations
"Time"      ""     F0      S[6]    Time(A0)
```

# Timer(), Timer

## Synopsis

**Timer**(DELAY, ON, OFF)

| | |
|---|---|
| **DELAY**[1] | Initial delay for dead period (integer). |
| **ON**[1] | Time for which the function will be on (integer). |
| **OFF**[1] | Time for which the function will be off (integer). |

## Description

This function is used to generate a pulse train for instrument control. After an initial dead period (**DELAY**), the function will return a one for the duration of **ON**, and a zero for the duration of **OFF**. The cycle will then repeat using the **ON** and **OFF** values.

## Result Type/Space

I[1]

## Example

```
; Name       Units  Number  Result  Computations
"Timer"      ""     F100    I[1]    Timer(10, 5, 25)
```

# TTemp(), Total Air Temperature

## Synopsis

**TTemp**(STEMP, PPRES, SPRES, RECOVERY)

| | |
|---|---|
| **STEMP**[*m*] | Static Temperatures (in C) (*m*≥1). |
| **PPRES**[*p*] | Pitot Pressures (in mb) (*p*≥1). |
| **SPRES**[*r*] | Static Pressures (in mb) (*r*≥1). |
| **RECOVERY**[1] | Recovery factors. |

## Description

This function computes total air temperature from static temperature, pitot pressure, static pressure, and an installation specific recovery constant. The recovery constant varies between 0.0 and 1.0. This function uses interpolation [**See Interpolation**].

$$f[i] = (STEMP + 273.15) \times \left(1 + RECOVERY \times \left(\left(1 + \frac{PPRES}{SPRES}\right)^{0.285867} - 1\right)\right) - 273.15$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], *n* = max(*m, p, r*)

## Example

```
; Name                Units    Number  Result  Computations
"TAT"          "°C"    F200    F[20]   TTemp(F100, F101, F102, 1.0)
```

# Unfold(), Unfolding (Doppler)

## Synopsis

**Unfold**(V1, V2)

| | |
|---|---|
| **V1**[*m*] | Formula for array of values of 1st Velocity ($m \geq 1$) |
| **V2**[*p*] | Formula for array of values of 2nd Velocity ($p \geq 1$) |

## Description

Performs the Standard Doppler Unfolding function using the ration of 5/4.

## Result Type/Space

D[*n*], $n = \min(m, p)$

## Example

```
; Name      Units  Number  Result   Computations
"Unfold"     ""     F300    D[10]   Unfold(F1002, F1003)
```

# Units(), Unit Conversion

## Synopsis

**Units**(F, TO, FROM)

| | |
|---|---|
| **F**[*n*] | Formula of an array of values to be converted (*n*≥1) |
| **TO** | Units '**F**' is to be converted to (see below) (string) |
| **FROM** | Units '**F**' was originally described with (see below) (string) |

## Description

This function takes in the '**F**' value as the magnitude. Coupled with the **FROM** field, the returned value will be **F***(**TO** multiplier) units. For example to go from 12 inches to 1 foot, you would call the function as such:

- *Units(12, "ft", "in")*

The following table shows which units may be converted **TO** and **FROM**. This function is bi-directional, meaning that any member of a Unit category may be used in either **TO** or **FROM**.

| Unit of Measure | TO/FROM |
|---|---|
| Length | km (Kilometers), m (Meters), dm (Decimeters), cm (Centimeters), mm (Millimeters), nmi (Nautical miles), mi (Miles), ft (Feet), yd (Yards), in (Inches) |
| Area | m2 (Sq. Meters), cm2 (Sq. Centimeters), mm2 (Sq. Millimeters), ft2 (Sq. Feet), yd2 (Sq. Yards), in2 (Sq. Inches) |
| Velocity | m/s (Meters per second), cm/s (Centimeters per second), ft/s (Feet per second), km/h (Kilometers per hour), m/min (Meters per minute), ft/min (Feet per minute), mi/h (Miles per hour), knots (Knots) |
| Volume | m3 (Cubic meters), cm3 (Cubic centimeters), dm3 (Cubic decameters), l (Liters), ft3 (Cubic feet), in3 (Cubic inches) |
| Mass | kg (Kilograms), g (Grams), mg (Milligrams), lb (U.S. Pounds) |
| Pressure | Pa (Pascals), kPa (Kilo-Pascals), N/m2 (Newtons per sq. meter), bar (Bars), atm (Atmospheres), mbar (Millibars), psi (Pounds per sq. inch), mmHg (Millimeters of mercury), inHg (Inches of mercury) |
| Angular | deg (Degrees), rad (Radians) |

**TO/FROM Possible Conversion Arguments**

| Unit of Measure | TO/FROM |
|---|---|
| Acceleration | m/s2 (Meters per second sq.), cm/s2 (Centimeters per second sq), ft/s2 (Feet per second sq.), g (G's) |
| Angular Velocity | rpm (Rotations per minute), rps (Rotations per second), rad/s (Radians per second) |
| Temperature | C (Celcius), F (Farenheit), R (Rankine), re (Reaumur), K (Kelvin) |

**TO/FROM Possible Conversion Arguments (Continued)**

## Result Type/Space

D[*n*]

## Example

```
; Name              Units  Number  Result  Computations
"SqFeetToSqMeters"  "m2"   F300    F[15]   Units(1000, "m2", "ft2")
```

# VaxTime(), VAX Time

## Synopsis

**VaxTime**(A)

**A** Time acquisition tag for VAX clock (tag).

## Description

This function converts the VAX clock from milliseconds to an ASCII string for display purposes. The resulting string as the format '**HH:MM:SS**' where '**HH**' stands for hours, '**MM**' stands for minutes and '**SS**' stands for seconds.

## Result Type/Space

S[10]

## Example

```
; Name      Units  Number  Result   Computations
"VAXTime"    ""     F0      S[10]   VaxTime(A100)
```

# VaxTimeDiff(), VAX Time Difference

## Synopsis

**VaxTime**(A)
**A** Acquisition tag for VAX clock (tag).

## Description

This function uses the VAX clock data and the Model 300 time to compute the difference between the times in milliseconds.

## Result Type/Space

D[1]

## Example

```
; Name                Units  Number  Result   Computations
"VAXTimeDifference"    ""     F100    F[1]    VAXTimeDiff(A100)
```

# VectorAngle(), Vector Angle

## Synopsis

**VectorAngle**(X, Y)

| | |
|---|---|
| **X**[*m*] | Magnitude in the X direction (*m*≥1). |
| **Y** [*p*] | Magnitude in the Y direction (*p*≥1). |

## Description

This function is used to return the angle for a vector, whose Cartesian coordinates are (**X**, **Y**).

$$f[i] = \text{atan}\left(\frac{Y[i]}{X[i]}\right) \text{ (rad)}$$

$$if\ (f[i] < 0)\ \text{then}\ f[i] = f([i] + 2 \times \pi)$$

## Result Type/Space

D[*n*], *n* = min(*m*, *p* )

## Example

```
; Name           Units   Number   Result   Computations
"VectorAngle"    "rad"   F300     F[15]    VectorAngle(F100, F101)
```

# VectorLen(), Vector Length

## Synopsis

**VectorLen**(X, Y)

| | |
|---|---|
| **X**[*m*] | Magnitude in the X direction (*m*≥1). |
| **Y** [*p*] | Magnitude in the Y direction (*p*≥1). |

## Description

This function is used to return the length of a vector, whose Cartesian coordinates are (**X**, **Y**).

$$f[i] = \sqrt{(X[i]^2 + Y[i]^2)}$$

## Result Type/Space

D[*n*], *n* = min(*m*, *p*)

## Example

```
; Name         Units   Number   Result  Computations
"VectorLen"    "nm"    F200     F[1]    VectorLen(F100, F101)
```

# Vols(), Volumes

## Synopsis

**Vols**(PROBE, F, CFAC, TAS)
**Vols**(F, PROBE, RANGE, CFAC, TAS, INTERVAL)

| | |
|---|---|
| **PROBE** | Probe name/number (probe). |
| **F**[*m*] | Function for the sums of channel samples ($m \geq 1$). |
| **RANGE**[1] | Range used in probe configuration file (integer). |
| **CFAC**[1] | Correction factor. |
| **TAS**[1] | True air speed value. |
| **INTERVAL**[1] | Summation interval (integer). |

☞ ***Note: If the RANGE is not specified, it is passed via the probe entry argument (PROBE).***

## Description

This function converts the summed up channel counts and the probe definition table to compute volumes. The result is typically used for mean, median, mode and total volume calculations as well as X vs. Y display plots. This function should be 'refreshed' at the same time interval as the summation routine generates data, so as to eliminate redundant calculations on the same input data.

The 'SAREA' and 'VOLUME' originate from the user specified channel files via the probe name/number. The 'BUFLIFE' and 'SYSFREQ' refer to the values entered in the system table. The SYSFREQ is associated with the system frequency in the time data. This comes from the frequency values in the system board entry. The BUFLIFE is associated with the buffer life in the time data. In the M300 system, the buffer life and system frequency are the same (For synchronous buffers). The following formula summarizes the computations.

$$f[i] = \frac{F[i] \cdot VOLUME[i, RANGE]}{SAREA[i, RANGE] \cdot TAS \cdot \frac{BUFLIFE}{SYSFREQ} \cdot INTERVAL \cdot CFAC}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \min(m, [probe\ channels])$

## Example

```
; Name   Units  Number  Result   Computations
"Vols"    ""     F300    F[15]   Vols(F100, P0, 1, F54, F102, 1)
```

# Volts(), Volts

## Synopsis

**Volts**(A)
**A** Acquisition tag (tag).

## Description

This function uses all the information for the acquisition type, and converts the raw counts into volts. The function pays attention to acquisition type and gain settings.

## Result Type/Space

D[*n*], *n* = number of data samples

## Example

```
; Name   Units  Number  Result   Computations
"Volts"   ""     F100    F[10]   Volts(A100)
```

# Math Function Reference

Math operations operate directly on the floating point stack and work with floating values. The result of these operations is placed in the floating point stack, so that other operations may be performed on them. When the formula is completely evaluated, the final result is placed in the formula result space. The formula result space is then available, for example to text displays, strip charts, etc.

There are a few points of consideration to note when using M300 Mathematical Functions. The M300 formula tables use reverse polish notation (rpn) to represent mathematical evaluations. If you are unfamiliar with rpn this may be a good time to review our rpn basic concept section or find a more comprehensive resource. [Reverse Polish Notation Basic Concepts].

The following is a list of the math operations implemented (B is the last element and A is the next to last element on the stack).

| Operation | Operator | Function Description | Syntax | Page |
|---|---|---|---|---|
| Add | + | Adds last two numbers. | A B + | 452 |
| Subtract | - | Subtracts last two numbers. | A B - | 453 |
| Multiply | * | Multiplies last two numbers . | A B * | 454 |
| Divide | / | Divides last two numbers. | A B / | 455 |
| Modulus | % | Returns the remainder of the last two numbers. | A B % | 456 |
| Increment | ++ | Increments last number by one. | B ++ | 457 |
| Decrement | -- | Decrement last number by one. | B -- | 458 |
| Boolean AND | & | Bitwise AND operation on the last two numbers. | A B & | 459 |
| Boolean OR | \| | Bitwise OR operation on the last two numbers . | A B \| | 460 |
| Boolean XOR | ^ | Bitwise Exclusive OR operation on the last two numbers. | A B ^ | 461 |
| Boolean NOT | ~ | Bitwise inversion of the last number. | B ~ | 462 |
| Shift Left | << | Shifts next to last number left by last number bits. | A B << | 463 |
| Shift Right | >> | Shifts next to last number right by last number bits. | A B >> | 464 |
| Absolute Value | Abs | Absolute value of last number. | B abs | 465 |
| Arc Cosine | Acos | Inverse cosine of last number. | B acos | 466 |
| Hyperbolic Arc Cosine | Acosh | Inverse hyperbolic cosine of last number. | B acosh | 467 |
| Arc Sine | Asin | Inverse sine of last number. | B asin | 468 |

**Math Functions**

| Operation | Operator | Function Description | Syntax | Page |
|---|---|---|---|---|
| Hyperbolic Arc Sine | Asinh | Inverse hyperbolic sine of last number. | B asinh | 469 |
| Arc Tangent | Atan | Inverse tangent of last number. | B atan | 470 |
| Arc Tangent 2 | Atan2 | Inverse tangent, determining the quadrant. | B A atan2 | 471 |
| Hyperbolic Arc Tan | Atanh | Inverse hyperbolic tangent of the last number. | B atanh | 472 |
| Ceiling | Ceil | Integer ceiling of last number. | B ceil | 473 |
| Change Sign | Chs | Change sign of last number. | B - | 474 |
| Cosine | Cos | Cosine of last number. | B cos | 475 |
| Hyperbolic Cosine | Cosh | Hyperbolic cosine of last number. | B cosh | 476 |
| Exponential | Exp | Natural antilog of last number. | B exp | 477 |
| Floor | Floor | Integer floor of last number. | B floor | 478 |
| Hypotenuse | Hypot | Length of hypotenuse or right triangle with sides A and B. | A B hypot | 479 |
| Natural Logarithm | Ln | Natural log of last number. | B ln | 480 |
| Logarithm | Log | Logarithm base 10 of last number. | B log | 481 |
| Logarithm 2 | Log2 | Logarithm base 2 of last number. | B log2 | 482 |
| Long Rotate Left | Lrotl | Rotates a unsigned long integer A to the left by B bits. | A B lrotl | 483 |
| Long Rotate Right | Lrotr | Rotates a unsigned long integer A to the right by B bits. | A B lrotr | 484 |
| Power | Pow | Raises the next to last number to last power. | A B Pow | 485 |
| Rotate Left | Rotl | Rotates an integer A to the left by B bits. | A B rotl | 486 |
| Rotate Right | Rotr | Rotates an integer A to the right by B bits. | A B rotr | 487 |
| Sine | Sin | Sine of last number. | B sin | 488 |
| Hyperbolic Sine | Sinh | Hyperbolic sine of last number. | B sinh | 489 |
| Square Root | Sqrt | Square root of last number. | B sqrt | 490 |
| Swap 2 Bytes | Swap2 | Swap 2 bytes for last number. | B swap2 | 491 |
| Swap 4 Bytes | Swap4 | Swap 4 bytes for last number. | B swap4 | 492 |
| Swap 8 Bytes | Swap8 | Swap 4 bytes for last number. | B swap8 | 493 |
| Tangent | Tan | Tangent of last number. | B tan | 495 |
| Hyperbolic Tangent | Tanh | Hyperbolic tangent of last number. | B tanh | 494 |

**Math Functions**

| Operation | Operator | Function Description | Syntax | Page |
|---|---|---|---|---|
| Exchange | Xchg | Exchange last with next to last. | A B xchg | 496 |

**Math Functions**

# +, Add

## Synopsis

A B +

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function returns an array of values representing the addition of the two given arrays, element by element. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A[i] + B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name    Units  Number  Result   Computations
"Add"     ""     F101    F[10]    F10 F11 +
```

# -, Sub

## Synopsis

A B -

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function returns an array of values representing the difference of the two given arrays, element by element. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A[i] - B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name    Units  Number  Result   Computations
"Sub"     ""     F101    F[10]    F10 F11 -
```

# *, Multiply

## Synopsis

A B *

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function returns an array of values representing the multiplication of the two given arrays, element by element. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A[i] \cdot B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name     Units  Number  Result  Computations
"Mul"      ""     F101    F[10]   F10 F11 *
```

# /, Divide

## Synopsis

A B /

**A**[*m*] Next to last operand ($m \geq 1$).

**B**[*p*] Last operand ($p \geq 1$).

*Note: B should not contain any zero values or divide by zero will occur. For the values which are zero this function will generate an unknown value in the result of that operation.*

## Description

This function returns an array of values representing the division of the two given arrays, element by element. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[\ i] = \frac{A[i]}{B[i]}$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name      Units  Number  Result  Computations
"Div"       ""     F101    F[10]   F105 4.2 /
```

# %, Modulus

## Synopsis

A B %

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function performs the modulus operation on **A** with respect to **B**. The result is the remainder when **A** is divided by **B**. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = \text{Remainder}\left(\frac{A[i]}{B[i]}\right)$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name   Units  Number  Result   Computations
"Mod"     ""     F101    I[1]    F105 2 %
```

# ++, Increment

## Synopsis

**B** ++

**B**[*n*] Last operand ($n \geq 1$).

## Description

This function increments the last operand by one. The following formula summarizes the calculations.

$$s[\,i] = B[i] + 1$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name       Units  Number  Result  Computations
"Inc"        ""     F101    I[25]   F109 ++
```

# --, Decrement

## Synopsis

**B** --

**B**[*n*] Last operand ($n \geq 1$).

## Description

This function decrements the last operand by one. The following formula summarizes the calculations.

$$s[\,i] \;=\; B[i] - 1$$
$$\text{for } i \;=\; 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name       Units  Number  Result  Computations
"Dec"        ""     F101    I[25]   F109 --
```

# &, Boolean AND

## Synopsis

A B &

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function returns an array of values representing bit-wise AND operation of the two given arrays, element by element. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A[i] \& B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name      Units  Number  Result  Computations
"And"       ""     F101    I[1]    F105 0xFA &
```

# |, Boolean OR

## Synopsis

A B |

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function returns an array of values representing bit-wise OR operation of the two given arrays, element by element. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A[i] | B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result  Computations
"Or"        ""     F101    I[1]    F105 0xFA |
```

# ^, Boolean Exclusive OR

## Synopsis

A B ^

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function returns an array of values representing bit-wise Exclusive OR operation of the two given arrays, element by element. The bit position of the **A** is *X-OR-ed* with the same bit position in **B**. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A([i] \wedge B[i])$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result  Computations
"Xor"       ""     F101    I[1]    F105 0xFA ^
```

# ~, Boolean NOT

## Synopsis

**B** ~

**B**[*n*] Last operand ($n \geq 1$).

## Description

This function returns a value or an array of values representing the bit-wise *Boolean Not* operation of the given array, element by element. This operation ensures each bit position of **B** is inverted. (i.e. 00110011 = 11001100).

## Result Type/Space

D[*n*]

## Example

```
; Name      Units  Number  Result  Computations
"Not"       ""     F101    I[1]    F104 ~
```

# <<, Shift Left

## Synopsis

A B <<

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function shifts the bits that comprise the value of **A** to the left by **B** bit positions. For each bit position shifted, the Most Significant Bit (MSB)/leftmost bit is dropped off and a 0 is appended to the Least Significant Bit (LSB)/right most bit. This function uses Interpolation [See Interpolation].

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name          Units  Number  Result  Computations
"ShiftLeft"     ""     F101    I[1]    F105 2 <<
```

# >>, Shift Right

## Synopsis

A B >>

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function shifts the bits that comprise the value of **A** to the right by **B** bit positions. For each bit position shifted, the Least Significant Bit (LSB)/right most bit is dropped off and a 0 is appended to the Most Significant Bit (MSB)/left most bit.This function uses Interpolation [See Interpolation].

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name            Units  Number  Result  Computations
"ShiftRight"      ""     F101    I[1]    F105 2 >>
```

# abs(), Absolute Value

## Synopsis

B **abs**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This function returns a value or an array of values representing the absolute value of **B**. The following formula summarizes the calculations.

$$f[i] = |B[i]|$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name      Units   Number  Result  Computations
"Abs"       ""      F101    I[1]    F105 abs
```

# acos(), Inverse Cosine

## Synopsis

B **acos**

**B**[*n*] Last operand ($n \geq 1$).

***Note: B must be in the range [-1,1] or a domain error will occur.***

## Description

This function computes the inverse cosine in the range [0, π] for the given elements in **B**. The following formula summarizes the calculations.

$$s[i] = \text{acos}[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"ArcCosine"     ""     F302    F[5]    F105 acos
```

# acosh(), Inverse Hyperbolic Cosine

## Synopsis

B **acosh**

**B**[*n*] Last operand ($n \geq 1$).

*Note: B must be greater than 1.0 or a domain error will occur*

## Description

This function computes the inverse hyperbolic cosine of **B**. The following formula summarizes the calculations.

$$s[i] = \text{acosh}[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name           Units  Number  Result  Computations
"ArcHypCosine"   ""    F101    F[1]    F1012 acosh
```

# asin(), Inverse Sine

## Synopsis

B **asin**
**B**[*n*] Formula containing value or array of values ($n \geq 1$).

*Note: All Values in **B** must be in the range [-1,1] or a domain error will occur.*

## Description

This function computes the inverse sine in the range $[-\pi/2, \pi/2]$. The following formula summarizes the calculations.

$$s[i] = \text{asin}[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"InvSine"       ""     F101    F[1]    F122 asin
```

# asinh(), Inverse Hyperbolic Sine

## Synopsis

B **asinh**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This function computes the inverse hyperbolic sine of **B**. The following formula summarizes the calculations.

$$s[i] = \text{asinh}[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name           Units  Number  Result  Computations
"ArcSineHyp"     ""     F101    I[1]    1.2 asinh
```

# atan(), Inverse Tangent

## Synopsis

B **atan**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This function computes the inverse tangent in the range $[-\pi/2, \pi/2]$. The following formula summarizes the calculations.

$$f[i] = \text{atan}[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"InvTangent"    ""     F101    F[1]    F105 atan
```

# atan2(), Inverse Tangent, determining quadrant

## Synopsis

A B **atan2**

**A**[*m*] Next to last operand ($m \geq 1$).

**B**[*p*] Last operand ($p \geq 1$).

☞ ***Note: A and B cannot both equal zero or a domain error will occur. Or if B is zero, a division by zero error will occur***

## Description

This function computes the inverse tangent of (**A/B**) using the signs of both **A** and **B** to determine the quadrant of the return value. Function will return the inverse tangent in the range [–π, π]. The following formula summarizes the calculations.

$$s[i] = \operatorname{atan}\left[\frac{A[i]}{B[i]}\right]$$

$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name              Units  Number  Result  Computations
"ArcTanQuadrant"    ""     F101    F[1]    F105 F106 atan2
```

# atanh(), Inverse Hyperbolic Tangent

## Synopsis

B **atanh**
**B**[*n*] Formula containing value or array of values ($n \geq 1$).
***Note: B must be within the range [-1, 1] or a domain error will occur.***

## Description

This function computes the inverse hyperbolic tangent of **B**. The following formula summarizes the calculations.

$$s[i] = \text{atanh}[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"ArcTanHyp"     ""     F101    F[1]    F125 atanh
```

# ceil(), Ceiling

## Synopsis

B **ceil**
**B**[*n*] last operand ($n \geq 1$).

## Description

This function computes the smallest integer that is not less than **B** (Ceiling). The following formula summarizes the calculations.

$$s[i] = \lceil B[i] \rceil$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name        Units  Number  Result  Computations
"Ceiling"     ""     F101    I[1]    10.45 ceil
```

# chs(), Change Sign

## Synopsis

B **chs**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This function changes the sign of **B**. The following formula summarizes the calculations.

$$s[i] = -(B[i])$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name       Units  Number  Result  Computations
"Invert"     ""     F101    I[1]    F105 chs
```

# cos(), Cosine

## Synopsis

B **cos**

**B**[*n*] Last operand ($n \geq 1$).

*Note: Any values in B with large magnitude may yield a result with little or no significance.*

## Description

This function computes the cosine of **B**. The following formula summarizes the calculations.

$$s[i] = \cos[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name       Units  Number  Result  Computations
"Cosine"     ""     F1104   F[2]    F300 cos
```

# cosh(), Hyperbolic Cosine

## Synopsis

B **cosh**
**B**[*n*] Last operand ($n \geq 1$).
*Note: If any values in B are too large, a range error will occur.*

## Description

This function computes the hyperbolic cosine of **B**. The following formula summarizes the calculations.

$$s[i] = \cosh[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"HypCosine"     ""     F101    F[1]    F105 cosh
```

# exp(), Exponential

## Synopsis

B **exp**
**B**[*n*] Last operand ($n \geq 1$).
*Note: If any values in B are too large, a range error may occur.*

## Description

This function computes the natural exponential function of **B**. The following formula summarizes the calculations.

$$s[i] = e^{B[i]}$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name           Units  Number  Result  Computations
"Exponential"    ""     F101    F[10]   F13 exp
```

# floor(), Floor

## Synopsis

B **floor**
**B**[*n*] Last formula containing value or array of values ($n \geq 1$).

## Description

This function computes the smallest integer that is not less than **B** (Floor). The following formula summarizes the calculations.

$$s[i] = \lfloor B[i] \rfloor$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name        Units  Number  Result  Computations
"Floor"       ""     F101    I[1]    F105 floor
```

# hypot(), Hypotenuse

## Synopsis

A B **hypot**

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

☞ ***Note: Some computations may result in an overflow.***

## Description

This function computes the hypotenuse of a right triangle whose sides are **A** and **B** adjacent to that right angle. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = \sqrt{A[i]^2 + B[i]^2}$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name          Units  Number  Result  Computations
"Hypotenuse"    ""     F101    I[5]    F106 F107 hypot
```

# ln(), Natural Logarithm

## Synopsis

B **ln**

**B**[*n*] Last operand ($n \geq 1$).

*Note: If any values in B are negative or zero, a domain or range will occur respectively.*

## Description

This function computes the natural log of **B**. The following formula summarizes the calculations.

$$s[i] = \log_e B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"NaturalLog"    ""     F11     F[5]    F10 Ln
```

# log(), Logarithm

## Synopsis

B **log**
**B**[*n*] Last operand ($n \geq 1$).

*Note: If any values in* ***B*** *are negative or zero, a domain or range will occur respectively.*

## Description

This function computes the logarithm (base 10) of **B**.

$$s[i] = \log_{10} B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"Logarithm"     ""     F11     F[1]    F10 log
```

# log2(), Binary Logarithm

## Synopsis

B **log2**
**B**[*n*] Last operand ($n \geq 1$).
*Note: If any values in B are negative or zero, a domain or range will occur respectively.*

## Description

This function computes the logarithm (base 2) of **B**.

$$s[i] = \log_2 B[i]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name                Units  Number  Result  Computations
"BinaryLogarithm"     ""     F10     F[5]    F9 log2
```

# lrotl(), Long Rotate Left

## Synopsis

A B **lrotl**

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function rotates the number **A** by **B** bits to the left. This function will handle long integer types (4 bytes) and uses Interpolation [See Interpolation].

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name              Units  Number  Result   Computations
"LongRotateLeft"     ""     F101    L[1]    F105 4 lrotl
```

# lrotr(), Long Rotate Right

## Synopsis

A B **lrotr**

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function rotates the number **A** by **B** bits to the right. This function will handle long integer types (4 bytes) and uses Interpolation [See Interpolation].

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name              Units  Number  Result  Computations
"LongRotateRight"    ""     F101    L[1]   F105 4 lrotr
```

# pow, Power Function

## Synopsis

A B **pow**

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This function raises **A** to the power of **B**. This function uses Interpolation [See Interpolation]. The following formula summarizes the calculations.

$$s[i] = A[i]^{B[i]}$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name            Units  Number  Result   Computations
"SquareFormula"    ""     F306    I[10]   F305 2 Pow
```

# rotl(), Rotate Left

## Synopsis

A B **rotl**

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

*Note: For long integer types, use Lrotl(). See "lrotl(), Long Rotate Left"*

## Description

This function rotates the number **A** by **B** to the left. This function uses Interpolation [See Interpolation].

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name          Units  Number  Result  Computations
"RotateLeft"     ""     F101    I[1]   F105 4 rotl
```

# rotr(), Rotate Right

## Synopsis

A B **rotr**

**A**[*m*] Next to last operand ($m \geq 1$).

**B**[*p*] Last operand ($p \geq 1$).

*Note: For long integer types, use Lrotl(). See "lrotr(), Long Rotate Right"*

## Description

This function rotates the number **A** by **B** bits to the right. This function uses Interpolation [See Interpolation].

## Result Type/Space

D[*n*], $n = \max(m, p)$

## Example

```
; Name           Units  Number  Result  Computations
"RotateRight"    ""     F101    I[1]    F105 4 rotr
```

# sin(), Sine

## Synopsis

B **sin**
**B**[*n*] Last operand ($n \geq 1$).

*Note: Values in B of large magnitude may yield a result with little or no significance.*

## Description

This function computes the sine of **B**.

$$s[i] = \sin[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name     Units  Number  Result  Computations
"Sine"     ""     F1104   F[2]    F300 sin
```

# sinh(), Hyperbolic Sine

## Synopsis

B **sinh**
**B**[*n*] Last operand ($n \geq 1$).
***Note: If any values in B are too large, a range error will occur.***

## Description

This function computes the hyperbolic sine of **B**.

$$s[i] = \sinh[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name        Units  Number  Result  Computations
"HypSine"     ""     F201    F[5]    F200 sinh
```

# sqrt(), Square Root

## Synopsis

B **sqrt**
**B**[*n*] Last operand ($n \geq 1$).
*Note: If any values in B are negative, a domain error will occur.*

## Description

This function calculates the square root of **B**.

$$s[i] = \sqrt{B[i]}$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result   Computations
"SquareRoot"     ""     F101    F[5]     F105 sqrt
```

# swap2(), Swap 2 Bytes

## Synopsis

B **swap2**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This operator performs a bytes swap on the 2 bytes of **B**.

## Result Type/Space

D[*n*]

## Example

```
; Name            Units  Number  Result   Computations
"Swap"              ""     F101   I[15]   F105 swap2
```

# swap4(), Swap 4 Bytes

## Synopsis

B **swap4**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This operator performs a bytes swap on the 4 bytes of **B**.

## Result Type/Space

D[*n*]

## Example

```
; Name            Units  Number  Result   Computations
"Swap"              ""     F101   L[15]   F105 swap4
```

# swap8(), Swap 8 Bytes

## Synopsis

B **swap8**
**B**[*n*] Last operand ($n \geq 1$).

## Description

This operator performs a bytes swap on the 8 bytes of **B**.

## Result Type/Space

D[*n*]

## Example

```
; Name            Units  Number  Result   Computations
"Swap"              ""     F101   I[15]   F105 swap8
```

# tanh(), Hyperbolic Tangent

## Synopsis

B **tanh**
**B**[*n*] Last operand ($n \geq 1$).

***Note: Values of large magnitude may yield partial or total loss of significance.***

## Description

This function returns the hyperbolic tangent of **B**.

$$s[i] = \tanh[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name          Units  Number  Result  Computations
"HypTangent"    ""     F106    F[25]   F105 tanh
```

# tan(), Tangent

## Synopsis

B **tan**
**B**[*n*] Last operand ($n \geq 1$).

*Note: Values of large magnitude may yield partial or total loss of significance.*

## Description

This function computes the tangent of **B**.

$$s[i] = \tan[B[i]]$$
$$\text{for } i = 0 \ldots [n-1]$$

## Result Type/Space

D[*n*]

## Example

```
; Name            Units  Number  Result   Computations
"Tangent"         ""     F101    F[15]    F105 tan
```

# xchg(), Exchange

## Synopsis

A B **xchg**

| | |
|---|---|
| **A**[*m*] | Next to last operand ($m \geq 1$). |
| **B**[*p*] | Last operand ($p \geq 1$). |

## Description

This functions put the values of **A** into **B** and vice versa. The new size of array **B** will be equal to '*m*' as shown above. Also, the new size of array **A** will be equal to '*p*' as shown above. The following two formulas summarize this process.

$$A[i] = B[i]$$
$$\text{for } i = 0 \ldots [n-1], \text{ where } n = p$$
$$B[i] = A[i]$$
$$\text{for } i = 0 \ldots [n-1], \text{ where } n = m$$

## Result Type Space

## Example

```
; Name       Units  Number  Result  Computations
"Exchange"   ""     F100    F[1]    F105 F106 xchg
```

# Command Manager Reference

The Command Manager is a runtime portion of the M300 system that will execute commands specified in the Trigger Command Table (tic.300) and the Command Table (cmd.300) tables. For commands that the user executes based on predefined keystrokes, see "Command Table, (cmd.300)". For commands that will execute on predefined triggers, see "Triggered Command Table, (tic.300)". The user may also enter commands from the M300 command line (check the M300 User's guide for more information on the command line).

The following is a list of the commands presently available.

***Note: M300 Command Manager commands are not case sensitive.***

| Command Name | Command Description | Page |
|---|---|---|
| asc | Control ASCII manager. | 508 |
| back | Sends the M300 Main Window to the back of display. | 518 |
| broadcast | Toggles the M300 broadcast. | 515 |
| clear error | Clears the M300 Main Window Error box. | 518 |
| cmd1d | Control the command byte for a 1D Board. | 504 |
| cmd2g | Control the command byte for a 2D Grey Board. | 505 |
| cmdaimms | Control the AIMMS purge. | 507 |
| end | Jumps to the end of a playback file. | 511 |
| file | Starts and stops M300 record. | 512 |
| fml | Performs various M300 formula operations. | 513 |
| front | Brings the M300 Main Window to the front of display. | 518 |
| fwa | Sends commands to a Formula Watch and Alter display. | 514 |
| lbl | Sends commands to a Label display. | 516 |
| lst | Sends commands to a List display. | 517 |
| mam | Sends commands to a Moving Air Mass display. | 520 |
| minimize | Minimizes the M300 Main Window. | 518 |
| next | Skips to the next buffer of a playback file. | 511 |
| open | Opens the M300 to the current console or vice versa. | 518 |
| pause | Pauses the M300. | 511 |

**Table 16: Command Reference**

| Command Name | Command Description | Page |
|---|---|---|
| pos | Performs various operations on the Position display. | 521 |
| quit | Exits the M300 software. | 515 |
| restart | Perform a stop followed by a short delay and start. | 511 |
| restore | Restores the M300 Main Window to normal size. | 518 |
| rewind | Rewinds a playback file. | 511 |
| run | Runs a specified application. | 515 |
| scn | Performs QNX screen console operations. | 523 |
| shutdown | Shuts down the M300 and the QNX OS. | 515 |
| skt | Sends commands to a Skew-T display. | 525 |
| start | Starts. | 511 |
| stop | Stops. | 511 |
| stp | Performs various Strip Chart data operations. | 526 |
| tas2d | Changes 2D True Air Speed frequency. | 506 |
| tas2g | Changes 2D Grey True Air Speed frequency. | 505 |
| tascip | Changes CIP True Air Speed frequency. | 509 |
| tascipgs | Changes CIPGS True Air Speed frequency. | 510 |
| time | Search for specific time in the data file. | 511 |
| txt | Sends commands to a Text display. | 528 |
| wnd | Performs various data display window operations. | 529 |
| xvy | Sends commands to an X vs. Y display. | 531 |

**Table 16: Command Reference (Continued)**

# Command Manager Prototype Quick Reference

The following table lists the command prototypes for quick reference purposes.

| Command Prototype | Page |
|---|---|
| **asc on\|off**<br>**asc** from [to] **on\|off**<br>**asc** from [to] **close**<br>**asc** from [to] **create**<br>**asc** name **create** [filename]<br>**asc** from [to] **fire** | 508 |
| **back** | 518 |
| **broadcast on\|off** | 515 |
| **clear error** | 518 |
| **cmd1d** board command<br>**cmd1d** board **auto** | 504 |
| **cmd2g** board command [**and\|or**]<br>**cmd2g** board **auto** | 505 |
| **cmdaimms** board purgeTime<br>**cmdaimms** board **purge**<br>**cmdaimms** board **heatontemp** temp | 507 |
| **end** | 511 |
| **file on\|off**<br>**file close** [**read**]<br>**file open** filename<br>**file create** [filename] | 512 |

**Command Quick Reference**

| Command Prototype | Page |
|---|---|
| **fml** formula **auto**<br>**fml** formula **hold**<br>**fml** formula [index] value [**auto**]<br>**fml** formula **+** value [**auto**]<br>**fml** formula **-** value [**auto**]<br>**fml** formula * value [**auto**]<br>**fml** formula **/** value [**auto**]<br>**fml** formula **AND** value [**auto**]<br>**fml** formula **&** value [**auto**]<br>**fml** formula **OR** value [**auto**]<br>**fml** formula **\|** value [**auto**]<br>**fml** formula **XOR** value [**auto**]<br>**fml** formula **^** value [**auto**] | 513 |
| **front** | 518 |
| **fwa new** | 514 |
| **lbl** from [to] **on\|off** | 516 |
| **lst** from [to] **on\|off**<br>**lst** from [to] **clear** | 517 |
| **mam** from [to] **max** max<br>**mam** from [to] **rings** rings<br>**mam** from [to] **set** | 520 |
| **minimize** | 518 |
| **next** | 511 |
| **open** [current] | 518 |
| **pause** [**cancel**]<br>**pause** hh:m:ss | 511 |

**Command Quick Reference (Continued)**

| Command Prototype | Page |
|---|---|
| **pos** from [to] **auto** [value]<br>**pos** from [to] **center**<br>**pos** from [to] **clat** clat<br>**pos** from [to] **clear** [**all**]<br>**pos** from [to] **clon** clon<br>**pos** from [to] **ewmiles** ewmiles<br>**pos** from [to] **freq** freq<br>**pos** from [to] **freq auto**<br>**pos** from [to] **nsmiles** nsmiles<br>**pos** from [to] **on\|off**<br>**pos** from [to] **set**<br>**pos** from [to] **wbarb on\|off**<br>**pos** from [to] **zoom** zoom<br>**pos** from [to] **zoom in\|out** | 521 |
| **quit** | 515 |
| **restart** [delay] | 511 |
| **restore** | 518 |
| **rewind** | 511 |
| **run** command [&] | 515 |
| **scn** console<br>**scn up**<br>**scn down**<br>**scn left**<br>**scn right**<br>**scn prev**<br>**scn next**<br>**scn home**<br>**scn end**<br>**scn last** | 523 |
| **shutdown** | 515 |
| **skt** from [to] **clear** | 525 |
| **start** | 511 |
| **stop** | 511 |

**Command Quick Reference (Continued)**

| Command Prototype | Page |
|---|---|
| **stp** from [to] **on\|off**<br>**stp** from [to] color<br>**stp** "group name" **group**<br>**stp** from [to] **base** base<br>**stp** from [to] **lim** min max<br>**stp** from [to] **max** max<br>**stp** from [to] **min** min<br>**stp** from [to] **offset** offset<br>**stp** from [to] **range** range | 526 |
| **tas2d** board frequency<br>**tas2d** board **auto** | 506 |
| **tas2g** board frequency<br>**tas2g** board **auto** | 505 |
| **tascip** board frequency<br>**tascip** board **auto** | 509 |
| **tascipgs** board frequency<br>**tascipgs** board **auto** | 510 |
| **time cancel**<br>**time** hh:mm:ss | 511 |
| **txt** from [to] **on\|off**<br>**txt** from [to] color | 528 |
| **wnd** from [to] **back**<br>**wnd** from [to] **close**<br>**wnd** from [to] **front**<br>**wnd** from [to] **maximize**<br>**wnd** from [to] **minimize**<br>**wnd** from [to] **open**<br>**wnd** from [to] **restore**<br>**wnd** from [to] **print**<br>**wnd** from [to] **bmp** [file]<br>**wnd** from [to] **jpg** [file]<br>**wnd** from [to] **tif** [file]<br>**wnd move**<br>**wnd next**<br>**wnd prev**<br>**wnd** [from] [to] **lock\|unlock**<br>**wnd** [from] [to] **pause\|unpause**<br>**wnd on/off** | 529 |

Command Quick Reference (Continued)

| Command Prototype | Page |
|---|---|
| **xvy** from [to] **on\|off**<br>**xvy** from [to] color<br>**xvy** from [to] **clear**<br>**xvy** "group name" **group**<br>**xvy** from [to] x**base** base<br>**xvy** from [to] **xlim** min max<br>**xvy** from [to] **xmax** max<br>**xvy** from [to] **xmin** min<br>**xvy** from [to] **xoffset** offset<br>**xvy** from [to] **xrange** range<br>**xvy** from [to] y**base** base<br>**xvy** from [to] **ylim** min max<br>**xvy** from [to] **ymax** max<br>**xvy** from [to] **ymin** min<br>**xvy** from [to] **yoffset** offset<br>**xvy** from [to] **yrange** range | 531 |

**Command Quick Reference (Continued)**

# 1D Commands

## Synopsis

**cmd1d** board command
**cmd1d** board **auto**

| | |
|---|---|
| board | 1D Board name (board). |
| command | Command to be sent (integer). |

## Description

Changes the command byte for the specified 1D board. The **board** is the name defined in the M300 setup for the 1D card to be changed. The possible values for the **command** is [0, 255], although most of the time the values go from 0-3. This command is normally used to change range on the FSSP probes or turn the pump on/off on the PCASP probes.

The **auto** is used to restore control of the 1D probe control to the control functions.

## Example

```
;
.A F1
cmd1d fssp 1
```

# 2D Grey Commands

## Synopsis

**cmd2g** board command [**and**|**or**]
**cmd2g** board **auto**
**tas2g** board frequency
**tas2g** board **auto**

| | |
|---|---|
| board | 2D Grey Board name (board). |
| command | Command to be sent (byte: 0-255). |
| frequency | Frequency to be sent (float). |

## Description

### 2D Grey Byte Command

This command is used to change the command byte for the 2D Grey probe. The **board** is the name defined in the M300 setup for the 2D Grey card to be changed. The **and** or **or** operations are optional. These operations are used to turn on (**or**) or off (**and**) individual bits to the probe command (see example below).

**command** - Current command value.

**auto** - Restore control of the 2D Grey probe command to the control functions.

### 2D Grey True Air Speed Command

This command can be used to change the 2D Grey probe true air speed frequency. The value specified will override the control function in the formula table (fml.300).

**board** - The board name specifies a unique 2D Grey interface card as defined in the board table.

**frequency** - The desired fequency value in MHz.

**auto** - Restore control of the 2D Grey probe true air speed frequency to the control functions.

## Example

```
;
.A F1
cmd2g 2dg 0x01
.A F2
cmd2g 2dg 0x02 OR
.A F3
cmd2g 2dg 0x04 AND
. F4
cmd2g 2dg auto
;
.A F6
tas2g 2dg 3.5
.A F5
tas2g 2dg auto
```

# 2D Mono Commands

## Synopsis

**tas2d** board frequency
**tas2d** board **auto**

| | |
|---|---|
| board | 2D board name (string). |
| frequency | Frequency to be sent (float). |

## Description

This command can be used to change the 2D probe true air speed frequency. The value specified will override the control function from formula table.

**frequency** - The frequency must be specified in MHz.
**board** - The board name specifies a unique 2D interface card as defined in the board table.
**auto** - Restore control of the 2D probe true air speed frequency to the control functions.

## Example

```
;
.A F1
tas2d 2dc 2.5
.A F5
tas2d 2dc auto
```

# AIMMS Commands

## Synopsis

**cmdaimms** board purgeTime [**on|off|auto**]
**cmdaimms** board **purge** [**on|off|auto**]
**cmdaimms** board **heatontemp** [temp]

| | |
|---|---|
| board | AIMMS (or ADP) board name (board). |
| purgeTime | Purge time to be sent (ms) (integer, 10 - 65535). |
| on | Turn on all purge ports. |
| off | Turn off all purge ports. |
| auto | Let AIMMS (or ADP) perform the purge sequence. |
| temp | Temperature (in C) (float). |

## Description

AIMM/ADP commands.

Perform the purge cycle for the specified AIMMS board (or ADP). The purgeTime is in ms. It has to be larger than 10 ms.

The M300 send the command for the purge for each port with the specified purge time (port 1, port 2, port 3). There are three valid ports. Each port is purged separatly.The M300 waits 4 seconds before sending the next purge command.

When specifying a purgeTime, this doesn't change the purge time in the AIMMS board entry.

The **purge** is used to send the default purge command with purge time from the board table for AIMMS/ADP.

The on keyword instructs the AIMMS/ADP to turn on all purge ports (port 0, if supported).

The off keyword instructs the AIMMS/ADP to turn off all purge ports (port 255, if supported).

The auto keyword instructs the AIMMS/ADP to perform the purge sequence (port 4, if supported).

Change the low temperature threshold value for the AIMMS 20, with the 'heatontemp' command.

## Example

```
;
.A F1
cmdaimms Aimms20 purge
```

# ASCII Commands

## Synopsis

**asc on|off**
**asc** from [to] **on|off**
**asc** from [to] **close**
**asc** from [to] **create**
**asc** name **create** [filename]
**asc** from [to] **fire**
**asc** name **send** data ... data

| | |
|---|---|
| from | First ASCII name\|number to perform operation on. |
| to | Last ASCII name\|number to perform operation on [optional]. |
| name | Name of ASCII entry. |
| filename | Name of ASCII file to create [optional]. |
| data ... data | Data string to be transmitted on serial output. |

## Description

The ASCII commands are used to control the ASCII manager entries.

**on|off** - Changes the state of the ASCII output to either **on** or **off**. When the state is off, the ASCII will not update.

**on|off** - When using a list of ASCII entries to change the state, this command controls the active state of individual entries.

**close** - Close the file name for the specified entries.

**create** - Automatically create an ASCII output file for the specified entries. The file extension is '.csv', for command separated values.

**create** - If a file name given for a specific ASCII entry, then a file will be created with the given name.

**fire** - Run an ASCII entry one time only. The entry must be in the off state in order to run.

**send** - This will send data to a serial type device. The termination character must not be 0.

## Example

```
;
.A F1
asc off
. F1
asc 0 create gps.csv
. F2
asc 0 close
```

# Cloud Imaging Probe (CIP) Commands

## Synopsis

**tascip** board frequency
**tascip** board **auto**

| | |
|---|---|
| board | CIP Board name (string). |
| frequency | Frequency to be sent (float). |

## Description

This command can be used to change the CIP probe true air speed frequency. The **frequency** should be specified in MHz. The value specified will override the control function in the formula table (fml.300). The **board** name specifies a unique CIP interface card as defined in the board table.

The use of **auto** in place of **frequency** is used to restore control of the CIP probe true air speed frequency to the control functions.

## Example

```
;
.A F1
tascip cip 2.5
.A F5
tascip cip auto
```

# Cloud Imaging Grey Scale Probe (CIPGS) Commands

## Synopsis

**tascipgs** board frequency
**tascipgs** board **auto**

| | |
|---|---|
| board | CIPGS Board name (string). |
| frequency | Frequency to be sent (float). |

## Description

This command can be used to change the CIPGS probe true air speed frequency. The **frequency** should be specified in MHz. The value specified will override the control function in the formula table (fml.300). The **board** name specifies a unique CIPGS interface card as defined in the board table.

The use of **auto** in place of **frequency** is used to restore control of the CIPGS probe true air speed frequency to the control functions.

## Example

```
;
.A F1
tascipgs cipgs 2.5
.A F5
tascipgs cipgs auto
```

# Control Commands

## Synopsis

**rewind**
**start**
**stop**
**pause** [**cancel**]
**pause** hh:mm:ss
**next**
**end**
**restart** [delay]
**time cancel**
**time** hh:mm:ss
delay Delay time in ms, default is 250 ms (integer).

## Description

**rewind** - Rewinds the M300 file being played back. Not that this operation does not work when the M300 is recording.

**start** - Pushes in the M300 play button to begin file operations. If the record button has already been set, then the M300 will immediately begin to record a file. If the record button is not pressed in, then normal playback will begin. Not that a file must have been created or opened prior to this command.

**stop** - Stops all M300 file operations (record or playback).

**pause** - Pauses all M300 file operations (playback only). Specify time to pause at with hh:mm:ss field. The time based command may be canceled, with cancel keyword.

**time** - Search the M300 data file (playback only). Specify the time to search for with hh:mm:ss field. The timed search may be canceled with the cancel keyword.

**next** - Skips to the next buffer during playback.

**end** - Moves the file pointer to the end of the file during playback.

**restart** - Stop, delay and start.

## Example

```
;
. F5
rewind
;
.AC F2
start
;
.AC F3
stop
```

# File Operations Commands

## Synopsis

**file on|off**
**file close** [**read**]
**file open** filename
**file create** [filename]
filename File name to create or open.

## Description

Toggles the recording mode on and off. If there is a file currently open, then the write will begin with operation set to **on**, otherwise the M300 will wait until a file is opened before it begins the writing to a file. Starts or stops M300 recording to a file. This is the same as clicking on the record button.

**on|off** - Changes the state of the M300 recording output to either **on** or **off**. When the state is off, the record will not update.

**close** - Close M300 binary file (write file). Or if using **read** option, then close the M300 playback file (read file).

**open** - Open specified file name for reading (playback mode).

**create** - Create a new M300 binary file for writing. Use with extreme caution, if file already exists, it will be overwritten. If no file name is given for the create, then the file name is automatically generated by the M300. In either case, the '.sea' extension is added to the file name.

## Example

```
;
. F1
file On
#
file Off
. F2
file create
. F3
file close
```

# Formula Commands

## Synopsis

**fml** name|number **auto**
**fml** name|number **hold**
**fml** name|number [index] value [**auto**]
**fml** name|number [operation] value [**auto**]

| | |
|---|---|
| name\|number | Formula name or number. |
| index | Index value in formula [optional] (integer). |
| operation | Math operation to be performed [optional] (string) |
| value | Value argument. |
| **auto** | Restore auto control (string constant) |

## Description

The formula commands have various meanings. Prototype 1 above is used to restore control of all the given formula **name/number** to the formula manager. Prototype 2 is used to hold the current formula value. The formula value will be overridden with the current value until is set in auto mode or a new value is specified. Prototype 3 above is used to override the given formula **name/number** with the given **value**. If **index** is given, element[**index**] will be overwritten, else element[0] will be overwritten. If **auto** is passed, the **value** is retained for one-cycle, after which it will return to the auto state; else the formula value will remain overwritten**.** Prototype 4 **above** is used to perform a mathematical operation on the given formula **name/number** with the original formula value as the first operand and the given **value** as the second operand. Currently, the valid mathematical operations that can be passed as the **operation** argument are: addition (+), subtraction (-), division (/), multiplication (*), Boolean AND (**AND** or **&**), Boolean OR (**OR** or |), and Boolean Exclusive OR (**XOR** or **^**).

## Example

```
;
. F5
fml F1001 auto
;
.A F1
fml F200 10 15.5 auto
.C F5
fml F200 0.5
#
fml F200 auto
.A F2
fml F101 10 auto
```

# Formula Watch and Alter Display Commands

## Synopsis

**fwa new**

## Description

This command can be used to change various settings of the Formula Watch and Alter display.
**new** - Creates a new Formula Watch and Alter window on the current console.

## Example

```
;
. F1
fwa new
```

# General Commands

## Synopsis

**broadcast on|off**
**run** command [&]
**shutdown**
**quit**

command The command that is to be run by the M300 (string).

## Description

**broadcast** - Starts or stops broadcasting across the network using UDP. Valid values for operation are **on** or **off**. The broadcast button in the M300 main window will change according to this value as well. This is the same as click on the broadcast button.

**run** - Runs the specified **command** in the background. Note that although the **&** argument is optional, if it is not included at the end of the command, the M300 will stop and wait for the **command** to terminate before continuing (i.e. it will not run in the background).

**shutdown** - Exits the M300 application. Following the closing of the M300 application, QNX will shutdown. If you decide to setup a **shutdown** or **quit** key for the M300, make sure you pick a combination that cannot be hit inadvertently. WARNING: You will not be prompted for a confirmation.

**quit** - Closes the M300 application. Any open data files are saved and closed automatically.

## Example

```
;
. F10
broadcast on
#
broadcast off
;
.CAS F12
shutdown
;
. F5
quit
;
. F10
run RawView &
```

# Label Display Commands

## Synopsis

**lbl** from [to] **on|off**

| | |
|---|---|
| from | First label name\|number to perform operation on. |
| to | Last label name\|number to perform operation on [optional]. |

## Description

This command performs various **operations** on the M300 Label displays. The M300 will perform the **operation** on all of the Label displays in the entry list that lie between and including **from** and **to**. Currently, the M300 supports the following **operations**:

**on|off** - Changes the state of the list displays to either **on** or **off**. When the state is off, the display will not update.

## Example

```
;
. F1
lbl 0 3 off
#
lst 0 3 on
```

# List Display Commands

## Synopsis

**lst** from [to] **on|of**
**lst** from [to] **clear**
**lst** from [to] **fire**

| | |
|---|---|
| from | First list entry to perform operation on. |
| to | Last list entry to perform operation on [optional]. |

## Description

This command performs various operations on the M300 List display. The M300 will perform the operation on all of the List displays in the entry list that lie between and including **from** and **to**. Currently, the M300 supports the following operations.

**on|off** - Changes the state of the list displays to either **on** or **off**. When the state is off, the display will not update.

**clear** - Clears the list displays.

**fire** - Run an list entry one time only. The entry must be in the off state in order to run.

## Example

```
;
. F1
lst 0 7 state off
#
lst 0 7 state on
;
. F2
lst 0 7 clear
```

# Main Window Commands

## Synopsis

**front**
**back**
**restore**
**minimize**
**clear error**
**open** [**current**]

## Description

**front** - Brings the M300 Main Window to the foreground, in front of all other windows.

**back** - Puts the M300 Main Window behind all other open windows.

**restore** - Restores the M300 Window to a normal state. This will work only if the M300 has been previously minimized or maximized.

**minimize** - Minimizes the M300 main window to the QNX taskbar.

**clear error** - Clears the last error message displayed in the M300 Main Window Error Message.

**open** - Automatically opens and switches the console to where the M300 Main Window is. Note that this opens the Main Window only. If the **current** string is included in the command, the M300 will be open in the current console with the same x and y offset.

***Note: That these commands affect the M300 Main Window only. To control data displays,*** *See "Display Window Commands"*

## Example

```
;
. F5
front
;
. F6
back
;
. F1
restore
;
. F11
minimize
;
.A F10
move
;
. F5
open
```

```
. F6
open current
;
clear error
```

# Moving Air Mass Display Commands

## Synopsis

**mam** name **set**
**mam** name **max** value
**mam** name **rings** value

| | |
|---|---|
| name | Moving Air Mass display name (string). |
| action | Action to be carried out (string). |
| value | Value argument (integer). |

## Description

**set** - sets the center latitude and longitude values to be the current attitude and longitude values. This essentially marks the moving air mass.

**rings** - sets the number of rings to display on the Moving Air Mass display as specified by the **value** argument.

**max** - this will set the maximum range of the Moving Air Mass display as specified by the **value** argument.

## Example

```
;
.A F1
mam cloudPointer set
.A F2
mam cloudPointer rings 10
#
mam cloudPointer rings 5
```

# Position Display Commands

## Synopsis

**pos** from [to] operation [value]
**pos** from [to] **clear** [**all**]
**pos** from [to] **freq auto**
**pos** from [to] **wbarb on|off**
**pos** from [to] **zoom** [direction]

| | |
|---|---|
| from | First position name\|number to perform operation on. |
| to | Last position name\|number to perform operation on [optional]. |
| operation | Operation to be carried out (string). |
| value | Value argument. |
| direction | Direction to zoom (string constant) |

## Description

These commands perform various operations on the M300 position display. The M300 will perform this **operation** on all of the position displays in the entry list that lie between and including **from** and **to**. Currently, the M300 supports the following **operations**:

**auto** - Change the auto (nautical miles to edge) value for the position entry.
**center** - Centers the position display on the aircraft.
**clat** - Changes the Center of Latitude (cLat) to the given **value** (the value is required).
**clear** - Clears any wind barb or data point that has been drawn.
**clon** - Changes the Center of Longitude (cLon) to the given **value** (the value is required).
**ewmiles** - Changes the East-West Miles (ewMiles) to the given **value** (the value is required).
**freq** - Changes the display frequency (the frequency is required).
**nsmiles** - Changes the North-South Miles (nsMiles) to the given **value** (the value is required).
**on|off** - Turn on or off the indicate entry.
**set** - Set a new data point or wind bard in to memory.
**wbarb on|off** - Turn on or off the wind barb for the indicated entry.
**zoom** - Changes the zoom percentage to the given **value**.

The valid values for **direction** are **in** and **out**. If the string constant **in** is specified, the position display will zoom in use a doubling factor (2x) of the current zoom value. Likewise, if the string constant **out** is specified, the position display will zoom out using a halving factor (x/2) of the current zoom value.

## Example

```
;
. F1
pos 0 clon 150.0
#
pos 0 clon 300.0
. F2
pos 0 1 zoom 100.0
;
.A F5
pos 0 1 zoom in
.A F6
pos 0 1 zoom out
;
. F10
pos 0 1 center
```

# Screen Console Commands

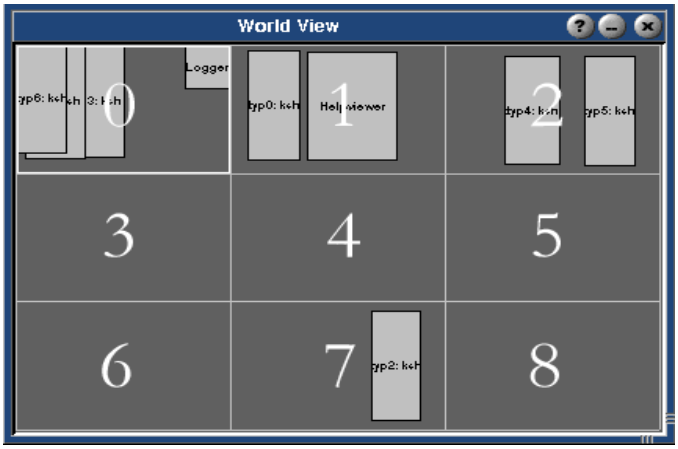
## Synopsis

**scn** console
**scn** operation

| | |
|---|---|
| console | Console number to switch to (integer). |
| operation | Operation to be carried out (string). |

## Description

This function performs various operations on the QNX console display. The following describe the different **console** commands that are valid:




**M300 Figure 17: QNX Console World View**

[**0-8**] - Switches the current console being displayed to a console specified by the integer value given. The QNX desktop manager has 9 desktop consoles in a 3x3 grid, with each one being independent from the other (i.e. console 0 and console 1 have different windows on them). If a multi-monitor configuration exists, one console will span all monitors.

The following describe the different **operation** commands that are valid

**left** - sets the active console to next console to the left. This command forces the display to stay on one level of the 3x3 console array. (Horizontal).

**right** - sets the active console to next console to the right. This command forces the display to stay on one level of the 3x3 console array. (Horizontal).

**up** - sets the active console to the next upper level in the 3x3 display array. If the user is currently at the top of the display array, then console switches to the bottom most console level. (Vertical).

**down** - sets the active console to the next lower level in the 3x3 display array. If the user is currently at the bottom of the console array, then console switches to the top most console level. (Vertical).

**next** - sets the active console to the next console (in sequence), regardless of which console level it is on. If the console is currently console 8, then console 0 will become active.

**prev** - sets the active console to the previous console (in sequence), regardless of which console level it is on. If the console is currently console 0, then console 8 will become active.

**home** - sets the active console to console 0.

**end** - sets the active console to console 8.

**last** - switch between current and last console.

## Example

```
;
. F1
scn 0
#
scn 1
#
scn 2
;
. F2
scn prev
. F3
scn next
```

# Skew-T Display Commands

## Synopsis

**skt** from [to] **clear**

| | |
|---|---|
| from | First entry to perform operation on. |
| to | Last entry to perform operation on [optional]. |

## Description

This command performs various operations on the M300 Skew-T display. The M300 will perform the operation on all of the Skew-T displays in the entry list that lie between and including **from** and **to**. Currently, the M300 supports the following operations.

**clear** - Clears the Skew-T displays.

## Example

```
;
. F2
skt 3 clear
```

# Strip Chart Display Commands

## Synopsis

**stp** from [to] **on|off**
**stp** from [to] color
**stp** "group name" **group**
**stp** from [to] **base** base
**stp** from [to] **lim** minimum maximum
**stp** from [to] **max** maximum
**stp** from [to] **min** minimum
**stp** from [to] **offset** offset
**stp** from [to] **range** [range]

| | |
|---|---|
| from | First strip entry to perform operation on. |
| to | Last strip entry to perform operation on [optional]. |
| color | Color to changes strip entry to (string). |
| "group name" | Group name (string). |
| minimum | Minimum to changes strip entry to (float). |
| maximum | Maximum to changes strip entry to (float). |
| range | Range to changes strip entry to (float). |
| base | Base to changes strip entry to (float). |
| offset | Offset to changes strip entry to (float). |

## Description

The M300 will perform the operation on all of the strip chart entries in the entry list that lie between and including **from** and **to**. Currently, the M300 supports the following operations.

**on|off** - turns the strip chart entries to the on or off state respectively.

**color** - changes the color of the strip chart entries based on the **color** string passed (See "Color System" for listing of valid colors).

**group** - turn on a group of strip chart entries.

**lim** - sets the strip chart entry minimum and maximum values.

**min** - sets the strip chart entry minimum value.

**max** - sets the strip chart entry maximum value.

**range** - sets the strip chart entry range to the current value ± (0.5 * **range**). If **range** is omitted, the Strip Chart will perform auto-ranging. That is, the limits of the Strip Chart will be changed to accommodate all points visible.

**base** - sets the strip chart entry minimum to the **base** specified. The new maximum is the (new minimum + current range).

**offset** - changes the minimum and maximum by the **offset** specified. The new minimum is (current min + **offset**) and the new maximum is (current max + **offset**).

## Example

```
;
.A F10
stp 0 3 off
stp 4 6 on
.C F1
stp 0 2 red
.C F2
lim 0 2 -100 100
#
lim 0 2 -1 1
```

# Text Display Commands

## Synopsis

**txt** from [to] [**on|off**]
**txt** from [to] color

| | |
|---|---|
| from | First text entry name\|number to perform operation on. |
| to | Last text entry name\|number to perform operation on [optional]. |

## Description

This command performs various operations on the M300 Text display entries. The M300 will perform the operation on all of the Text display entries in the entry list that lie between and including **from** and **to**. Currently, the M300 supports the following operations.

color - changes the color of the text entries based on the **color** string passed (See "Color System" for listing of valid colors).

**on|off** - turns the text entries to the on or off state respectively.

## Example

```
;
. F5
txt 52 61 dgreen
#
txt 52 61 dred
. F6
txt 0 98 off
#
txt 0 98 on
```

# Display Window Commands

## Synopsis

**wnd** from [to] **close**
**wnd** from [to] **open**
**wnd** from [to] **front**
**wnd** from [to] **back**
**wnd** from [to] **minimize**
**wnd** from [to] **maximize**
**wnd** from [to] **restore**
**wnd** from [to] **print**
**wnd** from [to] **bmp** [filename]
**wnd** from [to] **jpg** [filename]
**wnd** from [to] **tif** [filename]
**wnd move**
**wnd next**
**wnd prev**
**wnd** [from] [to] **lock|unlock**
**wnd** from [to] **pause|unpause**
**wnd primary|secondary**
**wnd on|off**

| | |
|---|---|
| from | First window to perform operation on. |
| to | Last window to perform operation on [optional]. |
| filename | File name [optional]. |

## Description

This command performs various operations on the M300 data display windows. Some operations can be performed on multiple windows, while certain operations can only be performed on one window at a time, which is usually the current window that has focus. If no M300 window has the focus then the operation would be performed on the window which last had the focus. The following describe the different window operations.

**close** - Closes the specified windows. Note that these windows can then be opened, either by another defined command (i.e. wnd 0 10 open) or by using the M300 graphical interface. No data updates are performed on windows that are closed.

**open** - Opens the display windows.

**front** - Brings the specified windows to the front of the screen.

**back** - Puts the specified windows behind all other windows in the display.

**minimize** - Minimizes the display windows. Note that these windows are not closed and can be restored quickly. Data updates are still performed on windows that are minimized.

**maximize** - Maximizes the data display windows.

**restore** - Restores the specified windows to the display.

**print** - Print the specified windows to the default printer.

**bmp** - Capture the specified windows to bit map format files. If a single window is captured the file name may be given. If the file name is omitted the system will automatically generate a file name (using date and time).

**jpg** - Capture the specified windows to JPEG format files. If a single window is captured the file name may be given. If the file name is omitted the system will automatically generate a file name (using date and time).

**tif** - Capture the specified windows to TIFF format files. If a single window is captured the file name may be given. If the file name is omitted the system will automatically generate a file name (using date and time).

**move** - Begins a move operation on the window currently in focus. The M300 will place the mouse pointer over the window to be moved. Drag the highlighted rectangle to the point you move the window and press the mouse button. This will complete the move operation. Note that the **move** command can be used on windows without a title bar.

**next** - Brings the next data display window in the M300 window list that is not closed into focus.

**prev** - Brings the previous data display window in the M300 window list that is not closed into focus.

**lock** - Lock display windows (global lock). This is similar to the "Toggle Window Display" button in the M300 window. Certain window features will be locked. The window features to be locked can be controlled by the window properties selecting the flags tab.

**unlock** - Lock display windows (global unlock). This feature unlocks all windows. Notice that local locks on the window might still apply. In other words, if a window has it's lock button pressed, then that window will still be locked.

**pause** - Pause display windows.

**unpause** - Un-pause display windows.

**primary** - Switch to primary window position scheme.

**secondary** - Switch to secondary window position scheme.

**on** - Open all windows.

**off** - Close all windows.

## Example

```
;
. F1
wnd 0 5 close
#
wnd 0 5 open
#
. F2
wnd 4 minimize
#
wnd 2 3 minimize
;
. F3
wnd prev
. F4
wnd next
```

# X vs. Y Display Commands

## Synopsis

**xvy** from [to] **on|off**
**xvy** from [to] color
**xvy** from [to] **clear**
**xvy** "group name" **group**
**xvy** from [to] x**base** base
**xvy** from [to] x**lim** minimum maximum
**xvy** from [to] x**max** maximum
**xvy** from [to] x**min** minimum
**xvy** from [to] x**offset** offset
**xvy** from [to] x**range** [range]
**xvy** from [to] y**base** base
**xvy** from [to] y**lim** minimum maximum
**xvy** from [to] y**max** maximum
**xvy** from [to] y**min** minimum
**xvy** from [to] y**offset** offset
**xvy** from [to] y**range** [range]

| | |
|---|---|
| from | First X vs. Y entry to perform operation on. |
| to | Last X vs. Y entry to perform operation on [optional]. |
| color | Color to changes X vs. Y entry to (string). |
| "group name" | Group name (string). |
| minimum | Minimum to changes X vs. Y entry to (float). |
| maximum | Maximum to changes X vs. Y entry to (float). |
| range | Range to changes X vs. Y entry to (float). |
| base | Base to changes X vs. Y entry to (float). |
| offset | Offset to changes X vs. Y entry to (float). |

## Description

This command performs various operations on the M300 X vs. Y display. The M300 will perform the operation on all of the X vs. Y displays in the entry list that lie between and including **from** and **to**.

**on|off** - turns the X vs. Y entries to the on or off state respectively.
**color** - changes the color of the X vs. Y entries based on the **color** string passed (See "Color System" for listing of valid colors).
**clear** - Clears the X vs. Y displays.
**group** - turn on a group of X vs. Y entries.

**xlim | ylim** - sets the X vs. Y entry minimum and maximum values (x or y).

**xmin | ymin** - sets the X vs. Y entry minimum value (x or y).

**xmax | ymax** - sets the X vs. Y entry maximum value (x or y).

**xrange | yrange** - sets the X vs. Y entry range to the current value ± (0.5 * **range**). If **range** is omitted, the X vs. Y will perform auto-ranging. That is, the limits of the X vs. Y will be changed to accommodate all points visible (x or y).

**xbase | ybase** - sets the X vs. Y entry minimum to the **base** specified. The new maximum is the (new minimum + current range) (x or y).

**xoffset | yoffset** - changes the minimum and maximum by the **offset** specified. The new minimum is (current min + **offset**) and the new maximum is (current max + **offset**) (x or y).

## Example

```
;
. F5
xvy 0 3 clear
```

# Setup Tables Reference

The M300 uses and extensive array of tables to store computations, display settings and many other types of configuration information. One difference between the M200 and the M300 is the shrinking need to become extensively familiar with these setup tables. Ultimately, the M300 will be refined to the point where the setup tables will be completely transparent and there will be no requirement for the user to know about them. The setup tables in this reference are described in detail so that a user can configure the system to do exactly what they require. There are a couple of important points to remember whenever changing or creating setup tables:

- The setup tables require a strong degree of syntactical correctness but the M300 does not have extensive syntax or error checking because these tables are designed to implement a wide variety of system configurations designed by the user.
- If you are changing your setup tables, you should back them up to another place to ensure that you can always go back to the original project.
- Backup individual project files before modifying them. For example, if you want to make changes to the txt.300 file, first make a backup of this file, say txt.300.bak. Then you can modify the txt.300 file. If the changes you made look good you can keep the file. If the changes caused undesired effects or trouble with the normal system operation, you can always go back to the last good known file.
- Keep in mind that most features can be easily and quickly added by copying from other project or even copying from the same project.
- Most M300 setup tables will end with the "* .300" file extension.
- We keep a current set of template tables in the '/test/tables' directory of the M300 system. These tables are ASCII text tables that can be viewed with any basic editor.
- In addition to the template tables mentioned above, we also keep several other different project setups under the '/test' directory. These can be used to test the instruments and interface cards. These projects are simple and can be used standalone without the complexity of larger research project setups. Another benefit for these projects is that they can also be used as a starting point to add a particular instrument to the an existing M300 project. For example, check the '/test/seaadc' to see a project which shows all the analog voltages from the SEA Analog System, or the '/test/1d0' project to see a project which shows bin counts, reference voltage and other variables from a 1D type probe.

***WARNING:* After making changes to the tables the user must always test for correctness in the data displays.**

The following is a list of the setup tables used by the M300.

| File Name | Config File | Other Files | Setup Table | Page |
|---|---|---|---|---|
| 2dg.300 | n/a | n/a | 2D Grey Particle Display Table | 541 |
| 2dm.300 | n/a | n/a | 2D Mono Particle Display Table | 543 |
| acq.300 | n/a | n/a | Acquisition Event Table | 545 |

**M300 Setup Tables**

| File Name | Config File | Other Files | Setup Table | Page |
|---|---|---|---|---|
| asc.300 | *.asc | *.csv | Ascii (text) File Output Table | 548 |
| *.asc | n/a | n/a | Ascii Output Config Files | 550 |
| brd.300 | *.brd | n/a | Board Setup Table | 552 |
| *.brd | n/a | *.dsp, *.eee, *.cap, *.def | Board Config Files | 555 |
| btn.300 | n/a | n/a | Button Display Table | 563 |
| buf.300 | n/a | n/a | Buffer Setup Table | 567 |
| cfg.300 | n/a | n/a | Project Configuration Table | 570 |
| cgs.300 | n/a | n/a | CIP Grey Scale Display Table | 571 |
| cip.300 | n/a | n/a | CIP Display Table | 573 |
| cmd.300 | n/a | n/a | Command Table | 575 |
| fml.300 | n/a | n/a | Formula Table | 577 |
| fwa.300 | n/a | n/a | Formula Watch & Alter Table | 586 |
| his.300 | n/a | n/a | Histogram Display Table | 588 |
| hod.300 | n/a | n/a | Hodograph Display Table | 590 |
| hsa.300 | n/a | n/a | High Speed Analog Display Table | 592 |
| hti.300 | n/a | n/a | Height Time Indicator Display Table | 594 |
| hvp.300 | n/a | n/a | HVPS Particle Display Table | 596 |
| lbl.300 | n/a | n/a | Label Display Table | 598 |
| lst.300 | *.lst | n/a | List Display Table | 601 |
| *.lst | n/a | n/a | List Display Config Files | 603 |
| lup.300 | *.lup | n/a | Lookup Table | 604 |
| *.lup | n/a | n/a | Lookup Config Files | 605 |
| mam.300 | n/a | n/a | Moving Air Mass Pointer Display Table | 606 |
| pdi.300 | n/a | n/a | Probe Distribution Display Table | 608 |
| pos.300 | *.tgt, *.map | n/a | Position Display Table | 610 |
| *.tgt, *.map | n/a | n/a | Target/Map Files | 610 |

**M300 Setup Tables (Continued)**

| File Name | Config File | Other Files | Setup Table | Page |
|---|---|---|---|---|
| ppi.300 | n/a | n/a | Plan Position Indicator Display Table | 618 |
| prb.300 | *.chn, *.prb | n/a | Probe Table | 620 |
| *.chn, *.prb | n/a | n/a | Probe Channel Files | 622 |
| prj.300 | n/a | n/a | Project Table | 624 |
| rdr.300 | n/a | n/a | Radar Table | 625 |
| saq.300 | n/a | n/a | Secondary Acquisition Table | 627 |
| skt.300 | n/a | n/a | Skew-T Display Table | 628 |
| stp.300 | n/a | n/a | Strip Chart Display Table | 630 |
| tic.300 | n/a | n/a | Triggered Command Table | 632 |
| txt.300 | n/a | n/a | Text Display Table | 634 |
| wnd.300 | *.wnd | n/a | Window Table | 637 |
| *.wnd | n/a | n/a | Window Config Files | 640 |
| xvy.300 | n/a | n/a | X vs. Y Display Table | 644 |

**M300 Setup Tables (Continued)**

In addition to backing up the setup files mentioned above raw binary M300 data files (*.sea) should be backed up and/or transferred to another medium or system. This will ensure the safety of the configuration and data files.

# Standard conventions for parameters in setup project files.

String identifiers are allocated a total of 32 characters internally. Since we use C as the programming language, strings are terminated with the '\0' or null character. This means that string identifiers can have a maximum of 31 characters (32-1).

Identifiers in the M300 system are case sensitive. This mean that if you define an acquisition event 'DewPoint' and then you want to use it in the formula table (fml.300), you should use 'Aq.DewPoint' and not 'Aq.dewpoint'.

Most identifiers can have spaces in their names. Sometimes, this may or may not be desirable. If we have an acquisition event defined as 'Dew Point' with tag number 100 and then we want to use it in the formula table, we must use A100 as opposed to trying to use 'Aq.Dew Point'. The formula table would see the 'Aq.Dew Point' as two tokens and this would cause all sorts of problems.

If you are not familiar with hexadecimal and binary notations, then we recommend you find a source of information for this and learn the basics. The M300 system uses hexadecimal numbers everywhere. In C the hexadecimal numbers start with '0x' and this is the notation we use for the M300 system.

The maximum number of characters in a line is 1023 (1024-1). This should be sufficient for most if not all tables.

The following parameters described are seen in almost all of the setup files. We have listed them here as a quick reference. Other setup parameters that are specific to a particular type of board can be found in their respective sections.

**Comments**

Comments can be entered in almost all setup files. Anything after a semi-color ';' will be treated as a comment and ignored by the system for processing. You can specify a comment anywhere on a line. Instances where comments are not allowed will be specifically documented. It is not recommend to place comments at the end of files. They are allowed as comments but will be lost when the system saves the tables.

**Name**

The name is the identifier for the entry. If the name has spaces it must be enclosed in double quotes. The name can be a maximum of 31 characters and it must be unique.

**Window**

This parameter is the name of the window where the display will be performed. The window name represents also a window configuration file and therefore can't contain any spaces. The window can have a maximum of 31 characters. There must be a valid window name specified in the window table (See "Window Table, (wnd.300)"). The type for the window entry must match the desired display. There must also be a window configuration file (See "Window Table Configuration File, (*.wnd)") for the specified window.

**Color**

The color parameter can be used to specify the desired color for a particular entry or for a specify feature of an entry. Each color value represents a 24 bit RGB value (0xRRGGBB). The red, green and blue values can go between 0 and 255 (0-0xFF). Normally the user doesn't need to know the specific RGB values for a color to be able to select the desired color. The colors are normally picked through a color dialog picker and then save to the file when the project is saved. Examples of color values are, 0x000000 for black, 0xFFFFFF for white, 0x00FF0000 red, 0x00FF00 for green, 0x0000FF for blue. In addition to color values in RGB, the user can also specify color names, such as red, blue, green, etc. For a complete list of colors (See "Color System").

**Board**

This parameter is the name of the board specified in the board table (See "Board Table, (brd.300)"). In the board table there is an entry for each board as well as a board type. Each board is configured via the board configuration file (See "Board Table Configuration File, (*.brd)"). This entry can have a maximum of 31 characters (no spaces allowed). Board entries in the formula table must be preceded by 'Bd.' to indicate a board entry. For example if we have defined an 'arinc' board in the board table and we want to use it in the formula table (See "Formula Table, (fml.300)"), then we have to use 'Bd.arinc'. Board and Address can be used interchangeably, as long as they are pointing to the same entity.

**Probe**

A probe entry from the probe table (See "Probe Table, (prb.300)"). This entry can have a maximum of 31 characters (no spaces allowed). Probe entries in the formula table must be preceded by 'Pr.' to indicate a probe entry. For example if we have defined a 'fssp' probe entry in the probe table and we want to use it in the formula table (See "Formula Table, (fml.300)"), then we have to use 'Pr.fssp'.

**Tag**

Acquisition event tag. This identifies the data source. Valid tag numbers are integers in the range of 0 to 65535, not including the reserved tag (See "Reserved Tag Numbers").

**Address**

The address field is commonly used to identify a board address. Each board entry must have unique address. In some instances the board address is preferred way to specify a board, as opposed to the board name. This parameter is specified in hexadecimal notation and it starts with a '0x', for example '0x1700' would typically refer to the first 2D board.

**State**

The state field is used to control whether or not an entry is active/on (use 1) or off (use 0). Other state values may be possible/defined in the future.

**Formula**

The formula number or name from the formula table (See "Formula Table, (fml.300)") for the data to be used. Formula numbers must start with the 'F' character followed by a number (for example, F2000). In the M200 system the valid formula numbers where in the range of 0 to 65535

($2^{16}$ - 1). The M300 can have formulas in the range of 0 to 2147483647 ($2^{31}$ - 1). Formula names must be valid identifiers with a maximum of 31 characters. You can use spaces in formula names, but this may be a problem if you want to refer to the formula by name.

**Format**

The format control string consists of:

*Ordinary characters:* These are written exactly as they occur in the format string.

*Conversion specifiers:* These cause argument values to be written as they are encountered during the processing of the format string.

An ordinary character in the format string is any character, other than a percent character (%), that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format string that begins with a percent character (%) and is followed, in sequence, by the following:

Zero or more format control flags that can modify the final effect of the format directive

An optional decimal integer, or an asterisk (*), that specifies a minimum field width to be reserved for the formatted item

An optional precision specification in the form of a period (.), followed by an optional decimal integer or an asterisk (*)

An optional type length specification:

One of h, l, L, w, N or F

A character that specifies the type of conversion to be performed:

One of the characters: cdeEfFgGinopsuxX.

The valid format control flags are:

**Minus sign "-"**

The formatted item is left-justified within the field; normally, items are right-justified.

**Plus sign "+"**

A signed, positive object will always start with a plus character (+); normally, only negative items begin with a sign.

**Space " "**

A signed, positive object will always start with a space character; if both + and a space are specified, + overrides the space.

**Pound sign**

"#" an alternate conversion form is used.

**Octal "O"**

For o (unsigned octal) conversions, the precision increments, if necessary, so that the first digit is 0.

**Hexadecimal "X"**

For x or X (unsigned hexadecimal) conversions, a non-zero value is appended with a '0x' or '0X' respectively for e, E, f, g or G (any floating-point) conversions, the result always contains a

decimal-point character, even if no digits follow it; normally, a decimal-point character appears in the result only if there is a digit to follow it. In addition to the preceding, for g or G conversions, trailing zeros are not removed from the result.

**Field Width**

If no field width is specified, or if the value that is given is less than the number of characters in the converted value (subject to any precision value), a field of sufficient width to contain the converted value is used.

If the converted value has fewer characters than are specified by the field width, the value is padded on the left (or right, subject to the left-justification flag) with spaces or zero characters (0). If the field width begins with a zero, the value is padded with zeros, otherwise the value is padded with spaces.

If the field width is *, a value of type int from the argument list is used (before a precision argument or a conversion argument) as the minimum field width. A negative field width value is interpreted as a left-justification flag, followed by a positive field width.

**Precision Specifier**

As with the field width specifier, a precision specifier of * causes a value of type int from the argument list to be used as the precision specifier. If no precision value is given, a precision of 0 is used. The precision value affects the following conversions:

**Integer**

For d, i, o, u, x and X (integer) conversions, the precision specifies the minimum number of digits to appear.

**Floating Point (fixed)**

For e, E and f (fixed-precision, floating-point) conversions, the precision specifies the number of digits to appear after the decimal-point character.

**Floating Point (variable)**

For g and G (variable-precision, floating-point) conversions, the precision specifies the maximum number of significant digits to appear.

**String**

For s (string) conversions, the precision specifies the maximum number of characters to appear.

# 2D Grey Probe Display Table, (2dg.300)

## Overview

This is used to display particle image data for 2D Grey types (advanced or regular). The user can select a color for each intensity level. This display has the capability of hashing out old images via a user selectable age limit. The image data is identified via the board address for the 2D Grey data. The image may be scaled. The 2D Grey display has an age counter, which keeps track of how many seconds have elapsed since the last valid image display. The number of displays (buffers) that the user can see per second can be controlled via the primary trigger frequency for the window. A secondary trigger of one second is necessary for the age counters and hashing feature.

In the case of 2D Grey advanced data, one entire buffer is displayed per window. This would typically mean several particles per window.

For regular 2D Grey data, each M300 data buffer has only one particle. The 2D Grey display will buffer the 2D Grey particles (buffers) into one larger buffer for the display. The 2D Grey display will fit as many M300 buffers (particles) as it can per window or internal buffer. When we have a slow data rate for 2D Grey data, keep this in mind as to why it takes a little while before the data gets displayed.

## Parameters

**Name**

The name is the identifier for the 2D Grey entry. For example, “2DGC" (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300 system. If the user has multiple 2DG displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a 2DG display does also, then a command set up to change the color of the 2DG display will not affect the HVPS display.

**Window**

Each entry in the 2D Grey display table need to belong to a window. This parameter is the name of the window where the 2D Grey display will be done. The type of the window must be 2D Grey display. For example, “2dgc” (see also, "Window" on page 537).

**Color25, Color50, Color75**

The 2D Grey probe has 2 bits per pixel. Each bit is assigned an intensity level (25%, 50% and 75%). The 2D Grey display allows the user to pick whatever colors he desires for each intensity level. The color for each intensity level maybe the same as another or even the background color. The use of

the background color for the lowest intensity level could be useful in removing undesirable noise from the display (such as splash, stuck diodes, etc.) (see also, "Color" on page 538).

**Address**

The address selects the 2D Grey data. The user doesn't need to know the tag number for the 2D Grey data, just the address of the board where the 2D Grey data is coming from. Valid addresses are 0x2300, 0x2700 and 0x2B00 (other address possible). When the user changes the address, the primary trigger for the window also gets changed. This allows the display to run only when there is 2D Grey data available (see also, "Address" on page 538).

**Timebars**

Along with each 2D Grey particle there are also two slices containing the timing data for the particle. The 2D Grey display can display these in the same color as the particle (use a 1) or as the background (use a 0). When the time bars are shown with the background color, they are not 'removed' from the display. This leaves the particles in the same position, regardless of whether or not the time bars are shown.

**Scale**

The user can scale the 2D Grey particles by a desired value. The default scale value is 1. The larger the scale value, the larger the particles will appear on the display. Larger particles may mean less particles per display window.

**AgeLimit**

The ageLimit is used to hash out an old display. Once the current 2D Grey display is older than the specified ageLimit, then the display gets hashed out as an indication of old data. This parameter is specified in seconds. The window must have the secondary trigger set to expire once per second on the synchronous buffer.

**Probe**

This is the probe name from the probe table (See "Probe Table, (prb.300)" on page 620.). This is used to associate a probe table entry with a 2D Grey display entry (see also, "Probe" on page 538).

## Example

```
; Version = 3
; 2dg.300
; Name Number Window Color25 Color50 Color75 Address Timebars Scale AgeLimit    Probe
"2DG"       0    2DG     red    blue   green    2DG        1     2      10        2dg
```

# 2D Mono Probe Display Table, (2dm.300)

## Overview

This display is used to display particle image data of 2D Mono type. The user can select a color for the images. This display has the capability of hashing out old images via a user selectable age limit. The image data is identified via the board address for the 2D Mono data. The image may be scaled. The 2D Mono display has an age counter, which keeps track of how many seconds have elapsed since the last valid image display. The number of displays (buffers) that the user can see per second can be controlled via the primary trigger frequency for the window.

The 2D Mono display is made up of several strips. Each strip, displays as many slices as possible. There are a total of 1024 slices in the 2D Mono display. Slices are 32 bits/pixels wide (4 bytes). We have a one bit/pixel mapping for the 2D Mono display.

## Parameters

**Name**

The name is the identifier for the 2D Mono entry. For example, “2DC 25" (see also, "Name" on page 537).

**Number**

A unique integer (Note that multiple 2DM displays can have the same integer) used to identify this display to the M300. If the user has multiple 2DM displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a 2DM display does also, then a command set up to change the color of the 2DM display will not affect the HVPS display.

**Window**

Each entry in the 2D Mono display table need to belong to a window. This parameter is the name of the window where the 2D Mono display will be done. The type of the window must be 2D Mono display. For example, “2dg25” (see also, "Window" on page 537).

**Color**

The 2D Mono probe has 1 bit per pixel. Each bit can be on or off. The 2D Mono display allows the user to pick whatever color he desires for the images (see also, "Color" on page 538).

**Address**

The address selects the 2D Mono data. The user doesn’t need to know the tag number for the 2D Mono data, just the address of the board where the 2D Mono data is coming from. Valid addresses are 0x1700, 0x1B00 and 0x1F00 (other addresses possible). When the user changes the address, the primary trigger for the window also gets changed. This allows the display to run only when there is 2D Mono data available (see also, "Address" on page 538).

**Timebars**

Along with each 2D Mono particle there are also two slices containing the timing data for the particle. The 2D Mono display can display these in the same color as the particle (use a 1) or as the background (use a 0). When the time bars are shown with the background color, they are not 'removed' from the display. This leaves the particles in the same position, regardless of whether or not the time bars are shown.

**Scale**

The user can scale the 2D Mono particles by a desired value. The default scale value is 1. The larger the scale value, the larger the particles will appear on the display. Larger particles may mean less particles per display window.

**AgeLimit**

The ageLimit is used to hash out an old display. Once the current 2D Mono display is older than the specified ageLimit, then the display gets hashed out as an indication of old data. This parameter is specified in seconds. The window must have the secondary trigger set to expire once per second on the synchronous buffer.

**Probe**

This is the probe name from the probe table (See "Probe Table, (prb.300)" on page 620.). This is used to associate a probe table entry with a 2D Mono display entry.

Please note that the 'sync pattern' for the timebars is specified in the probe table. Since the 2D Mono display knows about the probe entry, it can find the 'sync pattern' and use it when necessary. We have used the probe entry as opposed to specifying the 'sync pattern' here, since this reduces the number of places where the 'sync pattern' must be entered. Basically, the 'sync pattern' goes with the probe and the user uses the probe entry where necessary (see also, "Probe" on page 538).

## Example

```
; Version = 3
; 2dm.300
; name      number window     color board timebars scale ageLimit probe
"2DC"            0    2DC       red    2DC        1     1       20   2dc
"2DP"            1    2DP     green    2DP        1     4       15   2dp
```

# Acquisition Event Table, (acq.300)

## Overview

The acquisition event table is used to define all the acquisition events and their properties. Each acquisition event must belong to a board and have a valid acquisition type. The tag number and the name identify the acquisition event and must be unique. Acquisition events can be on/off. Data size, parameter one, parameter two and parameter three must be setup correctly as per the acquisition type reference manual.

Some of the parameter values may not be necessary to setup in the M300 system as compared to the same acquisition types in the M200 system. This is a direct result of the fact that we now have entries for every board. The board entry is responsible to configure and setup the board. This task was done in part by the acquisition events in the early M200 days. We have tried to make it easier to understand the board configuration, by removing the parameter settings from the acquisition events. Some acquisition events still need to have settings entered via the parameters. Here are some general guidelines to follow when editing the acquisition table.

Each acquisition event can have a frequency up to the maximum system frequency (specified in the system board entry, which must be defined).

The tag number must not be a reserved tag number (link to reserved tag numbers). In fact no duplicate tag numbers can be used with other acquisition events.

Once you determine the acquisition event type that you require you should check the acquisition type reference for that event (See “Acquisition Reference” on page 37.).

For acquisition events to be complete they must belong to a board as mentioned before, but they must also be associated with a data buffer. Each acquisition event in the acquisition table, must also have an entry in the buffer table (See “Buffer Table, (buf.300)” on page 567.).

The recommended way to make changes to the acquisition events is in the acquisition setup menu in the M300 system.

***WARNING:*** **The order of the acquisition events must follow the same order as board table entries. Comments will be saved out of sequence otherwise.**

***WARNING:*** Manual **modifying this table by adding or removing an acquisition entry also requires modification of the ‘buf.300’ table.**

## Parameters

**Name**

The name is the identifier for the acquisition event entry. This name can be used in other tables to access the raw data. For example, “Temperature" (see also, "Name" on page 537).

**Tag**

Tag number for acquisition event. This number can be used in other tables to access the raw data. The tag number must be unique. Don't use a reserved tag number (link to reserved tags) (see also, "Tag" on page 538).

**Frequency**

Desired frequency value for acquisition event (integer value). This can be between 1 and the maximum system frequency (specified in the system board entry). Valid frequency values are entered in the acquisition setup dialog for the current system board frequency.

**State**

The state controls whether a particular acquisition event is active/on (1) or off (0). If you turn off a particular acquisition event, then there will be no directory nor data entry for this event in the raw data (see also, "State" on page 538).

**Size**

Data size for acquisition event. This value changes from acquisition event to acquisition event. Consult the acquisition type reference manual for information on the valid data sizes for each acquisition event type. When you use the acquisition setup dialog in the M300, the data size will get filled automatically for most acquisition events. There are some acquisition events where the data size must be provided by the user. If the size of the data is overridden care must be taken to ensure that a large enough value is specified.

***WARNING:*** **Not specifying enough data size is a serious problem which might cause the M300 to crash. At the very least the data will be missing or incorrect.**

**Type**

The data type for the acquisition event. Valid types are a byte (0-255) (see also, "Acquisition Reference" on page 37).

**Para1, Para2, Para3**

These are extra configuration/setup parameters for the acquisition event. The meaning of these parameters changes from acquisition type to acquisition type. Consult the acquisition type reference manual for information on the valid parameter values for each acquisition event type. In the M200 system, these parameters sometimes configured how a particular interface card was setup. Since we now have entries for each interface in the board table, some of these parameter values are no the same as in the case of the M200. When a parameter is used to configure the acquisition event, we have kept the same configuration values as the M200 system.

**Board**

Each acquisition event must belong to a board. This field links the acquisition event to the correct board. It is very important to have the correct acquisition events associated with the correct board. For example, it makes no sense to do a serial acquisition type from a 1D type board as it doesn't make any sense to do the 1D acquisition from a serial type board. The M300 acquisition dialog setup ensures that only the valid acquisition types are presented for each board. The user must take special care when doing this manually.

**SampleOffset**

The sample offset select in which acquisition tick the acquisition event gets acquired. In the past all synchronous acquisition events got done in the first tick. This can "overloaded" the first tick. Usually the duration in the proxy was longest for the first acquisition tick. We can balance the acquisition load over several ticks (allowing us to do more). This is only applies to synchronous acquisition events.

## Example

```
; Version = 2
; acq.300
; Name              Tag     Freq. State Size Type Para1  Para2  Para3  Board SampleOffset
"Latitude"          1000    1     1     4    16   0xC8   0x00   0x00   Arinc429         0
"Longitude"         1001    1     1     4    16   0xC9   0x00   0x00   Arinc429         0
"True Heading"      1002    1     1     4    16   0xCC   0x00   0x00   Arinc429         0
"Magnetic Heading"  1003    1     1     4    16   0xD0   0x00   0x00   Arinc429         0
"Pitch"             1004    1     1     4    16   0xD4   0x00   0x00   Arinc429         0
"Roll"              1005    1     1     4    16   0xD5   0x00   0x00   Arinc429         0
"Altitude"          1006    1     1     4    16   0xF1   0x00   0x00   Arinc429         0
"FwdAntAz"          1500    2     1     4    16   0x00   0x01   0x00   Arinc429         0
"FwdAntTilt"        1501    2     1     4    16   0x01   0x01   0x00   Arinc429         0
"I and Q A"         2000    1     1     2400 100  0x00   0x00   0x00   PiraqA           0
"Config A"          2001    2     1     324  101  0x00   0x00   0x00   PiraqA           0
"Status A"          2002    2     1     76   102  0x00   0x00   0x00   PiraqA           0
"I and Q B"         3000    1     1     2400 100  0x00   0x00   0x00   PiraqB           0
"Config B"          3001    2     1     324  101  0x00   0x00   0x00   PiraqB           0
"Status B"          3002    2     1     76   102  0x00   0x00   0x00   PiraqB           0
```

# ASCII Output Table, (asc.300)

## Overview

This table can be used to create ASCII (text) data files from the data system during real-time and/or playback from a raw binary file. The text files can be used in another system to run data analysis software. There must be some formulas defined in the formula table so that we can output the data from these (output source).

The ASCII file output can be configured to output to the printer or serial port. To output to the printer port the user must specify the printer port name, for example '/dev/par1'. To output to the serial port the user must specify the serial port name, for example '/dev/ser1'. The serial port parameters can be controlled via a serial port entry in the board table. The 'stty' command can also be used to configure the serial port parameters. This needs to be done at least once per system power up, before the M300 runs.

The data delimiter is configured via the format parameter or a another ASCII entry. The termination of each line is user configurable. The user can choose which line termination characters to use and where to place them. Most often there is an entry in the formula table for carriage return and line feed and then one or more ASCII entries to control line termination.

The recording of ASCII data files can be turned on/off with the recording of the main binary data file (link to file properties). The ASCII files cannot be used by the M300 system for data playback. In order to output/create the ASCII data files one must have the raw M300 binary data file.

The ASCII File Output (asc.300) can have a Trigger entry to change the current trigger (See "Trigger" on page 19.). The default trigger is one second synchronous buffer.

The maximum number of characters per line is 8192. The M300 will give the user an error message if this maximum has been reached. Special care needs to be given to the first/tittle line. If a formula with an array of values is output, then each title string is output with an index. Minimizing the number of character in the name field will help reduce the final line size.

## Parameters

**Name**

The name is the identifier for the ASCII entry (see also, "Name" on page 537).

**Number**

A unique integer (Note that multiple ASCII outputs can have the same integer) used to identify this display to the M300. If the user has multiple ASCII outputs, they can assign different and/or the same integers to each output based on the intended usage of the M300 command manager. Note that these integers are unique to the output type only, they are not global to the M300.

**State**

The state can be used to turn on/off an ASCII entry (see also, "State" on page 548).

**Type**

The type parameter is used to select the different auto time options, please see the following table.

| Type | Description | Output |
|---|---|---|
| 0 | no time | |
| 1 | time | hh:mm:ss.hhhhh |
| 2 | seconds | s.hhhhh |
| 3 | seconds since midnight | sssss.hhhhh |
| 4 | date and time | yyyy/mm/dd,hh:mm:ss.hhhhh |

**UseASCIIRecord**

A '1' will allow this entry to be controlled by the ASCII record state/button. A '0' will ignore the ASCII record state/button. This allows selected ASCII entries to output regardless of ASCII record state. Serial output and network output most of the time don't need be control the ASCII record state.

**Delimiter/Termination**

The delimiter/termination parameter has an upper part for the delimiter and a lower part for the terminator. The delimiter/termination is optional.

The delimiter parameter can be used to in the asc.300 file for the valid count output feature. When the delimiter is a zero this is identical to not asking for a delimiter character. The typical values for the delimiter are 0x20 for a space character and 0x2C for a comma character.

The terminator character is used with the send command to transmit data to serial devices. If termination is 0, no termination character is added. Otherwise the termination will be sent. Since the termination is an integer number, it's possible to send out to terminating characters. For example, 0x0D0A would send carriage return and line feed in this order.

**MaxFreq**

The number of lines per sample per second, must be an integer value greater than or equal to 1. This is usually 1 or the number of samples to output.

**Title**

The ASCII file output can put a title entry at the start of the file for each column. The title doesn't repeat with every page, as the M300 system has no idea of how many lines there are per page. Use a '1' to put the title and a '0' for no title.

**CfgFile**

The file name for the ASCII configuration files, which is used to control the ASCII output. See "ASCII Output Table Configuration File, (*.asc)" on page 550.

**OutFile**

The file name for the ASCII output data. This can be set to '/dev/ser1' or '/dev/par1' to output to

# ASCII Output Table Configuration File, (*.asc)

## Overview

The ASCII Output Configuration Files are used to specify which data values to output. The user can format the data and specify the delimiter character as well as line termination.

## Parameters

**Name**

The name is the identifier for the ASCII configuration entry (see also, "Name" on page 537).

**Type**

The type is used to control the output placement of the samples in the ASCII/Binary output file. The user can specify 'RA' to have all samples in the same line. Use 'CA' instead to have each sample in a different line. The order of the character in the type is not important. The following are valid values for type.

| Type | Usage |
|---|---|
| Row | Row ASCII |
| Column | Column ASCII |
| R | Row ASCII |
| C | Column ASCII |
| RA | Row ASCII |
| CA | Column ASCII |
| RB | Row Binary |
| CB | Column Binary |
| V | Valid Counts |

**Index**

The index of the desired entry from the formula (0 for the first element). The formula must be an array of values and the index must be valid. For no index use a '-1'.

**Formula**

Formula number/name for the data to be output (see also, "Formula" on page 538).

**Format**

This parameter is used to format the output data. Since we use the 'printf' function from C to output the data this follows that standard. Special care must be taken not to use an invalid format for the type of the formula. Invalid format fields can cause the M300 to crash or at the very least provide data that doesn't make sense (see also, "Format" on page 539).

On very common mistake is to have a formula of float type and then specify the string format option "%s". This can sometimes cause the system to crash, depending on the actual data that is in memory (from where the string output will occur).

**Delimiter**

This parameter is can be used to change whether or not a delimiter character is necessary for the output data, so it acts as a state for the delimiter. This parameter is optional, the default is 0 for no delimiter. If the user specifies a 1 then the delimiter character is used allow the data value to generate the desired output.

## Example

```
; ASCII Configuration Files
; *.asc
; Name       Type      Index Formula Format
"Date"       CA       -1    F1      "%s"
"Time"       CA       -1    F0      ",%s"
;
"FsspSum"    RA       -1    F8020   ",%0.0f"
"OapSum"     RA       -1    F9020   ",%0.0f"
"FsspLwc"    RA       -1    F8051   ",%0.3f"
"OapLwc"     RA       -1    F9051   ",%0.3f"
"FsspMvd"    RA       -1    F8060   ",%0.1f"
"OapMvd"     RA       -1    F9060   ",%0.1f"
"FsspMd"     RA       -1    F8061   ",%0.1f"
"OapMd"      RA       -1    F9061   ",%0.1f"
"LwcFsp"     RA       -1    F8052   ",%0.2f"
"LwcOap"     RA       -1    F9052   ",%0.2f"
"LwcJnw"     RA       -1    F52305  ",%0.2f"
"LwcRmt"     RA       -1    F53007  ",%0.2f"
"LwcNevL"    RA       -1    F53121  ",%0.2f"
"LwcNevT"    RA       -1    F53131  ",%0.2f"
"LwcKng"     RA       -1    F53207  ",%0.2f"
"RmtRaw"     RA       -1    F53001  ",%0.4f"
"FSP VAC"    RA       -1    F8033   ",%0.3f"
"FSP RNG"    RA       -1    F8002   ",%1d"
;
"CR"         CA       -1    F65533  "%c"
"LF"         CA       -1    F65532  "%c"
```

# Board Table, (brd.300)

## Overview

The Board Table is used to keep all the information necessary to configure the board entries. In the M300 system all interfaces must be specified in the Board Table. There are different board types for each different board. For most boards the user can have more than one board per system of the same type at different addresses. In some cases there can only be one board of a given type (for example, System Board).

The Board Table and the board configuration files are totally configured from the M300 system using the Board Setup Dialog. This documentation is provided as an extra reference, in case a manual override is necessary.

Each line in the board table must have a valid board type, followed by the '=' and the board file name. Spaces can be used between the board type and the '=' sign and the '=' and the board file name.

The number of boards in a project is only limited by how many physical boards you can place in the system. The M300 software places no limit on the number of boards.

It is not possible to have more than one board at the same board address.

**The order of the acquisition events must follow the same order as board table entries. Comments will be saved out of sequence otherwise.**

## Parameters

### Type

The type for the board. The following is a list of the valid board types. Boards marked with a '*' are not supported at this time in the M300. Support for these boards will be added as needed. Each project must have at least a System board.

| Type | Description |
|---|---|
| 1D | SEA 1D Basic Interface |
| 1D256 | SEA 1D Advanced Interface (1D256) |
| 2DGREY | SEA 2D Grey Interface |
| 2DMONO | SEA 2D Mono Interface |
| AIMMS | AIMMS/ADP Interface |
| ARINC429 | SEA ARINC429 Interface |
| ARINC561 | SEA ARINC561 Interface |

**Board Types**

| Type | Description |
|---|---|
| ATDAQ141X | ATDAQ1411, ATDAQ1412 Interfaces |
| BALLARD708 | BALLARD 708 Interface |
| CAMAC1D | SEA CAMAC 1D Interface |
| CAMAC1D256 | SEA CAMAC 1D256 Interface |
| CAMACANALOG | CAMAC Analog Interface |
| CAS | CAS Interface (part of CAPS) |
| CASDPOL | CASDPOL Interface |
| CASPBP | CAS Particle-by-Particle Interface (part of CAPS) |
| CIP | CIP (part of CAPS) |
| CIPGS | CIPGS (part of CAPS) |
| CYCTM | Cyber Counter Interface |
| CYDDA | Cyber D/A Interface |
| CYDIO24 | Cyber 24-bit Digital I/O Interface |
| CYPDISO | Cyber Isolated Relay Interface |
| DT2801* | DT2801 Interface |
| DT2817 | DT2817 Interface |
| DT2827* | DT2827 Interface |
| FALCON* | SEA FALCON Interface |
| GPIBPCII | GPIB PCII Interface |
| GPS | SEA LORAN/GPS Interface |
| HAIL* | SEA HAIL Interface |
| HVPS* | SEA HVPS Interface |
| NETWORK | Network/Socket Connection |
| NRCFS* | NRC Frame Synch |
| NRCPT* | NRC Parallel Transfer |
| PCIDAC | PCIDAC Interface |
| PIRAQ | PIRAQ Interface |

**Board Types (Continued)**

| Type | Description |
|---|---|
| PIRAQ2 | PIRAQ-2 Interface |
| PMF | PMF Interface |
| PMS1058B* | PMS 1058B Interface |
| RTI802 | RTI802 Interface |
| SBUS | SEA BUS Interface |
| SEACOUNTER24 | SEA COUNTER 24 Interface |
| SEADA | SEA D/A Interface |
| SERIAL | SEA SERIAL Interface |
| SERIALPORT | Serial Port |
| SPP | SPP |
| SYSTEM | SEA SYSTEM Interface |
| VAX* | SEA VAX Interface |

**Board Types (Continued)**

**BoardFile**

File name of the board configuration file. See "Board Table Configuration File, (*.brd)" on page 555.

## Example

```
; Version = 2
; brd.300
; type     =    boardFile
System     =    System.brd
Arinc429   =    Arinc429.brd
Piraq      =    PiraqA.brd
Piraq      =    PiraqB.brd
Piraq      =    PiraqFwd.brd
Piraq2     =    Piraq2.brd
CYDIO24    =    Cydio_psi.brd
CYDIO24    =    Cydio_temp.brd
```

# Board Table Configuration File, (*.brd)

## Overview

The user can have as many board configuration files as necessary. Each file holds the configuration parameters for the specific board type as per the values specified in the M300 Board Setup Dialog. Please check the M300 User's Guide, Board Setup section.

## Files

The following examples list sample settings for the board configuration files.

## Example

```
; 1D
Address                 = 0x0700
State                   = 1
NonAcqState             = 0
Command                 = 0

; 1D256
Address                 = 0x3300
State                   = 1
NonAcqState             = 0
Command                 = 0
Range                   = 0
Gain                    = 1
SizeChannels            = 16
StrobeChannels          = 0
SourceFrequency         = 4
DivideFactor            = 1

; 2DGrey
Address                 = 0x2300
State                   = 1
NonAcqState             = 0
DMA                     = 7
IRQ                     = 10
BitShift                = 2

; 2DMono
Address                 = 0x1700
State                   = 1
NonAcqState             = 0
DMA                     = 5
BitShift                = 4

; AIMMS
```

```
Port                    = /dev/ser2
Address                 = 0xF002
State                   = 1
Baud                    = 19200
Data                    = 8
Stop                    = 1
Parity                  = 0
PurgeTime               = 5000
LowTempThreshold        = 100.0

; Arinc429
Address                 = 0x6700
State                   = 1
NonAcqState             = 0
ReceiveSpeed            = 1
TransmitSpeed           = 0

; Arinc561
Address                 = 0x6700
State                   = 1
NonAcqState             = 0

; Atdaq141x
Address                 = 0x220
State                   = 1
NonAcqState             = 0
Type                    = 2
Mode                    = 1
Range                   = -10.000000
VoltageReference        = -10.000000

; Ballard708
Address = 0xE100
State = 1
NonAcqState = 0

; CAMAC1D
Address                 = 0x0309
State                   = 1
NonAcqState             = 0
Command                 = 0
Slot                    = 1

; CAMAC1D256
Address                 = 0x0309
State                   = 1
NonAcqState             = 0
Command                 = 0
SizeChannels            = 16
StrobeChannels          = 0
SourceFrequency         = 4
DivideFactor            = 1
Slot                    = 1
```

```
; CAMACANALOG
Address                 = 0x0309
State                   = 1
NonAcqState             = 0
Slot                    = 1

; CAS
Address                 = 0x7300
State                   = 1
NonAcqState             = 0
Port                    =
Baud                    = 57600
Data                    = 8
Stop                    = 1
Parity                  = 0
Channels                = 30
ClockDivider            = 1
LaserAttenuation        = 22
FileName                = /test/cas/cas30.def

;
; CASDPOL
Address = 0x7700
State = 1
NonAcqState = 0
PBP = 1
Port =
Baud = 460800
Data = 8
Stop = 1
Parity = 0
Channels = 30
ADCThreshold = 83
NumberPBPPackets = 500
NumberRawChannels = 4
Controller1 = 32769
PBPSizeThreshold = 82
ControllerPBP = 30983
ControllerRaw = 2824
DAControllers = casdpolda.def
UpperBinSizes = /test/casdpol/casdpolubs.def

; CASPBP
Address                 = 0x7300
State                   = 1
NonAcqState             = 0
Port                    =
Baud                    = 57600
Data                    = 8
Stop                    = 1
Parity                  = 0

; CIP
Address                 = 0xF001
```

```
State                   = 1
NonAcqState             = 0
DMA                     = 5
Port                    = /dev/ser1
Baud                    = 57600
Data                    = 8
Stop                    = 1
Parity                  = 0
TASSource               = 1
SmallParticleReject     = 0
NValue                  = 1
DepthOfFieldReject      = 0
EndDiodeReject          = 0
UseRealParticleWidth    = 1
RecoveryCoefficient     = 255

; CIPGS
Address = 0x7302
State = 1
NonAcqState = 0
DMA = 7
Port =
Baud = 57600
Data = 8
Stop = 1
Parity = 0
TASSource = 1
DepthOfFieldReject = 0
ParticleSizeMeasurement = 1
RecoveryCoefficient = 255
MinimumThreshold = 87
MiddleThreshold = 75
MaximumThreshold = 50
LevelHistogram = 1
LevelImage = 0

; CYCTM
Address = 0x310
State = 1
NonAcqState = 0

; CYDDA
Address                 = 0x0320
State                   = 1
NonAcqState             = 0

; CYDIO24
Address                 = 0x0228
State                   = 1
NonAcqState             = 0
PortA                   = 0
PortB                   = 1
PortC                   = 0
```

```
; CYPDISO
Address                 = 0x228
State                   = 1
NonAcqState             = 0

; DT2817
Address                 = 0x0228
State                   = 1
NonAcqState             = 0
Port0                   = 0
Port1                   = 1
Port2                   = 0
Port3                   = 0

; GPIBPCII
Address                 = 0x2E1
State                   = 1
NonAcqState             = 0
DeviceAddress           = 1
InFile                  = filter.cap
OutFile                 = filter.eee
LogFile                 =

; GPS
Address                 = 0x4700
State                   = 1
NonAcqState             = 0
Baud                    = 4800
Data                    = 8
Stop                    = 1
Parity                  = 0

; NETWORK
Address                 = 0xFF00
State                   = 1
NonAcqState             = 0
Protocol                = tcp
IP                      = 192.9.200.200
Port                    = pos-primary
Direction               = 1

; PCIDAC
Address = 0xE200
State = 1
NonAcqState = 0

; Piraq
Address                 = 0x0340
State                   = 1
NonAcqState             = 0
IRQ                     = 10
Memory                  = 0x0D0000
DSP                     = /test/piraq/piraq.dsp
TimingMode              = 1
```

```
Delay                   = 5
Gates                   = 200
Hits                    = 400
GateWidth               = 4
PulseWidth              = 2
PulseRepetitionTime     = 64000
Watchdog                = 0
FirstGate               = 0
PhaseCorrect            = 0
ClutterFilter           = 0
TimeSeries              = 0
TimeSeriesGate          = 50
ScanRate                = 0
PulseRate               = 0
IndexOfRefraction       = 0

; Piraq2
Address                 = 0xE000
State                   = 1
NonAcqState             = 0
DSP                     = /m300/stagger.dsp
TimingMode              = 2
CoarseDelay             = 12
FineDelay               = 0
Gates                   = 512
Hits                    = 500
GateWidth               = 4
PulseWidth              = 4
PulseRepetitionTime     = 7680
Watchdog                = 0
FirstGate               = 1
PhaseCorrect            = 1
ClutterFilter           = 0
TimeSeries              = 1
TimeSeriesGate          = 6
ScanRate                = 0
PulseRate               = 0
IndexOfRefraction       = 0
AFCGain                 = 1e-007
AFCHigh                 = 9381.4
AFCLow                  = -10
LockTime                = 5
VelocitySign            = 1
PulseRepetitionTime2    = 9600
Trigger                 = 3
TestPulseWidth          = 1
TestPulseDelay          = 100
TestPulse               = 0
Frequency               = 60
RadioFrequencyTrigger   = 0
RadioFrequencySwitch    = 1
Stagger                 = 1
Sync                    = 1
TransmitPulseDelay      = 0
```

```
; PMF
Address = 0xE300
State = 1
NonAcqState = 0

; RTI802
Address                 = 0x0340
State                   = 1
NonAcqState             = 0

; SBUS
Address                 = 0x4F00
State                   = 1
NonAcqState             = 0

; SEADA
Address                 = 0x4300
State                   = 1
NonAcqState             = 0

;
; SeaCounter24
Address = 0x5F00
State = 1
NonAcqState = 1
Mode = 1
ThresholdVoltage = 2.5

; Serial
Address                 = 0x5700
State                   = 1
NonAcqState             = 0
Baud                    = 9600
Data                    = 8
Stop                    = 1
Parity                  = 0

; SerialPort
Port                    = /dev/ser1
State                   = 1
Baud                    = 19200
Data                    = 8
Stop                    = 1
Parity                  = 0

; SPP
Address                 = 0xF003
State                   = 1
NonAcqState             = 0
Port                    = /dev/ser3
Baud                    = 57600
Data                    = 8
Stop                    = 1
```

```
Parity                  = 0
Type                    = 0
TriggerThreshold        = 10
TransitReject           = 1
Channels                = 30
DOFRejection            = 1
Range                   = 0
AvgTransitRejection     = 5
AvgTransitTimeAccept    = 95
DivisorFlag             = 0
CountMethod             = 0
FileName                = /m300/chan30.def

; System
Address                 = 0x0300
State                   = 1
NonAcqState             = 0
IRQ                     = 3
Frequency               = 100
```

# Button Table, (btn.300)

## Overview

The Button table allows for the placement of buttons in any window. The buttons can be used to control as well as to monitor. The user can select from a few different button types. Buttons can be grouped together so that only one is selected from the group. Indicator buttons can be used to monitor results.

For indicator buttons to work, the user must turn on the status type and provide an indicator formula.

Not all features are available with all button types.

This table has a set of basic parameters used. There are some parameters which can be optional. Check the example at the end of this section for the current information on the different valid input lines.

Trigger lines are allowed in Button table to select different triggers.

## Parameters

**Name**

The identifier for the Button entry (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple Buttons, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300.

**Window**

The window where the Button entry will be placed (see also, "Window" on page 537).

**Type**

The button type selector. This is a bit-by-bit field. Bit zero is used to select a Toggle button (0x0001). Bit one is used to select a Repeat button (0x0002). Bit two is used to select a Status button (0x0004). The user can pick the bits for the desired features.

**State**

The state controls whether the button is displayed. If state is zero, the Button entry is not visible, otherwise it is visible.

**Font**

The font used to display the button label.

**Color, OnColor, OffColor**

The user can select the text color, on color and off color properties for the button (see also, "Color" on page 538).

**OnLabel, OffLabel**

Text entries which will be used for the on and off labels for the button.

**Style**

The style of the button.

| Button | Style |
|---|---|
| Standard | 0 |
| Diamond | 1 |
| Box | 2 |
| Round | 3 |
| Tick | 4 |
| Check | 5 |
| Indicator | 6 |

**Button Style**

**Group**

This field can be used to group together several button. This would create a group of button which would act as mutually exclusive. This means only one button can be selected at a time. For example, you have a group of four buttons to control the range on a FSSP probe. Then you can set the group to “FSSP” for all these buttons to indicate that they are mutually exclusive. Make sure you don’t specify other groups with the same name or you might get a undesired effect. For independent buttons just use “” for the group name (in other words an empty string).

**Flag**

The flag is used to select the initial state of a button. To select the on state use a one. To select the off state use a zero. For buttons in a group, only one button can be selected on at a time.

**X, Y, W, H**

The basic coordinates and dimensions for the button. This along with the window name selects where the button will be placed. The button should be placed in an area which will not interfere with other objects in the M300 system.

**Formula**

The formula whose value will be changed by the button click. If a Toggle button is used, then the formula value is overridden. If the button is not a Toggle type, then the formula value is momentarily

changed and then the formula is set to auto compute. This formula can be set to -1 to disable the control formula, this is usually required when sending out a command is desired instead.

**IndFormula**

The indicator formula. This formula is used to provide feedback on indicator/status buttons. For example, in the case of the FSSP range. When the user changes the range to the probe, it takes a few seconds for the probe to change range. Having a way to control the range and checking the feed back from the instrument allows us to be certain that the requested action actually took place.

**OnValue, OffValue**

The desired on and off values for the button. These fields take on several different meanings. They can specify an actual values which will be used to set the formula when the button is clicked. However, if the formula parameter is set to -1, then these fields can be setup for a command string which will be passed to the command manager. Multiple commands can be entered by separating them with the special '@' character.

## Example

```
; btn.300
; Version = 1
; name number window type state font color onColor offColor onLabel offLabel style group flag x y w h formula
; name number window type state font color onColor offColor onLabel offLabel style group flag x y w h formula indFormula
; name number window type state font color onColor offColor onLabel offLabel style group flag x y w h formula onValue offValue
; name number window type state font color onColor offColor onLabel offLabel style group flag x y w h formula indFormula onValue offValue
;
Trigger = "Sync" 1 None "Never" Never None
;
"Range0" 0 FSSPERtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "0" "0" 6 "FsspERRange" 1 480 60 60 18 F-1 F4052 "cmd1d FsspER 0" "0"
"Range1" 1 FSSPERtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "1" "1" 6 "FsspERRange" 0 480 82 60 18 F-1 F4052 "cmd1d FsspER 1" "1"
"Range2" 2 FSSPERtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "2" "2" 6 "FsspERRange" 0 480 104 60 18 F-1 F4052 "cmd1d FsspER 2" "2"
"Range3" 3 FSSPERtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "3" "3" 6 "FsspERRange" 0 480 126 60 18 F-1 F4052 "cmd1d FsspER 3" "3"
"Auto" 4 FSSPERtxt 0 1 Courier16b 0xFFFFFF 0x0000FF 0x0000FF "Auto" "Auto" 0 "" 1 480 148 60 18 F-1 F-1 "cmd1d FsspER auto@fml 4054 0 auto" "0"
;
; name number window type state font color onColor offColor onLabel offLabel style group flag x y w h formula indFormula onValue offValue
"PumpControl" 4 PCASPtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "ON" "OFF" 6 "" 1 460 60 60 60 F-1 F2052 "cmd1d Pcasp 0" "cmd1d Pcasp 1"
;
"2DGCEdgeReject" 5 2DGCtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "ON" "OFF" 6 "" 1 580 80 60 60 F-1 F6140 "cmd2g 2DGC 0x01 or" "cmd2g 2DGC 0x01 and"
"2DGCDOFOff" 6 2DGCtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "Off" "0" 6 "2DGCDOF" 0 580 180 60 18 F-1 F6141 "cmd2g 2DGC 0x60 and" "0"
"2DGCDOFMid" 7 2DGCtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "Mid" "1" 6 "2DGCDOF" 1 580 205 60 18 F-1 F6141 "cmd2g 2DGC 0x40 and@cmd2g 2DGC 0x20 or" "1"
"2DGCDOFMax" 8 2DGCtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "Max" "3" 6 "2DGCDOF" 0 580 230 60 18 F-1 F6141 "cmd2g 2DGC 0x60 or" "3"
;
"2DGPEdgeReject" 5 2DGPtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "ON" "OFF" 6 "" 1 580 80 60 60 F-1 F7140 "cmd2g 2DGP 0x01 or" "cmd2g 2DGP 0x01 and"
"2DGPDOFOff" 6 2DGPtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "Off" "0" 6 "2DGPDOF" 0 580 180 60 18 F-1 F7141 "cmd2g 2DGP 0x60 and" "0"
"2DGPDOFMid" 7 2DGPtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "Mid" "1" 6 "2DGPDOF" 1 580 205 60 18 F-1 F7141 "cmd2g 2DGP 0x40 and@cmd2g 2DGP 0x20 or" "1"
"2DGPDOFMax" 8 2DGPtxt 1 1 Courier16b 0xFFFFFF 0x00A000 0xFF0000 "Max" "3" 6 "2DGPDOF" 0 580 230 60 18 F-1 F7141 "cmd2g 2DGP 0x60 or" "3"
```

# Buffer Table, (buf.300)

## Overview

The Buffer table is used to configure the parameters for each buffer entry as well as the acquisition events that go with each buffer. In the M200 system, there was no buffer table. This is a new concept to the M300 system. In the M200 system, the user specified the buffer number for each acquisition entry in the Acquisition table and System table.

In the M300 system the buffer table is used to specify which acquisition events belong to each buffer and the order of acquisition for all the events.

For synchronous buffers, higher frequency events should/must be placed first.

For asynchronous buffers, the master event has to be first followed by the slave events.

The Buffer Table has two different types of entries, the buffer entry and the acquisition event entry. The first entry must be a buffer entry for buffer 0, followed by the synchronous acquisition events from the acquisition table. The order of the events is the same order that will be used when acquiring the data. After all acquisition events for buffer 0, we can have another entry for the next buffer, if necessary. Each time the acquisition events for the buffer must be listed, following the buffer entry.

When the user uses the M300 system to create/edit board and acquisition event entries, the buffers will automatically be generated. The Buffer Setup Dialog allows the modification of the default M300 generated setting to be altered.

***WARNING:*** Manual m**odifying this table by adding or removing an acquisition entry also requires modification of the 'acq.300' table.**

## Parameters

The following fields are part of the buffer configuration entry.

**Number**

The buffer number (0, 1, 2,..., 255). The first buffer number (0), is reserved for the 1 second synchronous buffer. The 1 hz synchronous buffer must always be present. Buffer number 251 is reserved for the Command data. Buffer number 252 is reserved for the Error data. Buffer number 254 is reserved for Secondary Acquisition Data. Buffer number 255 is reserved for Tables (this buffer doesn't show up in the buffer table). Other buffer numbers can be used for asynchronous or other synchronous buffers.

**Asynchronous Buffer Frequency / Synchronous Life**

In the case of synchronous buffers, the buffer frequency is the buffer life, in other words when the buffer expires. For the 1 second synchronous buffer, the buffer frequency is set for the same value as the maximum system frequency specified in the System Board entry.

For example, to have a maximum frequency rate of 100 hz, both the buffer frequency and the System Board frequency should be set at 100.

In the case of asynchronous buffers, the buffer frequency controls the buffer rearm frequency. This means the buffer rearm frequency controls the maximum possible number of buffers you can get each second. In the M200 this was usually specified in PARA 3 of the acquisition table for the master acquisition event.

For imaging type asynchronous acquisition buffers (2D Grey, 2D Mono, CIP, etc), this means the buffer rearm frequency controls the maximum number of buffers you can get each second. Assuming the system frequency is 100 hz and the buffer rearm frequency is 25 hz, this would give at most 25 buffers per second.

For other asynchronous acquisition types, such as serial data, this means the buffer rearm frequency controls the maximum number of buffers you can get per second. If the data rate of the serial data is 10 hz, then you should typically set the buffer rearm frequency to at least twice as much as the desired data rate. So this would require a 20 hz buffer rearm frequency.

The system frequency set in the board table, sets the maximum acquisition frequency. In the case of the buffer rearm frequency, the system frequency limits the number of possible buffer rearm frequencies.

For example, a system frequency of 100 hz, buffer frequencies of 20 hz and 25 hz are possible.

For system frequency of 160 hz, buffer frequencies of 20 hz is possible, but 25 hz would not be possible. For system frequency of 160 hz, the next possible frequency after 20 hz is 32 hz.

**Count**

For each buffer number there are several buffer available in shared memory. The buffer count controls the number of buffers that will be made available in shared memory. The user doesn't have to play with these values much, but 4 to 8 buffers of each type should be sufficient.

**Record**

The M300 system gives us the capability to record or not, buffers from a particular buffer number.

**Broadcast**

Just like with the record option, the broadcast option allows the user control over which buffer types will be broadcast over UDP.

**Sync**

This parameter tell the buffer manager whether a buffer is synchronous (1) or asynchronous (0). Buffer zero must be synchronous.

**Board**

The buffer manager needs to have a link to the board entry. For synchronous buffers there is no particular board associated with the buffer entry, so there is no board entry (address 0x0000). In the case of asynchronous buffers the board entry for the master acquisition event is required.

The following field(s) are part of the acquisition event entry.

**Event**

This is the name for the acquisition event, from the acquisition table. The name is case sensitive and it must match the acquisition event entry name character for character.

## Example

```
; Version = 1
; buf.300
; Number Freq  Count  Record  Broadcast  Sync  Board
0        100     8      1        1            1     0x0000
"Latitude"
"Longitude"
"True Heading"
"Magnetic Heading"
"Pitch"
"Roll"
"Altitude"
1         100    8      1        1           0      PiraqA
"I and Q A"
"Config A"
"Status A"
2         100    8       1       1           0      PiraqB
"I and Q B"
"Config B"
"Status B"
3         100    8       1       1           0      PiraqFwd
"I and Q Fwd"
"Config Fwd"
"Status Fwd"
"FwdAntAz"
"FwdAntTilt"
4 100 8 1 1 0 CCN
"CCNData"
5 100 8 1 1 0 Hygrometer
"HygrometerData"
6 100 8 1 1 0 PitotPress
"PitotPressData"
7 100 8 1 1 0 StaticPress
"StaticPressData"
8 100 8 1 1 0 ProPak
"ProPakData"
251 0 8 1 1 0 0x0000
252 0 8 1 1 0 0x0000
254 0 8 1 1 1 0x0000
```

# Project Configuration Table, (cfg.300)

## Overview

This table is used to store global M300 project information. This information is saved every time the M300 exits. Note that this data is global for a project only. When the user switches to another project, the M300 will load in new data from the cfg.300 file associated with the newly selected project.

## Parameters

**Console**

Stores the console number (0-8) the user was viewing at the time the M300 last exited.

**CommandHistory***0..n*

Stores the commands that have been entered into the Command Manager command prompt. The number of entries saved is dependent on the History Size set in the Properties dialog of the M300. When the Project is reloaded, this data will be used to populate the Command Manager History List on the M30 Main Window.

## Example

```
; Version = 1
; cfg.300
CommandHistory0 = cmd2g 2DGP 0x21
CommandHistory1 = cmd2g 2DGC 0
CommandHistory2 = cmd2g 2DGC 0x41
CommandHistory3 = cmd2g 2DGC 0x21
CommandHistory4 = cmd2g 2DGC 0x60
CommandHistory5 = cmd2g 2DGC 0x61
CommandHistory6 = cmd2g 2DGC 0x60 or
CommandHistory7 = cmd2g 2DGC 0x01 or
CommandHistory8 = cmd2g 2DGC 0x0
CommandHistory9 = cmd2g 2DGC 0x01
CommandHistory10 = cmd2g 2DGC 0xFE and
CommandHistory11 = cmd2g 2DGC 0x00
CommandHistory12 = cmd2g 2DGC 0x01 OR
CommandHistory13 = cmd2g 2DGC 0x01 and
CommandHistory14 = cmd2g 2DGC 0x61 or
CommandHistory15 = cmd2g 2DGC 0x60 and
CommandHistory16 = wnd 0 jpg
CommandHistory17 = wnd 1 jpg
CommandHistory18 = cmd1d FsspER auto
CommandHistory19 = clear error
Console = 0
```

# Cloud Image Probe Grey Scale Display Table, (cgs.300)

## Overview

This display is used to display particle image data of CIPGS Image type. The user can select a color for the images (minimum, middle and maximum shadows). This display has the capability of hashing out old images via a user selectable age limit. The image data is identified via the board address for the CIPGS Image data. The image may be scaled. The CIPGS Image display has an age counter, which keeps track of how many seconds have elapsed since the last valid image display. The number of displays (buffers) that the user can see per second can be controlled via the primary trigger frequency for the window.

The CIPGS Image display is made up of several strips. Each strip, displays as many slices as possible. Because of image compression, there are a variable number of slices in the CIPGS Image display. Slices are 128 bits wide (16 bytes). Each pixels is 2 bits wide. So there are 64 pixels per slice.

The CIPGS display looks for the first full image to display. The user can turn on/off the timebar.

## Parameters

**Name**

The name is the identifier for the CIPGS Image entry. For example, “CIPGS" (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple CIPGS displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a CIPGS display does also, then a command set up to change the color of the CIPGS display will not affect the HVPS display.

**Window**

Each entry in the CIPGS Image display table need to belong to a window. This parameter is the name of the window where the CIPGS Image display will be done. The type of the window must be CIPGS Image display. For example, “cipgs” (see also, "Window" on page 537).

**ColorMinimum, ColorMiddle, ColorMaximum**

The CIPGS probe has 2 bits per pixel. Each bit is assigned an intensity level (minimum, middle and maximum). The CIPGS display allows the user to pick whatever colors he desires for each intensity level. The color for each intensity level maybe the same as another or even the background color. The use of the background color for the lowest intensity level could be useful in removing undesirable noise from the display (such as splash, stuck diodes, etc.) (see also, "Color" on page 538).

**Address**

The address selects the CIPGS Image data. The user doesn't need to know the tag number for the CIPGS Image data, just the address of the board where the CIPGS Image data is coming from. Valid addresses are 0x7300, 0x7302, 0x7700 and 0x7702 (other addresses possible). When the user changes the address, the primary trigger for the window also gets changed. This allows the display to run only when there is CIPGS Image data available (see also, "Address" on page 538).

**Timebars**

Along with each CIPGS particle there are also two slices containing the timing data for the particle. The CIPGS display can display these in the same color as the particle (use a 1) or as the background (use a 0). When the time bars are shown with the background color, they are not 'removed' from the display. This leaves the particles in the same position, regardless of whether or not the time bars are shown.

**Scale**

The user can scale the CIPGS Image particles by a desired value. The default scale value is 1. The larger the scale value, the larger the particles will appear on the display. Larger particles may mean less particles per display window.

**AgeLimit**

The ageLimit is used to hash out an old display. Once the current CIPGS Image display is older than the specified ageLimit, then the display gets hashed out as an indication of old data. This parameter is specified in seconds. The window must have the secondary trigger set to expire once per second on the synchronous buffer.

**Probe**

This is the probe name from the probe table (prb.300). This is used to associate a probe table entry with a CIPGS Image display entry (see also, "Probe" on page 538).

## Example

```
; Version = 1
; cgs.300
; name  number  window colorMin colorMid colorMax  board timebars scale ageLimit probe
"CIPGS"      0   CIPGS      red    blue     green 0x7300       1     1      60   cipgs
```

# Cloud Image Probe Display Table, (cip.300)

## Overview

This display is used to display particle image data of CIP Image type. The user can select a color for the images. This display has the capability of hashing out old images via a user selectable age limit. The image data is identified via the board address for the CIP Image data. The image may be scaled. The CIP Image display has an age counter, which keeps track of how many seconds have elapsed since the last valid image display. The number of displays (buffers) that the user can see per second can be controlled via the primary trigger frequency for the window.

The CIP Image display is made up of several strips. Each strip, displays as many slices as possible. Because of image compression, there are a variable number of slices in the CIP Image display. Slices are 64 bits/pixels wide (8 bytes). We have a one bit/pixel mapping for the CIP Image display.

## Parameters

**Name**

The name is the identifier for the CIP Image entry. For example, “CIP" (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple CIP displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a CIP display does also, then a command set up to change the color of the CIP display will not affect the HVPS display.

**Window**

Each entry in the CIP Image display table need to belong to a window. This parameter is the name of the window where the CIP Image display will be done. The type of the window must be CIP Image display. For example, “cip” (see also, "Window" on page 537).

**Color**

The CIP Image probe has 1 bit per pixel. Each bit can be on or off. The CIP Image display allows the user to pick whatever color he desires for the images (see also, "Color" on page 538).

**Address**

The address selects the CIP Image data. The user doesn’t need to know the tag number for the CIP Image data, just the address of the board where the CIP Image data is coming from. Valid addresses are 0x7300, 0x7302, 0x7700 and 0x7702 (other addresses possible). When the user changes the address, the primary trigger for the window also gets changed. This allows the display to run only when there is CIP Image data available (see also, "Address" on page 538).

**Timebars**

Along with each CIP particle there are also two slices containing the timing data for the particle. The CIP display can display these in the same color as the particle (use a 1) or as the background (use a 0). When the time bars are shown with the background color, they are not 'removed' from the display. This leaves the particles in the same position, regardless of whether or not the time bars are shown.

**Scale**

The user can scale the CIP Image particles by a desired value. The default scale value is 1. The larger the scale value, the larger the particles will appear on the display. Larger particles may mean less particles per display window.

**AgeLimit**

The ageLimit is used to hash out an old display. Once the current CIP Image display is older than the specified ageLimit, then the display gets hashed out as an indication of old data. This parameter is specified in seconds. The window must have the secondary trigger set to expire once per second on the synchronous buffer.

**Probe**

This is the probe name from the probe table (prb.300). This is used to associate a probe table entry with a CIP Image display entry (see also, "Probe" on page 538).

## Example

```
; Version = 3
; cip.300
; name    number window  color    board    timebars scale ageLimit probe
"CIP"         0    CIP   Red    0x7300           1     1      60    cip
```

# Command Table, (cmd.300)

## Overview

The Command Table is used to execute any M300 commands that have been defined by the user. The user can choose from a large array of commands and define specific keyboard mappings for them. For instance, a user can define the key mapping of **Ctrl** and **F1**, that will change the FSSP range simply by pressing **Ctrl+F1**. Note that all commands are stored in a data file if acquiring at the time the command is input. The commands are stored under reserved tag 65532, buffer number 251.

## Parameters

### Comment

This entry describes what the particular defined function(s) is/are for. Comments must be started with a semi-colon ";" (see also, "Comments" on page 537). There are certain limitations on the use of comments and where they may be put. For instance, they may only be placed immediately before function key entries and command block separators '#' (see below).

### Function Key Entry

This is the key mapping that the M300 will look for to execute the associated command block entry. The function can have the following syntax:

```
.[C][A][S] functionKey
```

All function key definitions should be prefixed with a period '.' prior to any modifiers and/or function keys The `[C]`, `[A]` and `[S]` arguments are for the Control, Alt, and Shift keys respectively. These modifier keys are optional. Some, all or none may be used. Valid `functionKey` values are keys such as F1, F2, ... , F12. This is to minimize keystrokes interfering with pre-existing QNX shortcut definitions. Using combinations of the modifiers described with the function keys F1, F2,..., F12 gives a total of 96 different possible command definitions. This should satisfy any user requirements.

### Command Entry/Block

The command block is a series of commands to be executed when the preceding defined function key entry is pressed. The command block may contain a single command entry, or a series of commands that will be carried out in sequence. The number of command entries in a single command block can contain 1 to *n* commands, where *n* is limited only by memory constraints. The command blocks are carried out in the following manner:

```
.[C][A][S] functionKey
command1  [arg0  arg1  ... argm]
command2  [arg0  arg1  ... argm]
   .       .     .         .
   .       .     .         .
```

$command_n$ [$arg_0$ $arg_1$ ... $arg_m$]

The commands can contain from 0 to *m* additional arguments that will then be used in the command execution, where *m* is the maximum number of arguments accepted for a particular function (See "Commands" ). Some arguments are optional, while others are required. See the function specification for details.

For each defined function definition, more than one command block may be defined. The pound sign '#' is used to delimit one command block from another. The command blocks are executed based on the number of times the defined function keys have been pressed. The first time the function keys are pressed, the first command block will execute; the second time it is pressed, the second command block will execute and so forth. Again, the command blocks may rang from 0 to *n*, where *n* is limited only by memory constraints. When the defined function key(s) is/are pressed *n*+1 times, the first command block will execute, and the cycle will start over. All function calls are not case-sensitive. The syntax for the command blocks is as follows:

```
.[C][A][S] functionKey
command block1
#
command block2
#
   .       .
   .       .
#
command blockn
```

**Commands**

For a list of valid command manager commands, See "Command Manager Reference" .

## Example

```
; Version = 1
; cmd.300
.C F2
cmd1d fssp 1
#
cmd1d fssp 2
. F1
scn HOME
. F2
scn 0
#
scn 1
#
scn 2
#
scn 0
. F10
stop
. F11
start
```

# Formula Table, (fml.300)

## Overview

The Formula Table is used to perform any necessary computations. Each line in the formula table is a formula entry (variable). Each formula entry is made up of a name/number for identification and can have units. The result field determines both the type and number of elements that make up the formula value/data.

The computations are done at the end of the formula entry, using Reverse Polish Notation (RPN). For additional information on RPN please reference our Reverse Polish Notation on page 1. There is an extensive set of general purpose functions that can be used to do predefined tasks. The parameters to the functions are passed after the function name and are inside parentheses '()'. It is important to note that no spaces are allowed between the function name and the opening parentheses.

In addition to the general purpose functions there are also a large number of Math functions that can be used in the computations. Math functions are used differently than regular functions because they operate "on the stack". To use a Math function you simply use the math function name when you want to use it. Math functions can be used any where, except as a parameter to a function. This differs from Regular functions because they must have the parameters passed inside the parentheses and cannot be nested inside other functions.

The user does not have to define a formula number/name before it's used. This has several advantages as you will see later on in the documentation.

Each computation is made up from several factors/tokens. Spaces are used to delimit tokens and comas ',' are used to separate each function parameter. Factors can be an integer number, floating point number, a string, a math function, a regular function, probe entry, etc.

The Formula Table (fml.300) can have a Trigger entry to change the current trigger (see also, "Trigger" in M300 Miscellaneous Reference on page 19). The default trigger is one second synchronous buffer. This means all computations are performed from top to bottom by default once a second.

## Parameters

### Name

Name for the formula entry which is used to identify the formula (sort of a variable name) (see also, "Name" in M300 Miscellaneous Reference on page 537).

### Units

The units field is used to specify formula units. If the formula has no units then this field can be left as a blank, "". The formula units are useful in several ways. First it helps the user keep track and document the units for each formula. In the M200 this was normally part of the description field, which no longer exists. The units are also used by the different display type to automatically put units up. This saves the user the hassle of creating and displaying units for every desired entry.

The units field can have special ASCII characters such as °, µ, ². This is the only field that allows these characters.

**Number**

The formula number is used to identify a formula (link to formula). When the user assigns formula numbers, they should not be assigned from 0, 1, etc.... Instead the user should pick a formula number based on the tag number or some other criteria that may be currently used in the project setup. For example, if converting from analog to volts for tag numbers 100 to 131, it would make sense for the user to pick formula numbers 100 to 131.

Computations should be grouped and assigned a particular range of formula numbers. This allows future expansion to the project, without generating formula numbers which may be hard to remember.

Don't use the same formula number for two different variables, this will likely not work. If you are not sure before adding a new formula, you should search the formula table.

It is a common mistake to have the same formula number with two different results spaces. This is ambiguous and cannot be allowed.

**Result**

The result field is used to pick the result type and space. The format for this parameter is a valid result type letter followed by square brackets or regular brackets with the number of elements inside the brackets. For example 'F[10]', would indicate a floating point formula (variable) with an array of 10 elements.

The minimum number of elements for a formula is 1. The maximum number of elements for a formula is 2500 (this can be changed, but we need to recompile the software).

The use of square brackets and regular brackets affects how the data is copied from the floating point stack to the result space for the formula.

() - **Sample match.**

When brackets are used for the result space the system uses the number of elements in the result space to figure out how many elements to copy from the floating point stack. The number of elements in the floating point stack and the number of elements in the result space is used to do linear interpolation when copying the results (link to linear interpolation here). For example, if there are 10 elements for a formula in the result space and only 5 elements for an item on the stack, the formula manager would fill the 10 elements in the result space. Since there are only 5 elements on the stack, the formula manager would use the same value twice for each element on the stack.

R[0] = S[0]
R[1] = S[0]
R[2] = S[1]
R[3] = S[1]
R[4] = S[2]
R[5] = S[2]
R[6] = S[3]
R[7] = S[3]
R[8] = S[4]
R[9] = S[4]

[] - **Regular copy.**

When square brackets are used for the result space the system uses the minimum number of elements from the stack size and the formula result. Then it copies this number of elements from the floating point stack to the result space. Using the same example from above in this case, only 5 elements would be copied from the stack to the result. The last 5 elements of the result space would be left alone, only the first 5 are used. We copy every element from the stack to the corresponding elements from the result.

R[0] = S[0]
R[1] = S[1]
R[2] = S[2]
R[3] = S[3]
R[4] = S[4]

The user must take special care not to define/use the same formula more than once using different types. You can use the same formula as many times as necessary, as long as the type is kept the same.

The following table shows the data types used by the Formula Manager for the M300 system.

| Type Letter | Size (Bytes) | Type | Min | Max |
|---|---|---|---|---|
| S, s | 1 | String | 0 | 255 |
| c | 1 | Unsigned Char | 0 | 255 |
| C | 1 | Signed Char | -127 | 128 |
| i | 2 | Unsigned Integer | 0 | 65535 |
| I | 2 | Signed Integer | -32768 | 32767 |
| l | 4 | Unsigned Long | 0 | 4294967295 |
| L | 4 | Signed Long | -2147483648 | 2147483647 |
| F, f | 4 | Float | 1.175494e-38 | 3.402823e38 |
| D, d | 8 | Double | 2.2250738585072e-308 | 1.79769313426232e308 |

**Formula Result Data Types**

**Computations**

Computations are made up of series of factors (operands, operators, functions and parameters). The formula manager breaks up the computations into two major types. Stack based operations and functions.

When a function is encountered, the formula manager calls the function with the given parameters. Once the function is done, the result is put on the stack. When a function is computed, the parameters for the function never go on the stack, just the result of the function.

Otherwise the formula manager pushes factors (operands) on the stack until it finds a math function (operator). The operator takes the appropriate number of arguments from the stack and replaces them with the result of the operation.

When all computations have been performed the formula manager takes the first element from the stack and copies it to the formula result. This way the value from the computation gets passed to a formula (variable). Please note that we use the first element from the stack and not the top of the stack. This means that it is up to the user to ensure that when all computations are done, there is only one element on the stack. This would make the top of the stack the first element on the stack.

There are several basic 'factors' that can make up a computation. Here is a list of the permitted factors for computations.

| Factor | Sample | Sample |
|---|---|---|
| Constant | PI | C |
| Formula | F100 | Fo.Temperature |
| Acquisition | A100 | Aq.Fssp |
| Long | 12 | 0x80 |
| Double | 1.25e-7 | 343.78 |
| String | "On" | "Ready" |
| Board | Bd.2dc | Bd.Fssp |
| Probe | Pr.Fssp | P0 |
| Lookup | Lo.RK24 | K1 |
| Radar | Ra.CPR | |
| Math | / | sin |
| Function | Sum(...) | Volts(...) |

**Factors**

The following table shows the constant type factors possible for the M300 system.

| Name | Description |
|---|---|
| 2PI | 2 * PI |
| C | Speed of light |
| COMMA | Comma, 44 or 0x2C |
| CR | Carriage Return, 13 or 0xD |
| DEGTORAD | 180 / PI |
| LF | Line Feed, 10 or 0xA |

**Constant Factors**

| Name | Description |
|---|---|
| ONE | 1 |
| PI | PI |
| PI/2 | PI / 2 |
| RADTODEG | PI / 180 |
| SPACE | Space, 32 or 0x20 |
| ZERO | 0 |

**Constant Factors**

We will explain the computations with some basic examples.

```
"DegToRad""" F5 F[1] 0.01745329252
"Latitude" "deg" F1100 F[1] Ins429Bin(A1000, 20, 180.0)
"Latitude" "rad" F1200 F[1] F1100 F5 *
```

The first formula (5 or F5) is set to 0.01745329252. This formula will be used to convert from degrees to radians. It has no units. Only one element of float type.

The next formula (1100 or F1100) is named “Latitude”. It has “deg” for units. Only one element of float type. In the computation we see a function, Ins429Bin() used. This function has three parameters. The first is a tag number (1000, or A1000) for the raw ARINC429 data for latitude. The second is parameter indicates to use 20 bits. The final parameter specifies a range of 180.0. This function will take the raw data, perform the necessary conversion using the number of bits and range and then puts the result on the stack. Finally the result from the stack will get copied to F1100 result space.

The final formula is (1200 or F1200). It has “rad” for units. Again only one element and the type is float. In the computations we see that F1100 (latitude in degrees) is pushed on the stack. Then, F5 (DegToRad) is pushed on the stack. Finally the ‘*’ operator is used to multiply the two together and place the result on the stack. The result from the stack will be copied to F1200 result space.

The main purpose of this example is to setup two formulas. One with latitude in degrees and the other with latitude in radians.

If we didn’t need latitude in degrees, we could have just used the following formula entry instead.

```
"Latitude""rad"F1200 F[1] Ins429Bin(A1000, 20, 180.0) F5 *
```

In this case, the result from the Ins429Bin() function goes on the stack. The value from F5 gets pushed on the stack. The multiply is done and the result goes on the stack. Finally the result from the stack gets copied to the F1200 result space.

If you are not familiar with the use of Triggers in the M300 system or have not read the section explaining their use we highly recommend reviewing our Trigger before continuing with this section. Next we will be explaining how the Trigger affects computations in the formula manager with some examples.

```
;
Trigger = "Sync" Once None "Never" Never None
"DegToRad" "" F5 F[1] 0.01745329252
"RadToDeg" "" F6 F[1] 57.29577951
```

This is a basic trigger used to setup formulas that should be initialized once. The primary trigger uses the 'Sync' buffer, 'Once'. The address is 'None', which will cause the address to be ignored. The secondary trigger is not used since we pick 'Never' for the trigger type. We don't even have to look any further at the trigger frequency or address.

```
   ;
Trigger = "Sync" 1 None "Never" Never None
"Time" "" F0 S[10] Time(A0)
"Date" "" F1 S[10] Date(A0)
```

This is the basic 1 hertz trigger on the 'Sync' buffer. The primary address is not a factor nor is the secondary trigger.

```
   ;
Trigger = "Piraq I, Q & P" 1 PiraqA "Never" Never None
"Timing Mode""" F2100 l[1] PqConfig(A2001, 0)
"Delay" "" F2101 l[1] PqConfig(A2001, 1)
"Gates" "" F2102 l[1] PqConfig(A2001, 2)
```

This trigger is a bit more interesting. The trigger type is on the "Piraq I, Q & P" data. We have a 1 hz frequency. The address/board is PiraqA. So this trigger will fire at most once per second on "Piraq I, Q & P" data for the PiraqA board only. The secondary trigger is not a factor.

Here is another example which show how to handle 2DC data in the M300 system.

```
;
Trigger = "Sync" 1 None "Never" Never None
"2DC Sizes" "" F1000 F[32] PrData(Pr.2dc, 2)
```

The primary trigger is set for 'Sync' buffer, 1 hz and ignore address. The secondary trigger is not a factor. This basically gets the 2DC Sizes once per second and updates for F1000.

```
;
Trigger = "Sync" 1 None "2D Image" 10 2dc
"2DC Counts" "" F1001 F[32] MoSums(Pr.2dc, Aq.2DCImage, 0x00, 1)
```

Here the trigger needed to be changed to handle the 2DC data into the MoSums function. The MoSums function needs a 1 hz 'Sync' trigger to produce a result once per second. It also needs the secondary trigger to fire for the '2D Image' for the '2dc' board. The secondary trigger in this case sets a maximum of 10 hz, in other words a maximum of 10 buffers per second will be analyzed by the MoSums function.

```
;
Trigger = "2D Image" 1 2dc "Never" Never None
"2DC Tas Factors""" F1002 l[1] Aq.2DCTasFactors
```

```
"2DC Elapsed Time""" F1003 l[1] Aq.2DCElapsedTime
"2DC Elapsed Tas""" F1004 l[1] Aq.2DCElapsedTas
"2DC Elapsed Shadow Or" "" F1005 i[1] Aq.2DCElapsedShadowOr
"2DC Tas Mul Fac" "" F1100 i[1] F1002
"2DC Tas Div Fac" "" F1101 i[1] F1002 16 >>
"2DC Tas Clock In" "MHZ" F1102 F[1] PrTasClockIn(Aq.2DCTasFactors)
"2DC Elapsed Time" "s" F1103 F[1] F1003 40000 /
"2DC Elapsed Tas" "s" F1104 F[1] F1004 100 * F1102 / 1.0e-6 *
```

This block of formulas gets data from the 2D buffer for the slave events. In this case we only need to set the primary trigger to fire at 1 hz for the '2D Image' data type for the '2dc' board. The secondary trigger is not necessary.

```
;
Trigger = "Sync" 1 None "Never" Never None
"Total Shadow Or" "" F1006 i[1] Aq.2DCShadowOr
"House Data" "" F1007 i[8] Aq.2DCHouseData
"Tas Clock Out" "MHZ" F1110 F[1] PrTasClockOut(Pr.2dc, Fo.TAS)
"2DC Tas Control" "" F1111 D[1] Co2DTas(Bd.2dc, F1110)
```

## Example

```
; Version = 1
; fml.300
```

```
; Trigger     Trigger1  Freq1  Board1      Trigger2  Freq2   Board2
Trigger   =   Sync         1     None        Never   Never   None
;
; Name                  Units  Formula   Result    Computations
"2DG Sizes"             ""     F1000     F[64]     PrData(Pr.2dg, 2)
"2DG Counts"          ""     F1001     F[64]     GrSums(Pr.2dg, Aq.2DG, 0x02, 1)
"2DG Tas Factors"       ""     F1002     l[1]      Aq.2DGTasFactors
"2DG SOI"               "s"    F1003     F[1]      GrData(Aq.2DGAdvanced, 4)
"2DG SOB"               "s"    F1004     F[1]      GrData(Aq.2DGAdvanced, 5)
"2DG Minimum"           ""     F1005     i[1]      GrData(Aq.2DGAdvanced, 9)
"2DG Middle"            ""     F1006     i[1]      GrData(Aq.2DGAdvanced, 10)
"2DG Maximum"           ""     F1007     i[1]      GrData(Aq.2DGAdvanced, 11)
"2DG Slice Count"       ""     F1008     i[1]      GrData(Aq.2DGAdvanced, 6)
"2DG RPC"               ""     F1010     i[1]      GrData(Aq.2DGAdvanced, 1
"2DG RawParticleCount" ""     F1011     i[1]      GrData(Aq.2DGAdvanced, 0)
"2DG Tas Mul Fac"       ""     F1100     i[1]      GrData(Aq.2DGAdvanced, 7)
"2DG Tas Div Fac"       ""     F1101     i[1]      GrData(Aq.2DGAdvanced, 8)
"2DG Tas Clock In"      "MHz"  F1102     F[1]      PrTasClockIn(Aq.2DGAdvanced)
"2DG Elapsed Time"      "s"    F1103     F[1]      F1003 40000 /
"2DG Elapsed Tas"       "s"    F1104     F[1]      F1004 256 * F1102 / 1.0e-6 *
```

# Formula Watch and Alter Table, (fwa.300)

## Overview

The M300 has a new window type used to watch and alter the any formula value. The user can have has many formula watch and alter windows as necessary. It's not necessary to edit this file, the M300 takes care of everything. The information here is provided for reference.

## Parameters

**Name**
The formula watch and alter entry name (see also, "Name" on page 537).

**Number**
A unique integer (Note that multiple windows can have the same integer) used to identify this display to the M300. If the user has multiple Formula windows, they can assign different and/or the same integers to each window based on the intended usage of the M300 command manager. Note that these integers are unique to the window type only, they are not global to the M300.

**Window**
Link to the formula watch and alter window. This window must be of the formula watch and alter type (see also, "Window" on page 537).

**Index**
This is the element index for formulas with arrays. Use -1 for no index. Index 0 is the first element.

**Formula**
The formula number to watch and/or alter (see also, "Formula" on page 538).

**Format**
This parameter is used to format the output data. Since we use the 'printf' function from C to output the data this follows that standard. Special care must be taken not to use an invalid format for

the type of the formula. Invalid format fields can cause the M300 to crash or at the very least provide data that doesn't make sense (see also, "Format" on page 539).

On very common mistake is to have a formula of float type and then specify the string format option "%s". This can sometimes cause the system to crash, depending on the actual data that is in memory (from where the string output will occur).

The following table shows typical format syntaxes with their appropriate data type usage.

| Syntax | Description | Formula/Result | Conversion | Result |
|---|---|---|---|---|
| %s | String | `S[5]` | `%s` | `Hello` |
| %d | Decimal (base 10) | `I[1]/100` | `%5d` | `00100` |
| %x/X | Hexadecimal (base 16) | `I[1]/254` | `%02x/%02X` | `fe/FE` |
| %f | Float (single precision) | `F[1]/120.322` | `%3.2f` | `120.32` |
| %g | Float (double precision) | `D[1]/210.119191` | `%3.5g` | `210.11919` |

**Format Syntax**

## Example

```
; Version = 2
; fwa.300
; name           number window index formula  [format]
"TAS(m/s)"            0   fwa0     -1      F1
"Mach"                1   fwa1     -1      F9
"PressureTrans0"      2   fwa1     -1    F110    %4.1f
"PressureTrans2"      2   fwa2     -1    F111
"Time"                3   fwa3     -1      F0
```

# Histogram Display Table, (his.300)

## Overview

A histogram is a bar graph that shows how many data values fall into a certain interval. The width of the bar represents the interval, while the height indicates the number of data items in that interval. The total sum of the data items for the given frequency is represented by the area bar. The accumulated sum of the data items is represented by the line bar.

The area bar is updated at the specified frequency. A new area bar is not available until the next frequency cycle has expired. The line bar is usually updated every second. The line bar reset after the frequency expires and a new area bar is generated.

The X and Y limits are user specified. The user can select the area bar color and line bar color.

## Parameters

**Name**

Name for histogram display entry (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple Histogram displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a Histogram display does also, then a command set up to change the color of the Histogram display will not affect the HVPS display.

**Window**

Name of the window where the Histogram display will be performed. This window must be of Histogram type (see also, "Window" on page 537).

**BarColor, LineColor**

Color for the area bar and line (see also, "Color" on page 538).

**Frequency**

The frequency for the summation. For example, if the user specifies a frequency of 0.2 hz, then the area bar will represent 5 seconds of summation. The line bar is reset every 5 seconds ().

**Formula**

Data source representing an array of values for the bar graph (see also, "Formula" on page 538).

## Example

```
; Version = 3
; his.300
; name number      window    barColor lineColor frequency formula
"CAS FWD"   0       wnd21   0x0000FF  0xFF0000    0.2       F1000
"CAS BAK"   1       wnd23   0x0000FF  0xFF0000    0.2       F1001
"CAS INT"   2       wnd24   0x0000FF  0xFF0000    0.2       F1002
"CIP"       2       wnd25   0x0000FF  0xFF0000    0.2       F2000
```

# Hodograph Display Table, (hod.300)

## Overview

The Hodograph shows wind speed and direction as a function of height over time.

## Parameters

**Name**
The identifier for the Hodograph entry (see also, "Name" on page 537).

**Number**
A unique integer used to identify this display to the M300. If the user has multiple Hodograph displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a Hodograph display does also, then a command set up to change the color of the Hodograph display will not affect the HVPS display.

**Window**

The window where the Hodograph display will be performed. This must be a Hodograph type window (see also, "Window" on page 537).

**Color**
The color for the Hodograph display (see also, "Color" on page 538).

**Rings**
The number of rings to be displayed in the Hodograph display.

**Range**
The speed limit value.

**Entries**
Total number of display entries/points to keep in memory.

**AltFormula, SpdFormula, DirFormula**
Data sources for altitude, wind speed (knots) and wind direction (radians) (see also, "Formula" on page 538).

## Example

```
; Version = 2
; hod.300
; Name Number Window Color Rings Range Entries AltFormula SpdFormula DirFormula
"hod"      0    hod   Red     5   170     500     F553       F5604      F5711
```

# High Speed Analog Display Table, (hsa.300)

## Overview

The High Speed Analog display produces a line graph of all the sample points for an array of analog channels. This graph is useful in observing quickly changing analog signals.

## Parameters

**Name**

The identifier for the High Speed Analog display (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple HSA displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a HSA display does also, then a command set up to change the color of the HSA display will not affect the HVPS display.

**Window**

Link to the window where the High Speed Analog display will be performed. This window must be a High Speed Analog window type (see also, "Window" on page 537).

**Color**

Color for display (see also, "Color" on page 538).

**Type**

This parameter is used to select what gets drawn once a new point is found.

| Name | Type |
|---|---|
| Point | 0 |
| Line | 1 |
| Bullet | 2 |
| Line with bullet | 3 |

**Type**

**Width**

Line width for the High Speed Analog entry. This is normally 1 pixel wide. Larger value for line width will require more drawing and slow down the display. You should keep this in mind when changing the line width.

**Decimate**

The decimate is used to control the number of data points to be displayed from the data source. A value of one selects every data point. A value of five selects every fifth data point. A value of 'n' selects every $n^{th}$ data point.

**Formula**

Data source for High Speed Analog display (see also, "Formula" on page 538).

**YMin, YMax**

The minimum and maximum values for the y-axis.

## Example

```
; Version = 2
; hsa.300
; name    number window    color type width decimate formula minimum maximum
"Analog 00"    0    hsa 0xFF0000    1    1        1     F100     -10      10
"Analog 01"    1    hsa 0x00A000    1    1        1     F101     -10      10
"Analog 02"    2    hsa 0x0000FF    1    1        1     F102     -10      10
"Analog 03"    3    hsa 0xFF00FF    1    1        1     F103     -10      10
```

# Height Time Indicator Display Table, (hti.300)

## Overview

The Height Time Indicator display is used to show Reflectivity or Doppler versus Altitude. This display support two beams form the current Altitude. This way, it can display one beam in the up position (a) and one beam in the down position (b) (or left/right if so desired). If used on the ground, then we can use the beam in the up position (a) and set the altitude to zero (or provide no altitude). The minimum and maximum altitudes are stored in the YMinimum and YMaximum fields for the window configuration file.

## Parameters

**Name**

The identifier for the Height Time Indicator entry (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple HTI displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a HTI display does also, then a command set up to change the color of the HTI display will not affect the HVPS display.

**Window**

Link to window where Height Time Indicator display will be performed. This must be a window with a Height Time Indicator window type (see also, "Window" on page 537).

**Scheme**

Name of the Radar Scheme to use. This is a link to the Scheme name from the Radar Table. (See “Radar Table, (rdr.300)” on page 625.).

**aPowerFormula, aRefFormula, aRangeFormula**

Data sources for beam in the up direction. This includes Power, Reflectivity and angle (see also, "Formula" on page 538).

**bPowerFormula, bRefFormula, bRangeFormula**

Data sources for beam in the down direction. This includes Power, Reflectivity and Range (see also, "Formula" on page 538).

**altitudeFormula**

Data source for Altitude (see also, "Formula" on page 538).

## Example

```
; Version = 3
; hti.300
; name number window scheme aPwrFml aRefFml aRngFml bPwrFml bRefFml bRngFml altFml
"HtiPwr"   0 HtiPwr "Power"  F2020   F2020   F2000   F3020   F3020   F3000   F4000
```

# High Volume Particle Spectrometer Display Table, (hvp.300)

## Overview

This display is used to show particle image data from the High Volume Precipitation Spectrometer (HVPS) probe. The user can select the color for the images. The display has the capability to hash out old images via a user selectable age limit parameter. The display has an age counter that keeps track of how old an image is. The number of displays (buffers) that the user can see per second can be controlled via the primary trigger frequency for the window. The secondary trigger must be set to one second Sync buffer.

The HVPS display is made up of several strips. Each strip, displays as many slices as possible. The number of slices per buffer is variable due to the HVPS data compression. Slices are 256 bits/pixels wide (32 bytes). We have a one bit/pixel mapping for the HVPS display.

## Parameters

**Name**

The identifier for the High Volume Precipitation Spectrometer entry (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple HVPS displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a HVPS display does also, then a command set up to change the color of the HVPS display will not affect the HVPS display.

**Window**

The window where the display will be performed for the High Volume Precipitation Spectrometer image data. This window must be of the High Volume Precipitation Spectrometer type (see also, "Window" on page 537).

**Color**

The image color (see also, "Color" on page 538).

**Address**

The address for the HVPS card. The M300 system will automatically find the tag number for the image data for the specified hardware (see also, "Address" on page 538).

**Timebars**

This parameter is used to turn on/off the time bars for the HVPS display.

**Scale**

The user can scale the HVPS particles by a desired value. The default scale value is 1. The larger the scale value, the larger the particles will appear on the display. Larger particles may mean less particles per display window.

**AgeLimit**

The ageLimit is used to hash out an old display. Once the current HVPS display is older than the specified ageLimit, then the display gets hashed out as an indication of old data. This parameter is specified in seconds. The window must have the secondary trigger set to expire once per second on the synchronous buffer.

**Probe**

This is the probe name from the probe table. (See “Probe Table, (prb.300)” on page 620.). This is used to associate a probe table entry with an HVPS display entry (see also, "Probe" on page 538).

## Example

```
; Version = 2
; hvp.300
; Name    Number Window    Color      Address Timebars Scale  AgeLimit    Probe
"HVPS"    2      HVPS      Green      0x1700  0        1      15          hvps
```

# Label Table, (lbl.300)

## Overview

The Label display allows a user to display a text/data label on any M300 window. This allows the user to greater control and flexibility of where and what to display.

This display is not as effective as a text/data label on text windows. It is therefore not recommend for the user to place a large number of these in the project.

Also care should be taken not to place labels where they might interfere with objects.

## Parameters

**Name**

The identifier for the Label (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. If the user has multiple Label displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300.

**Type**

Dictates what kind of Label entry it is. Valid values are 0 (label - default) and 1 (data). Ensure you are familiar with the structure of this table before changing this value. Certain grammars apply to type 0 and others to type 1. See the example section below to familiarize with which grammars go with type 0 and 1 respectively.

**Window**

The window where the Label entry will be displayed (see also, "Window" on page 537).

**State**

The state controls whether the display is updated. If state is zero, the label is not updated, otherwise, it is updated.

**X,Y**

The *x* and *y* coordinates where the Label is to be displayed. It is recommended that these values are not such that the label blocks data from being displayed.

**W, H**

The width and height of the Label entry.

**Font**

The font used to display the Label entry. It is recommended that a fixed font be used to display data labels. Otherwise we run into problems with ‘erasing' the previous data value.

**Color**

The desired color of the Label text (see also, "Color" on page 538).\

**ColorFill**

The desired background color of the Label (see also, "Color" on page 538).\

**HorizontalAlign**

Controls whether the text in the Label is left, center, or right justified. Valid values are (left - default), (center) and (right).

**Border**

A value that controls the size of the border the Label displays. A value of 0 suppresses the border, otherwise it is displayed using the specified size.

**Margin**

A value that controls the size of the margin the Label displays. A value of 0 suppresses the margin, otherwise it is displayed using the specified size.

**Index**

Selects a particular value from a formula array. For instance a formula value of 1000 and an index value of 7 would select element 7 of formula 1000. Note that this value cannot exceed the number of elements in the formula value given.

**Formula**

Used with data types (Type = 1), displays the value of the formula onto the Label display.

**Format**

Formats the output of the data from a formula. For more information on this topic, see "Format" in "Text Display Table, (txt.300)" on page 635 and see "Format" in "Standard conventions for parameters in setup project files." on page 539.

## Example

```
; Version = 1
; lbl.300
; Trigger = type1 frequency1 board1 type2 frequency2 board2
; name number window 0 x y
; name number window 0 state font color colorFill horizontalAlign border margin x y w h
; name number window 1 x y formula [format]
; name number window 1 state font color colorFIll horizontalAlign border margin x y index formula [format]
; name number window 1 state font color colorFIll horizontalAlign border margin x y w h index formula [format]
;
; name number window 0   x   y
"EMB"     1     2dg  0 100  45
; name number window 0 state font   color colorFill horizontalAlign border margin  x  y   w   h
"ERJ"     0     2dg  0   1  cour16b 0xFF 0xFF             left       0      0    10 20  50  20
; name number window   1   x  y  formula [format]
"Time"    8  analogtxt 1 280 45     F0       %s
; name number window 1 state font  color fIll  horAlgn  border  margin x    y  index formula [format]
"Date" 9      2dgtxt 1 1     cour8 0x0   0xFF  center   1       1      280 70    0    F6304      %s
; name number window 1 state font  color fIll  horAlgn  border  margin x   y   w   h    index formula [format]
"Date" 9      mvdsp 1  1   cour8  0x0  0xFF  center   1       1   280 70  10  100     0      F6304  %s
```

# List Table, (lst.300)

## Overview

The List display allows for the rapid display of data in one cohesive line of display. This data is displayed in the form of an output line which then repeats as new values are read from a file or acquired. This display is similar to the ASCII output feature. Instead of the data going to a file, the data goes to a list widget. This widget displays one line at a time. New data is placed at the bottom. There is a scroll bar to look back at data from a previous time.

## Parameters

**Name**
The identifier for the List entry (see also, "Name" on page 537).

**Number**
A unique integer (Note that multiple List displays can have the same integer) used to identify this display to the M300. If the user has multiple List displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300.

**Window**
The window where the List entry will be displayed (see also, "Window" on page 537).

**State**
The state controls whether the display is updated. If state is zero, the list entry is not updated; otherwise, it is updated.

**Font**
The font used to display the List entry (see also, "Font System" on page 36).

**Type**
The type parameter is used to select the different auto time options. Select 0 for no time. Select 1 for time in the form of hh:mm:ss.hhhhh. Select 2 for seconds in the form of (s.hhhhh). Select 2 for seconds since midnight in the form of sssss.hhhhh

**MaxFreq**
The number of lines per sample per second.

**Title**
A boolean value (0 or 1) that controls whether the title from the specified **CfgFile** is displayed or not. A value of 1 displays the title, while a 0 suppresses the title display.

**CfgFile**

The configuration file that contains the list elements/formulas to be displayed.

## Example

```
; Version = 1
; lst.300
; name  number  window   state  font   type maxFreq title cfgFile
"drop1"      1 drop1lst     1 cour10     0     1       1  drop1.lst
```

# List Table Configuration File, (*.lst)

## Overview

The List table configuration file provides the information the list display needs to display lines of data. Each entry is a line of data itself that is displayed using the specifications provided.

## Parameters

**Title**

A fixed-positioned line of text that appears at the top of the List display. Most commonly used as the header to describe each column of data. There is a 256 character limit for this string of text.

**Type**

Determines how the data is to be display, in either column or row format.

**Index**

The index of the desired entry from the formula (0 for the first element). For no index use a '-1'.

**Formula**

Formula number/name for the data to be output (see also, "Formula" on page 538).

**Format**

The format is optional. If the format is blank, the default format will be used based on the formula type (see also, "Format" on page 539).

## Example

```
; dropsonde.lst
; title                                                    type    index formula format
"MsgType Id SondeId Date  Time  Press  Temp  RH  WDir  WSpd   VVel"   column     -1   F1302   " %s"
```

# Lookup Table, (lup.300)

## Overview

The Lookup Table is a general purpose tool used to enter data from a table of data points. Each lookup entry has a name/number which can be used as the identifier. The user can retrieve data from the Lookup table via the Lookup() function in the formula table. The Lookup() function does a linear interpolation to get the correct desired value. The LookupGet() and LookupSet() functions also work on lookup data.

## Parameters

**Name**
The identifier for the Lookup table entry (see also, "Name" on page 537).

**Number**
The identifier for the Lookup entry is a number. This number is used in the formula table to refer to the desired lookup entry.

**Rows**
The number of rows in the Lookup data file.

**Columns**
The number of columns in the Lookup data file.

**Filename**
The name of the Lookup data file. These file names have the '*.lup' extension (See "Lookup Table, (lup.300)" on page 604.)

## Example

```
; Version 2
; lup.300
; Name    Number  Rows Columns  FileName
"LWCREF "      0     21       2  lwc_ref.lup
```

# Lookup File, (*.lup)

## Overview

Each lookup file is made up of any number of rows and columns of values. This is similar to a spreadsheet with rows and columns of values. The number of rows and columns must match the corresponding lookup file.

The lookup entry can be used with the traditional linear interpolation lookup function. It can also be used to get and set calibration data into the M300 system.

## Parameters

Rows and columns of values representing the desired data. See example bellow.

## Example

```
; Lookup Files
; lwc_ref.lup
;X          Y
2.92        30000
6.64        25000
4.14        22000
4.51        20000
4.91        18000
5.31        16000
5.76        14000
6.23        12000
6.74        10000
7.28         8000
7.86         6000
8.47         4000
8.79         3000
9.12         2000
9.45         1000
9.81            0
```

# Moving Air Mass Display Table, (mam.300)

## Overview

The Moving Air Mass display provides a way for the user to keep track of a moving air mass parcel. This display shows a relative position between the aircraft and the moving air mass. The display provides range and bearing to the target. The aircraft heading and track are shown on this display. The user can also select the desired number of range rings.

## Parameters

**Name**
The identifier for the Moving Air Mass entry (see also, "Name" on page 537).

**Number**
A unique integer (Note that multiple Moving Air Mass displays can have the same integer) used to identify this display to the M300. If the user has multiple Moving Air Mass displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a Moving Air Mass display does also, then a command set up to change the color of the Moving Air Mass display will not affect the HVPS display.

**Window**
Link to window where Moving Air Mass display will be performed. This window must be a Moving Air Mass window type (see also, "Window" on page 537).

**Color**
The color for the display (see also, "Color" on page 538).

**Font**
The font name to be used for the range and bearing text display, which is part of the Moving Air Mass display.

**Rings**
The number of rings to display.

**Range**
The range limit in nautical miles.

**Entries**
The number of data points to keep in memory.

**latFormula, lonFormula, spdFormula, dirFormula, hdgFormula**

This is the data sources for the Moving Air Mass display. This includes formulas for aircraft latitude, longitude and heading. Also moving air mass speed and direction. The latitude, longitude, heading and direction must be in radians. The speed must be in knots (see also, "Formula" on page 538).

## Example

```
; Version = 2
; mam.300
; Name Number Window Color  Font  Rings Range Entries latFml lonFml spdFml dirFml hdgFml
"mam"     0      mam  0x0 cour12    5    50     600    F2001  F2002  F5604  F5711  F4005
```

# Probe Distribution Display Table, (pdi.300)

## Overview

The Probe Distribution display gives a line graph of the probe size vs. Y. The Y data can represent counts, concentrations, sums, volumes, areas, etc. The probe distribution display will only display data which is larger than the Y minimum limit specified. This creates gaps on the display for bins whose data value is less than the Y minimum limit specified.

## Parameters

**Name**
The identifier for the Probe Distribution entry (see also, "Name" on page 537).

**Number**
A unique integer (Note that multiple PDI displays can have the same integer) used to identify this display to the M300. If the user has multiple PDI displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a PDI display does also, then a command set up to change the color of the PDI display will not affect the HVPS display.

**Window**
Link to window where Probe Distribution display will be performed. This window must be of the Probe Distribution type (see also, "Window" on page 537).

**Color**
Color used for display (see also, "Color" on page 538).

**Type**
This field is used to specify the plot type for the Probe Distribution entry

| Name | Type |
|---|---|
| Horizontal Bars | 1 |
| Step (Horizontal and Vertical Bars) | 2 |

**Type**

**Width**
Line width for the X vs. Y entry. This is normally 1 pixel wide. Larger value for line width will require more drawing and slow down the display. You should keep this in mind when changing the line width.

**Probe**

This is the probe name from the probe table (See “Probe Distribution Display Table, (pdi.300)” on page 608.). This is used to associate a probe table entry with an Probe Distribution display entry.

The Probe Distribution display uses the data from the probe channel file to find out and display the necessary probe bins (x-axis data). The probe entry has an updated probe range value. This value is used when applicable automatically (see also, "Probe" on page 538).

**yFormula**

Data source for Probe Distribution display. This can normally be counts, concentrations, etc. (see also, "Formula" on page 538).

**xMin, xMax**

The minimum and maximum limits for the x-axis.

**yMin, yMax**

The minimum and maximum limits for the y-axis. Please remember that no data will be displayed if the data source value is less than the minimum value.

## Example

```
; Version = 2
; pdi.300
; Name       Number Window  Color  Type Width  Probe     yFormula xMin  xMax   yMin   yMax
"f100/um11"     0   wnd230 0xFF00    1    2   fssp-075   F5177    1   10000  1e-010  100
"2dg/um14"      1   wnd230 0x00FF    1    2     2dg      F9177    1   10000  1e-010  100
"f100/um31"     2   wnd231 0xFFAA    1    2   fssp-075   F5177    1   10000  1e-010  100
"2dg/um34"      3   wnd231 0xFF11    1    2     2dg      F9177    1   10000  1e-010  100
```

# Target Position Display Table, (pos.300)

## Overview

The target position display is used to show a map with the aircraft position and track. The display plots aircraft position relative to a center latitude and longitude. If desired wind speed and direction can be used to display wind barbs along the flight path. Data labels along the flight path are also possible. Markers can be display for given latitude and longitude.

There are four types of entries in this file. The map entries, the position entries, data entries and marker entries.

Map entries are the primary entry. They are used to specify the window, the center of latitude/longitude, setup the north/south and east/west miles as well as provide the map file.

The position entries are the secondary entries. They are attached to the preceding map entry and will be displayed in the window specified in the map entry.

The data entries are used to display data labels along the flight path. They must follow a position entry. It's possible to have multiple data entries per position entry.

The marker entries are used to display markers on the map for a given position entry. It's possible to have multiple marker entries if necessary.

In the M200 system, there was only one entry with all the necessary information. Sometimes some of the information had to be repeated/omitted several times for different position entries. This help define the separation of map, position and data entries for the M300 system.

## Map Entry

Map entries are used to specify the display window, center of latitude/longitude, north/south and east/west miles as well as the map file. The map entry is responsible for drawing the map in the window. Only one map entry per window is allowed. Multiple map entries/windows are possible.

**Name**

The name used to identify the map entry (see also, "Name" on page 537).

**Number**

A unique integer used to identify this display to the M300. The number can be used for commands.

**State**

The state variable is used to control when a map entry is visible and active (1) or not visible but active (0). (see also, "State" on page 538).

**Window**

The window where the target position map display will be performed (see also, "Window" on page 537).

**cLat, cLon**
These fields specify the center latitude and center longitude of the position display. These fields are to be specified in degrees and fractions of a degree. The M300 will convert these into radians internally.

**nsMiles, ewMiles**
These fields specify the distance north/south and east/west of the center point. These values are specified in nautical miles.

**Scale**
The scale to be used for map display. Normally the scale is a 1.0. The scale can be less than 1.0 or greater than 1.0. The scale must be greater than 0.0.

**AutoPercent**
Controls whether or not the position display will redraw the map to fit the current position in the plot. It is based on the width and height of the window (1-100). Typical values are 1 to 10 percent. A negative value disables this feature (i.e. -1).

**FileName**
The map file name '*.tgt'. If no map is desired, then provide an empty map file. The map file supports the M200 format and there are some new additional commands for the M300 system.

## Position Entry

The position entry shows aircraft track along latitude and longitude position. Aircraft heading, wind barbs are possible using the position entry. There must be a map entry in the file prior to position entries. Multiple position entries are possible per map entry.

**Name**
The identifier for the position entry (see also, "Name" on page 537).

**Number**
A unique number to identify the position entry. This can be used for commands.

**State**
The state variable is used to control when a position entry is visible and active (1) or not visible but active (0). Data for a position entry is always updated, even when the display is not visible (see also, "State" on page 538).

**Wind Barb State**
The wind barb state variable is used to control when a wind barb is visible and active (1) or not visible but active (0). Data for a wind barb entry is always updated, even when the display is not visible (see also, "State" on page 538).

**Color**
Color for display object.

**Type**

The type of object that will be displayed.

| | 16 x 16 Aircraft | 24 x 24 Aircraft |
|---|---|---|
| **Name** | **Type** | **Type** |
| None | -1 | -1 |
| Plane | 0 | 10 |
| Cross | 1 | 11 |
| Point | 2 | 12 |
| X | 3 | 13 |
| Diamond | 4 | 14 |
| Box | 5 | 15 |
| Triangle | 6 | 16 |
| Circle | 7 | 17 |

**Type**

Note: If you wish to display the map without a position marker use '-1' for type. This will generate an entry to display the track with no position marker.

**Width**

The track width in pixels ranging from 1 to 3. Smaller track width results in better image performance.

**Entries**

The number of entries kept in memory for each position entry. If the number of entries is '0', then the Position plot has no memory.

**Frequency**

The wind barb display frequency. The wind barb frequency should be set for an optimal value so that the wind bards are not displayed on top of each other.

**latFormula, lonFormula, hdgFormula, wspFormula, wdrFormula**

Data source for position display. This includes latitude, longitude, heading, wind speed and direction. The latitude, longitude, heading and wind direction are in radians. The wind speed is in knots.

The aircraft heading is used to display an aircraft marker with the aircraft symbol rotating to the correct heading. If this feature is not desired, leave the heading at -1 to indicate no heading formula.

The wind speed and direction are used to display wind bards. If the user doesn't wish to have wind barbs in the Position display, then leave this at -1 for no wind speed and direction (see also, "Formula" on page 538).

## Data Entry

The data entry shows data values along the flight track. There must be a position entry in the file prior to data entries. Multiple data entries per position entry are possible.

**Name**
The identifier for the data entry (see also, "Name" on page 537).

**Number**
A unique number to identify the data entry. This can be useful for necessary commands.

**State**
The state variable is used to control when a data entry is visible and active (1) or not visible but active (0). Data for a data entry is always updated, even when the display is not visible (see also, "State" on page 538).

**Frequency**
The data display frequency. The data frequency should be set for an optimal value so that the data values are not displayed on top of each other.

**Font**
The font used to display the data entry. It is recommended that a fixed font be used to display data labels. Otherwise we run into problems with 'erasing' the previous data value.

**Color**
The data display allows the user to pick whatever color he desires for the data entry (see also, "Color" on page 538).

**XOffset, YOffset**
The x and y offset for the data entry display in pixels. This can be used to offset the data entry of the flight track so as to maximize readability.

**Formula**
Formula link for data to be shown (see also, "Formula" on page 538).

**Format**
The format for the data output (see also, "Format" on page 539).

## Marker Entry

The marker entry can show different types of markers for a latitude and longitude pair. There can zero, one or many latitude and longitude pairs. The marker formula indicates how many markers are

passed each time, this number will vary all the time. Multiple marker entries per position entry are possible.

**Name**
The identifier for the marker entry (see also, "Name" on page 537).

**Number**
A unique number to identify the marker entry. This can be useful for necessary commands.

**State**
The state variable is used to control when a marker entry is visible and active (1) or not visible but active (0). Marker entry is always updated, even when the display is not visible (see also, "State" on page 538).

**Color**
The marker display allows the user to pick whatever color he desires for the data entry (see also, "Color" on page 538).

**Type**
The type of object that will be displayed. Since the system can handle a large number of markers on three marker types are available. Use 1 for a cross, 2 for a point and 3 for a X maker type,

**Entries**
The number of entries kept in memory for each marker entry. If the number of entries is '0', then the marker entry has no memory.

**latFormula, lonFormula, marFormula**
Formula link for latitude, longitude and marker to be shown (see also, "Formula" on page 538). The marker formula type must be a long type.

## Example

```
; Version = 5
;
;
"Map1"   0    1  FlightTrack 57 -111.4 80        80  1      1 oilsand.tgt
;
"AircraftHdg1" 10 1 0 red 0 1 1 1 F1338 F1339 F1318 F-1 F-1
;
;
"OverallTrack1" 20 1 1 green -1 1 1999 0.0166 F1338 F1339 F1318 F1351 F1352
;
"NO" 30 0 0.066 cour12b brown 10 10 F10364 "%3.1f"
"NOy" 31 0 0.066 cour12b brown 10 10 F10384 "%3.1f"
"SO2" 32 0 0.066 cour12b brown 10 10 F10324 "%3.1f"
"O3" 33 0 0.066 cour12b 0x777777 10 10 F10304 "%3.1f"
```

# Map File, (*.tgt)

## Overview

Target area files contain information for rendering maps on a window. Aircraft position and track can be displayed on top of these maps. Target files contain a set of commands that let the system know where to draw lines, circles, place text or markers. It should be noted that depending on the window size and the actual area of interest in the map, as well as the display pitch for the screen in question, that some shapes may not look as desired. For example a square may look like a rectangle or vice versa. The following pages contain a description of the valid commands and parameters associated with these.

## Commands/Parameters

**Comment**

A comment is started with a ';' in any line of the target map file.

**Circle**

A circle command places a circle on a target area display. The user must specify the latitude and longitude in degrees and minutes as well as the diameter in nautical miles.

```
c  latdeg latmin londeg lonmin diameter
```

**Color**

This command specifies the color (bit map) to be used while drawing objects. Once this command is read, the color is changed. The color can be a standard RGB hexadecimal value or one of the following predefined values (WHITE, RED, GREEN, BLUE, BLACK, DGRAY, MGRAY, GRAY, YELLOW, MAGENTA, CYAN, DGREEN, DCYAN, DBLUE, BROWN, PURPLE).

```
b  color
```

**Point**

A point command places one pixel point on a target area display. The point will be placed at the latitude and longitude specified. The point can be used to mark a particular interest area, or simply display points.

```
p  latdeg latmin  londeg lonmin
```

**Line**

A line command places a line on a target area. Lines can be used to draw roads, rivers and other boundaries. The first line command is used as a move to a point, rather than an actual line draw. The following line commands will draw lines from the previous position to the position specified by the current line command. Once a line (or several) have been drawn that make up, for example a road, an end of line command must be entered. Other lines sequences (roads) may be drawn by repeating this process.

```
l  latdeg latmin  londeg lonmin
```

**End Line**

This command must be used to finish a particular line drawing. For example, after a road is drawn and the user wishes to start drawing a river somewhere else, the pen must be lifted, and another line drawing started. This is the use of the end of line command.

```
e
```

**Marker**

A marker command places a marker on a target area at the specified latitude and longitude. The marker type can be either PLANE, CROSS, X, DIAMOND, BOX and TRIANGLE. The M300 supports the M200 markers (CROSS - 1, and POINT - 2) as well as some new marker types.

```
m  latdeg latmin  londeg lonmin  markertype
```

**Text**

A text command displays text (*text*) on a target map at the specified latitude and longitude. The actual text to be displayed follows the latitude and longitude position. The text can be placed in one of eight directions, at a particular x and y offset (*xoff* and *yoff*) in pixels from the latitude (*latdeg)* and longitude (*longdeg)* point. The font (*font*) used is eight pixel by default, but it can be changed (this parameter is optional).

If the text has spaces, it must be enclosed inside double quotes (for example, “sample text”). The maximum numbers of characters for the text is 15 characters.

The M300 supports two text commands. The ‘T' is the new M300 command for text. The font for the text is picked with the font command. The ‘t' is the M200 command for text.

```
T  latdeg latmin londeg lonmin  xoff  yoff  text
t  latdeg latmin londeg lonmin  text  dir  xoff  yoff  [font]
```

**Dir**

Direction for the text displayed

- 0 - right to left
- 1 - right to left 45 degrees up
- 2 - up
- 3 - left to right 45 degrees up
- 4 - left to right
- 5 - left to right 45 degrees down
- 6 - down

•7 - right to left 45 degrees down

**Font**

The font name (*fontName*) used can be changed. This must be a valid name for a Photon font. Once the font is changed, it stays in effect until a new font command is issued.

```
f fontName
```

**Parallels of Latitude**

Parallels of latitude on a target map can be drawn using a simple one line command. The latitude lines will be placed on multiples of the latitude specified in degrees (*degrees*) and minutes (*minutes*). The lines may or may not be labeled with the corresponding value. The label refresh (*labelReresh*) controls the frequency at witch lines will be labeled. The line style (*lineStyle*) can either be solid (*lineStyle* = 0) or dashed (*lineStyle* = 1).

```
a  degrees minutes  labelRefresh  lineStyle
```

**Longitude Lines**

Longitude lines on a target map can be drawn using a simple one line command. The longitude lines will be placed on multiples of the longitude specified in degrees (*degrees*) and minutes (*minutes*). The lines may or may not be labeled with the corresponding value. The label refresh (*labelReresh*)

controls the frequency at witch lines will be labeled. The line style (*lineStyle*) can either be solid (*lineStyle* = 0) or dashed (*lineStyle* = 1).

```
o  degrees minutes  labelRefresh  lineStyle
```

## Example

```
; Map File
; ct.tgt
;b Color
b BLACK
; l  latdeg     latmin    londeg    lonmin
l   41         25.728    -71       48.330
; m  latdeg     latmin    londeg    lonmin  markertype
m   42         26.735    -70       48.225  CROSS
```

# Plan Position Indicator Table, (ppi.300)

## Overview

Plan Position Indicator Display shows a radar beam (reflectivity) moving from -60 to +60 degrees from forward looking center position.

This display basically duplicates a pilots aircraft forward looking radar display.

## Parameters

**Name**

The identifier for the Plan Position Indicator entry (see also, "Name" on page 537).

**Number**

A unique integer (Note that multiple PPI displays can have the same integer) used to identify this display to the M300. If the user has multiple PPI displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a PPI display does also, then a command set up to change the color of the PPI display will not affect the HVPS display.

**Window**

Link to window where the Plan Position Indicator display will be performed. This window must be of the Plan Position Indicator type (see also, "Window" on page 537).

**Scheme**

Name of the Radar Scheme to use. This is a link to the Scheme name from the Radar Table (See “Radar Table, (rdr.300)” on page 625.).

**Radar**

Name of the Radar entry to use from the Radar Table (See “Radar Table, (rdr.300)” on page 625.).

**powerFormula, refFormula, rangeFormula**

Data source for power (dbm), reflectivity (dBz) and range (nmi) (see also, "Formula" on page 538).

**gatesFormula, hitsFormula, angleFormula, tiltFormula**

Data source for gates, hits, angle (rad) and tilt (deg) (see also, "Formula" on page 538).

## Example

```
; Version = 2
; ppi.300
; Name Num Window Scheme Radar pwrFml refFml rngFml gtsFml htsFml aglFml tltFml
"PPI"  0   PPI    "air"  "CPR" F4020  F4030  F4000  F4102  F4103  F1511  F1513
```

# Probe Table, (prb.300)

## Overview

The Probe Table is used to enter properties about the different probes used by the data system. Each probe has a basic set of parameters including range, number of channels and size. In addition to the basic probe parameters, each probe also has a table of information for each channel. This table is provided via the Probe Channel Configuration file.

There are two ways probe data is used. The first is with the PrData() function in the Formula Table. This function allows access to the different probe data specified via both files. The second use for the probe data is direct via a probe entry in one of the tables. For example in the case of the 2D Mono Display Table (2dm.300) the user can specify a probe entry for a 2D Mono probe directly in the 2D Mono display entry.

## Parameters

**Name**

Probe name used to identify this entry. Names are easier to remember that probe numbers. They are both valid ways to link up to the probe entry. The probe number is the traditional way, so we had to provide support for it.

The probe name can be used in the Formula Table (fml.300) as well as other tables as a link to the Probe Table entry (see also, "Name" on page 537).

**Number**

Probe number used to identify this entry.

**Ranges**

This field defines the number of ranges for a particular probe. Most commonly this field is one except for probes which have more than one range. Such is the case of the FSSP, which has four ranges.

**Channels**

The number of channels for a particular probe. Common values for this field are 15, 32 and 64. This number reflects the number of channels available on the 1D type probes. Imaging probes allow the user to set the number of channels to a value different than the number of pixels recorded.

**Size**

This field is the size of the Imaging probes (2D, 2D Grey, etc.) pixel and is used by spectrum routines. Non imaging, 1D type probes should have this value set to zero.

**Sync**

This is the sync pattern or marker for the Imaging probes. Actual only the 2D Mono has a sync pattern used by the display and spectrum routines. This sync pattern can be ignored by using '0xFF'. Zero should be used for all other probes.

**TasLimit**

This is the TAS limit for the probe. This will be used by the functions which are used to generate/control the TAS to the probes. For non imaging probes use zero.

**FileName**

The name of the Probe Channel file (See "Probe Channel File, (*.prb)" on page 622.). See next section for a description of the parameters for these files. These probe channel files used to have a '*.chn' extension in the M200 system. We have just renamed the file extension to keep the M300 naming convention standard. The format is the same as M200 system.

## Example

```
; Version = 1
; prb.300
; Name    Number RangeMax Channels Size    Sync tasLimit    FileName
fssp100     0       4        15       0    0        0    fssp100.prb
2dc         1       1        32       25   0xff   125        2dc.prb
pcasp100    3       1        15       0    0        0    pcasp100.prb
2dp         2       1        32       200  0xff   200        2dp.prb
```

# Probe Channel File, (*.prb)

## Overview

For each probe entry there is a Probe Channel configuration file. The file provides information about the channel sizes and other pre-computed parameters used for computations.

## Parameters

**Number**
This field is used to define the channel number of the data following on the same line.

**Minimum**
This field is used to specify the minimum channel size (diameter).

**Maximum**
This field is used to specify the maximum channel size (diameter).

**Middle**
This field is used to specify the 'average' channel size (diameter). This 'average' can be a linear or geometric 'average' depending on whether the channels are arranged in a linear or logarithmic fashion.

**dD**
This field is used to specify the linear difference of the maximum and minimum channel sizes.

$$dD = Maximum - Minimum$$

**dLogD**
This field is used to specify the logarithmic difference of the maximum and minimum channel sizes.

$$dLogD = \log(Maximum) - \log(Minimum)$$

**Area**
This field is used to specify the area of a circle of average diameter.

$$AREA = \pi \cdot \left(\frac{DIAMETER}{2}\right)^2 = \frac{\pi}{4} \cdot MIDDLE^2$$

**Volume**
This field is used to specify the volume of a sphere of average diameter.

$$VOLUME = \frac{4}{3} \cdot \pi \cdot \left(\frac{DIAMETER}{2}\right)^3 = \frac{\pi}{6} \cdot MIDDLE^3$$

**sampleArea**

This field is used to specify the two dimensional sample area (typically in mm2) of the sensing area of the probe. This sample area is multiplied by the distance propagated during the sampling interval (TAS * 'Delta Time'), to generate the sample volume.

The user should include all conversion constants needed to provide the final result in the desired units. By convention the TAS will always be in 'm/s' and 'Delta Time' in seconds.

## Example

```
; Probe Channel Files
; fssp100.prb
; Number Minimum Maximum Middle dD     dlogD   Area       Volume     SampleArea
1        12.50   37.50   25.00  25.000 0.4771  490.9       8181       48.848
2        37.50   62.50   50.00  25.000 0.2218  1963        6.545e+004 48.895
3        62.50   87.50   75.00  25.000 0.1461  4418        2.209e+005 48.943
4        87.50   112.50  100.00 25.000 0.1091  7854        5.236e+005 48.991
5        112.50  137.50  125.00 25.000 0.08715 1.227e+004 1.023e+006 49.038
6        137.50  162.50  150.00 25.000 0.07255 1.767e+004 1.767e+006 49.086
7        162.50  187.50  175.00 25.000 0.06215 2.405e+004 2.806e+006 49.134
8        187.50  212.50  200.00 25.000 0.05436 3.142e+004 4.189e+006 49.181
9        212.50  237.50  225.00 25.000 0.0483  3.976e+004 5.964e+006 49.229
10       237.50  262.50  250.00 25.000 0.04347 4.909e+004 8.181e+006 49.277
11       262.50  287.50  275.00 25.000 0.03951 5.94e+004  1.089e+007 49.324
12       287.50  312.50  300.00 25.000 0.03621 7.069e+004 1.414e+007 49.372
13       312.50  337.50  325.00 25.000 0.03342 8.296e+004 1.797e+007 49.420
14       337.50  362.50  350.00 25.000 0.03103 9.621e+004 2.245e+007 49.467
15       362.50  387.50  375.00 25.000 0.02896 1.104e+005 2.761e+007 49.515
16       387.50  412.50  400.00 25.000 0.02715 1.257e+005 3.351e+007 49.562
```

# Project Table, (prj.300)

## Overview

The Project table is used to store the additional project data/information that the user enters into the M300 Project display tab. This information is also placed at the beginning of any acquisition file is the store tables option is checked. Note that this is for informational purposes only and does not affect the way and data is recorded or handled.

## Parameters

**Project Name**

The name of the project as assigned by the user. Note that this does not have to be the same as the project folder name, in fact this is the field project name as opposed to the M300 project name. (Can be any string up to 256 characters in length.)

**Flight ID**

A flight identification assigned by the user.

**Aircraft Type**

The aircraft type being used if applicable.

**Aircraft ID**

An aircraft identification assigned by the user.

**Operator Name**

Name of the person(s) operating the M300 for the acquisition.

**Comments**

Any additional information can be added here.

**Data Prefix**

This is the prefix to use with the M300 data files for the auto name feature.

## Example

```
; Version = 1
; prj.300
ProjectName   = Experiment
FlightId      = APE03
AircraftType  = Boeing 707
AircraftId    =
OperatorName  = Operator
Comments      = None
DataPrefix    = hur
```

# Radar Table, (rdr.300)

## Overview

The Radar Table is used to define parameters for different Radar entries. The Radar data is then usable in the Formula Table (fml.300) and possibly other tables. For example the RaConstant() function needs to have a Radar entry specified.

The Radar table also defines color schemes used with the Radar entries.

## Parameters

**Version**
Shows which version of the M300 this setup table was created by.

Radar
- **Name**
- **Number**
- **Frequency**
- **TransmitPower**
- **NoisePower**
- **NoiseFigure**
- **ReceiverGain**
- **SaturationPower**
- **AntennaGain**
- **HorizontalBeamWidth**
- **VerticalBeamWidth**
- **TransmitPulseWidth**
- **ScanRate**
- **PulseRate**

**ColorSheme**
- **Name**
- **Number**
- **ANoiseLevel**
- **BNoiseLevel**
- **AMinimumRange**
- **BMinimumRange**
- **AMaximumRange**
- **BMaximumRange**
- **Range**
- **Level(1-16)**
- **Color**
- **State**
- **Minimum**
- **Maximum**

## Example

```
; Version = 1
; rdr.300
[RadarCPR]
Name                    = CPR
Number                  = 0
Frequency               = 3.486e+010
TransmitPower           = 8000
NoisePower              = -2.5
NoiseFigure             = 7.5
ReceiverGain            = 45
SaturationPower         = 11
AntennaGain             = 50
HorizontalBeamWidth     = 4.6
VerticalBeamWidth       = 0.5
TransmitPulseWidth      = 1.1e-007
ScanRate                = 28
PulseRate               = 118.5
[ColorSchemeAugie]
Name                    = Augie
Number                  = 0
ANoiseLevel             = -47
BNoiseLevel             = -42
AMinimumRange           = 0
BMinimumRange           = 0
AMaximumRange           = 20000
BMaximumRange           = 20000
Range   = 50
Level1          = 0x00FF00 1      0.000000   4.000000
Level2          = 0x66CC66 1      4.000000   8.000001
Level3          = 0x00A000 1      8.000001   12.000002
Level4          = 0x00A000 1      8.000001   12.000002
Level5          = 0x00FFFF 1      16.000002  20.000002
Level6          = 0x0099FF 1      20.000002  24.000004
Level7          = 0x0000FF 1      24.000004  28.000004
Level8          = 0x0000A0 1      28.000004  32.000004
Level9          = 0xFFFF7F 1      32.000004  36.000004
Level10          = 0xFFFF00 1      36.000004  40.000004
Level11         = 0xFE8001 1      40.000004  44.000004
Level12         = 0xFE5555 1      44.000004  48.000008
Level13         = 0xFF0000 1      48.000008  52.000008
Level14         = 0xFF00FF 1      52.000008  56.000008
Level15         = 0x8500B6 1      56.000008  60.000008
Level16         = 0x000000 1      60.000008  64.000008
```

# Secondary Acquisition Table, (saq.300)

## Overview

The M300 is able to store various secondary acquisition data. This is usually derived data from the raw data acquired. The source of the secondary acquisition is any formula entered into the formula table (fml.300). Note that secondary acquisition works for only for acquisition mode, it is not supported by playback or UDP mode.

## Parameters

**Name**

The name is the identifier for the secondary acquisition event entry. This name can be used in other tables to access the data. For example, “Temperature" (see also, "Name" on page 537).

**Number**

A unique integer to identify the entry.

**Tag**

The tag number will identify the for secondary acquisition event data. This number can be used in other tables to access the data. The tag number must be unique, which means it cannot be used in the acquisition table. Do not use a reserved tag number (see also, "Tag" on page 538).

**Index**

Used to extract a particular value from the given **Formula** (see below). If set to -1, it is assumed the subsequent given formula is comprised of only one value.

**Formula**

The formula holding the values to be stored as a secondary acquisition event

## Example

```
; Version = 1
; saq.300
; Name            Number Tag   Index Formula
"State"           0      9000  -1    F997
"1DCommandValue"  1      9001  -1    F998
"Range"           2      9002  -1    F1000
"Sizes"           3      9003  -1    F1010
"Counts"          4      9004  -1    F1020
```

# Skew-T Display Table, (skt.300)

## Overview

The Skew-T (skewed temperature/logarithmic pressure) diagram is a real time display of temperature and humidity in the atmosphere. The horizontal axis is temperature in Celsius (C). The vertical axis is atmospheric pressure in millibars (mb), which decreases with altitude. The measurements depicted in a Skew-T diagram are collected in a "sounding" of the air.

The Skew-T will usually have two entries, one for temperature and one for humidity.

This displays supports all the standard grid lines for a Skew-T diagram, such as isobars, isotherms, dry adiabats, saturated adiabats and mixing ration lines.

There is also a vertical bar which shows a profile of wind speed and directions (this is an option).

## Parameters

**Name**

The name is the identifier for the Skew-T entry (see also, "Name" on page 537).

**Number**

A unique integer (Note that multiple Skew-T displays can have the same integer) used to identify this display to the M300. If the user has multiple Skew-T displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a Skew-T display does also, then a command set up to change the color of the Skew-T display will not affect the HVPS display.

**Window**

The name of the window where the Skew-T display is to be performed. This window must be of Skew-T type (see also, "Window" on page 537).

**Color**

Color used for Skew-T display (see also, "Color" on page 538).

**WindColor**

Color used for Wind in Skew-T display (see also, "Color" on page 538).

**Type**

This parameter is used to select what gets drawn once a new point is found.

| Name | Type |
|---|---|
| Point | 0 |
| Line | 1 |
| Bullet | 2 |
| Line w/bullet | 4 |

**Type**

**Width**

Line width for the Skew-T entry. This is normally 1 pixel wide. Larger value for line width will require more drawing and slow down the display. You should keep this in mind when changing the line width.

**Entries**

The number of entries kept in memory for each Skew-T pair. If the number of entries is '0', then the Skew-T plot has no memory.

**xFormula**

Data source for x-axis. This is usually a formula for temperature or humidity in degrees Celsius (see also, "Formula" on page 538).

**yFormula**

Data source for y-axis. This is usually pressure altitude in millibars (mb) (see also, "Formula" on page 538).

**xMin, xMax**

Minimum and maximum limits for x-axis.

**yMin, yMax**

Minimum and maximum limits for y-axis. In the Skew-T display the y-axis limits are inverted, this means we have higher value for pressure on the minimum limit than we have for the maximum limit.

**SpdFml, DirFml**

Data sources for wind speed and direction. This speed must be knots and direction in radians (see also, "Formula" on page 538).

## Example

```
; Version = 3
; skt.300
; Name  Number Window Color WindColor Type Width Entries xFml yFml xMin xMax yMin yMax spdFml dirFml
"temp"      0   skewt  Red    Green     1     1    1500 F512 F510  -40   40  1050 100   F200   F300
"dewp"      1   skewt Blue    Green     1     1    1500 F513 F510  -40   40  1050 100    F-1    F-1
```

# Strip Chart Display Table, (stp.300)

## Overview

This display is made to emulate a strip char recorder. New data points appear in one side of the display while the data scrolls to the opposite side to show a history of the data over a period of time.

The number of strip chart display entries per window is not limited. It is up to the user to pick the desired number of strip chart entries. The user should consider window size and overall visual appearance when picking a large number of strip chart entries.

## Parameters

**Name**

The name is the identifier for the Strip Chart entry (see also, "Name" on page 537).

**Number**

A unique integer (Note that multiple Strip Chart displays can have the same integer) used to identify this display to the M300. If the user has multiple Strip Chart displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a Strip Chart display does also, then a command set up to change the color of the Strip Chart display will not affect the HVPS display.

**Window**

Each entry in the Strip Chart Table must belong to a window. This parameter is the name of the window where the Strip Chart display will be done. The type of the window must be Strip Chart display. For example, "stp" (see also, "Window" on page 537).

**Color**

The color used for the Strip Chart entry. The user can specify any color, there are no restrictions. Some color choices are better than others. The user should pick colors that will be highly visible versus the window background. The desired colors should be different enough from other colors to avoid confusion (see also, "Color" on page 538).

**Type**

Plot type for strip charts. Use ‘0’ for points. Use ‘1’ for lines (default).

**Width**

Line width for the Strip Chart entry. This is normally 1 pixel wide. Larger values for line width will require more drawing and slow down the display. You should keep this in mind when changing the line width.

**State**

The state variable is used to control when a strip chart entry is visible and active (1) or not visible but active (0). Data for a strip chart entry is always updated, even when the display is not visible (see also, "State" on page 538).

**Decimate**

This field is used to control the number of data points to be displayed. A value of *n* selects every *n*th data point for display.

**Group**

Strip chart entries can be placed in groups. This field is a string with the strip chart group name. When running the M300 the user can easily select different groups of strip chart entries for display.

**Index**

Index of the formula value to display. Use -1 for no index.

**Formula**

Link to formula which is to be used in the Strip Chart entry. The data value from this formula will be used to update the Strip Chart display (see also, "Formula" on page 538).

**yMin**

This specifies the minimum limit for the y-axis.

**yMax**

This specifies the maximum limit for the y-axis.

## Example

```
; Version = 5
; stp.300
; Name Number Window Color Type Width State Decimate Group Index  Formula yMin  yMax
"0:LWC"     0 pristp   red    1     1     1        1  "main"    -1    F3521 -0.2  0.8
"1:AmbTemp" 1 pristp green    1     1     1        5  "main"    -1    F3315  -15   10
"2:AmbRH"   2 pristp  blue    1     1     0       10  "main"    -1    F3402    0  100
```

# Triggered Command Table, (tic.300)

## Overview

The Triggered Command Table is used to execute any M300 command manager commands automatically, based on user trigger definitions. This table is used to define triggered event commands. For user keystroke executed commands, See "Command Table, (cmd.300)"

## Parameters

### Comment

This entry describes what the particular defined function(s) is/are for. Must be started with a semi-colon ";" (see also, "Comments" on page 537). There are certain limitations on the use of comments and where they may be put.

### Trigger

The trigger entry here has the same syntax as all trigger entries in setup tables (See "Trigger" ). This particular trigger entry will tell the M300 when the subsequent command block(s) will be executed (See "Command Entry/Block" on page 632.).

### Command Entry/Block

The command block is a series of commands to be executed when the preceding defined trigger fires. The command block may contain a single command entry, or a series of commands that will be carried out in sequence. The number of command entries in a single command block can contain 1 to *n* commands, where *n* is limited only by memory constraints. The command blocks are carried out in the following manner:

```
.Trigger = "PriType" PriFreq PriAddress "SecType" SecFreq SecAddress
command1  [arg0  arg1  ... argm]
command2  [arg0  arg1  ... argm]
   .        .     .         .
   .        .     .         .
commandn  [arg0  arg1  ... argm]
```

The commands can contain from 0 to *m* additional arguments that will then be used in the command execution, where *m* is the maximum number of arguments accepted for a particular function (See "Commands" ). Some arguments are optional, while others are required. See the function specification for details.

For each defined function definition, more than one command block may be defined. The pound sign '#' is used to delimit one command block from another. The command blocks are executed based on the number of times the defined trigger has fired. The first time the trigger fires. the first command block will execute; the second time it fires, the second command block will execute and so forth. Again, the command blocks may rang from 0 to *n*, where *n* is limited only by memory

constraints. When the defined trigger fires *n*+1 times, the first command block will execute, and the cycle will start over. All function calls are not case-sensitive. The syntax for the command blocks is as follows:

```
.Trigger = "PriType" PriFreq PriAddress "SecType" SecFreq SecAddress
command block1
#
command block2
#
   .       .
   .       .
#
command blockn
```

### Commands

For a list of valid M300 Command Manager commands, See "Command Manager Reference"

## Example

```
; Version = 1
; tic.300
.Trigger = "Sync" 1 None "Never" Never None
cmd1d fssp 1
#
cmd1d fssp 2
.Trigger = "Sync" 1 None "Never" Never None
scn 0
#
scn 1
#
scn 2
.Trigger = "Sync" 1 None "Never" Never None
scn NEXT
#
scn PREV
.Trigger = "Sync" 0.1 None "Never" Never None
dummy
#
quit
```

# Text Display Table, (txt.300)

## Overview

The simplest type of display is the text display. This type of display renders alphanumeric data to a window. Unlike the M200 system where text entries could be displayed in any window type, in the M300 system the window must be a text window. Another difference is that the M300 system has several different types of text entries. These are Labels, Data, Table, Data No Units and Table No Units entries. In the M200 system the text entry consisted of a label/data pair.

The format field is usually optional. If there is no format field specified, the text display manager assumes the correct default values based on the formula type.

The text entry will automatically display the units for a formula following the data, this applies to Data and Table entry types. If no units are desired, then use the Data No Units and Table No Units entry types.

## Parameters

**Name**

The name is the identifier for the Text entry. For example "Temperature" (see also, "Name" on page 537).

**Number**

A unique integer (Note that multiple Text displays can have the same integer) used to identify this display to the M300. If the user has multiple Text displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a Text display does also, then a command set up to change the color of the Text display will not affect the HVPS display.

**Window**

Each entry in the Text display table needs to belong to a window. This parameter is the name of the window where the text display will be done. The type of the window must be Text window. For example "txt" (see also, "Window" on page 537).

**Type**

The type of the text entry. There several types of text entries, see table bellow. Different text entry types are provided to facilitate the textual display process.

The Label entry is used to place any kind of text on the window.

The Data entry is used to place a textual data value representing the current value of a particular formula.

The Table entry is basically the same as the Data entry. It has the added capability of displaying all the elements of a formula array in a tabular grid.

The Data No Units entry is the same as Data, without displaying the units field from the formula table.

The Table No Units type is the same as Table entry, without displaying the units field from the formula table.

| Type | Usage |
|---|---|
| 0 | Label |
| 1 | Data |
| 2 | Table |
| 3 | Data No Units |
| 4 | Table No Units |

**Type**

**Font**

The font used to display the Text entry. It is recommended that a fixed font be used to display data labels. Otherwise we run into problems with 'erasing' the previous data value.

**Color**

The Text display allows the user to pick whatever color he desires for the text entry (see also, "Color" on page 538).

**X, Y**

The x and y position for the text entry display in pixels. The upper left corner of the text window is the origin (0, 0).

The Label type text entries are drawn with left/top alignment. This means that the x and y position mark the top left corner of the text entry.

For Data and Table entries the entries are drawn with right/top alignment. This means that the x and y position mark the top right corner of the text entry. The units will be shown after the data with the same alignment as the Label entry.

**W, H**

The width and height of the text entry. This is necessary for several reasons, like the rectangular selection box and controlling the spacing between table items.

**Index**

The index of the desired entry from the formula (0 for the first element). The formula must be an array of values and the index must be valid. For no index use a '-1'.

**Formula**

Formula link for data to be shown (see also, "Formula" on page 538).

**Format**

The format is optional. If the format is blank, the default format will be used based on the formula type (see also, "Format" on page 539).

**Rows, Columns**

Number of desired rows and columns for table entry display.

| Syntax | Description | Formula/Result | Conversion | Result |
|---|---|---|---|---|
| %s | String | S[5] | %s | Hello |
| %d | Decimal (base 10) | I[1]/100 | %5d | 00100 |
| %x | Hexadecimal (base 16) | I[1]/254 | %02x | FE |
| %f | Float (single precision) | F[1]/120.322 | %3.2f | 120.32 |
| %g | Float (double precision) | D[1]/210.119191 | %3.5g | 210.11919 |

**Format Syntax**

## Example

```
; Version = 2
; txt.300
; Type = 0, Label
; Name      Number Window Type Font    Color      X    Y
"TAS"       1      text   0    lu20    0x00FFFF   3    21
; Name      Number Window Type Font    Color      X    Y    W    H
"Counts"    1      text   0    cour14  0xFEFEBD   252  281  42   15
; Type = 1, Data
; Name      Number Wndw Type Font  Color   X    Y    Index Formula
"RawPC"     0      text 1    lu20  0xCCCC  201  235  -1    F1011
; Name      Number Wndw Type Font  Color   X    Y    Index Formula  Format
"Command"   0      text 1    lu20  0x7FFE  433  110  -1    F1009    "0x%02X"
; Name      Number Wndw Type Font  Color   X    Y    W   H   Index  Formula
"TasClk"    1      text 1    lu20  0x7FF0  209  45   10  20  -1     F1102
; Name     Number Wndw Type Font  Color   X    Y    W   H   Index Formula Format
"TasClk"    1      text 1    lu20  0x7FFC  209  45   10  20  -1     F1102  "0x%3l"
; Type = 2, Table
; Name      Num Wndw Type Font   Color    X  Y    W   H   Rows Columns Formula
"Sizes"     0   text 2    cour14 0xBDFFBE 42 281  42  15  16   4       F1000
; Name     Num Wndw Type Font   Color    X  Y  W  H  Rows Columns Formula Format
"Counts"    2   text 2    cour14 0xFEFEBD 10 5  42 15 16   4      F1001   "%4.1f"
```

# Window Table, (wnd.300)

## Overview

The Window Table is one of the most important tables in the M300 system. It keeps track of all the current windows for the project. Each window is identified by a window name and type. The window name is used in other displays/tables to pick where the displays will be performed. The window type selects what type of display will be performed for each window.

In the M200 system all window properties where specified in the window table. The M300 system has a window table (wnd.300) and for each window in a project a window configuration file (See "Window Table Configuration File, (*.wnd)" on page 640.). The window configuration file keeps all the properties for the specific window.

The user doesn't need to edit any of the window properties via the project files. All properties can be modified via the configuration dialogs in the M300 software.

## Parameters

**Filename**

This doubles as the window name and the name of the file where the window configuration properties will be stored. The M300 system appends a '.wnd' to this name to lookup the specific window configuration file. The filename should not contain any spaces, in fact it's recommended to use only letter and numbers.

**Number**

An integer identifying the window to the M300. This is needed in order to identify windows during command manager operations. For instance, if a command is set up to alter the display of a strip chart, an integer is used to identify all strip chart displays to be updated by that command.

**Type**

This parameter is used to specify the window type. There are several different kinds of window/displays supported under the M300 system. The M300 has some new displays which didn't exist in the M200 system. This include the Skew-T display and displays for Radar data (Height Time Indicator and Plan Position Indicator).

| Type | Window | Table |
|---|---|---|
| 0 | 2D Grey | 2dg.300 |
| 1 | 2D Mono | 2dm.300 |
| 2 | Height Time Indicator | hti.300 |

**Window Types**

| Type | Window | Table |
|---|---|---|
| 3 | High Speed Analog | hsa.300 |
| 4 | Histogram | his.300 |
| 5 | HVPS | hvp.300 |
| 6 | Moving Position Indicator | mpi.300 |
| 7 | Multi Text | mlt.300 |
| 8 | Plan Position Indicator | ppi.300 |
| 9 | Position | pos.300 |
| 10 | Probe Distribution | pdi.300 |
| 11 | Strip Chart | stp.300 |
| 12 | String | str.3001 |
| 13 | Text | txt.300 |
| 14 | Time vs. Y | tvy.300 |
| 15 | Vector | vec.300 |
| 16 | X vs. Y | xvy.300 |
| 17 | Cloud Imaging Probe | cip.300 |
| 18 | Hodograph | hod.300 |
| 19 | Skew-T | skt.300 |
| 20 | Moving Air Mass | mam.300 |
| 21 | Formula Watch & Alter | fwa.300 |
| 22 | Reserved | |
| 23 | Reserved | |
| 24 | Reserved | |
| 25 | Cloud Imaging Probe Grey Scale | cgs.300 |

**Window Types (Continued)**

## Example

```
; Version = 2
; wnd.300
; Name        Number      Type
PiraqARaw         0         16
PiraqAPower       1         16
HtiPower          2         2
HtiDoppler        3         2
HtiReflectivity   4         2
PiraqAConfig      5         13
PiraqAStatus      6         13
Arinc429          7         13
fwa0              8         21
fwa1              9         21
```

# Window Table Configuration File, (*.wnd)

## Overview

The window configuration files are used to keep all the properties for a window. This includes window position, size, etc.

## Parameters

**Trigger**

The primary trigger is used to determine how often the display will be done for a particular window. For example, in the case of text data for a FSSP probe the user may trigger the window once per second. On the other hand for a 2D Image, the trigger is selected for 2D Image data type and the frequency would select the maximum number of buffers displayed.

Certain display types require a secondary trigger. For example in the case of the 2D Image a secondary trigger of 1 Hz is required to keep track of the image age and to perform the hashing.

**Area**

The area is used to keep the window position (x, y) and the window size (w, h). These are specified in window coordinates.

**Split**

The split values are used for the displays that utilize the split feature on the M300 data displays. The format is (split = 'x-split value' 'y-split value'). A value of -1 instructs the M300 to use auto mode, while a positive value *n* instructs the M300 to offset the display from the x or y labels by *n* pixels.

**Closed**

Closed keeps the window open/closed state. If a window is closed, this would be a one, otherwise a zero for open windows.

**Minimized**

This keeps the window minimized state. A one indicates the window is minimized and a zero indicates the window is normal.

**GridState**

This keeps track of the grid state. A one indicates the grid is turn on and should be displayed. A zero indicates no grid desired.

**Background**

Window background color (see also, "Color" on page 538).

**Grid**

Window grid color (see also, "Color" on page 538).

**Text**

Window text color (see also, "Color" on page 538).

**XScale, YScale**
X-axis and y-axis scale.

| Name | Scale |
|---|---|
| LINEAR | 0x00000001 |
| LOGARITHMIC | 0x00000002 |
| TIME | 0x00000004 |
| DIVISIONS | 0x00000010 |
| UNITS | 0x00000020 |
| PIXELS | 0x00000040 |
| BINS | 0x00000080 |

**Scale**

**XMode, YMode**
X-axis and y-axis mode.

**XTypeMinor, XTypeMajor, yTypeMinor, yTypeMajor**
X-axis and y-axis type for both minor and major grids.

| Name | Type |
|---|---|
| SOLID | 0x00000100 |
| DOTTED | 0x00000200 |
| DASHED | 0x00000400 |
| BOTH | 0x00001000 |
| FIRST | 0x00002000 |
| LAST | 0x00004000 |
| LEFT | 0x10000000 |
| RIGHT | 0x20000000 |
| UP | 0x40000000 |

**Type**

| Name | Type |
|---|---|
| DOWN | 0x80000000 |

**Type (Continued)**

SOLID, DOTTED and DASHED are used to pick the grid line type and therefore either a horizontal or vertical line grid.

BOTH, FIRST and LAST are used to pick the grid placement. BOTH picks tick marks on the left and right sides or top and bottom. FIRST picks tick marks on the right side or top. LAST picks tick marks on the left side or bottom.

**XGridMinor, XGridMajor, YGridMinor, YGridMajor**

X-axis and y-axis minor and major grid frequency.

**XLabel, YLabel**

X-axis and y-axis grid label frequency.

**XMinimum, XMaximum, YMinimum, YMaximum**

Minimum and maximum limits for x-axis and y-axis.

**XRange, YRange**

Range value for x-axis and y-axis.

**Color0 ... Color15**

Each window has a 16 color pallet (see also, "Color" on page 538).

**Mode**

Determines the behavior of the X vs. Y plot. If mode is zero, the X vs Y plot will be refreshed as normal. If mode is one, the plot history will remain on the screen any new data points will accumulate. To perform a sounding, mode must be set to one.

## Example

```
; Text.wnd
; Type = type
; Trigger      = Trigger1 Frequency1 Board1 Trigger2 Frequency2 Board2
Trigger        =     Sync          1 Never     Never      Never   None
Area           = 499  5    586  589
Split          = -1 -1
Closed         = 0
Minimized      = 0
GridState      = 1
XScale         = 0x00000009
Background     = 0xFFFFFF
Grid           = 0x606060
Text           = 0x000000
XMode          = 0x00000000
XTypeMinor     = 0x00000200
```

```
XTypeMajor      = 0x00000200
XGridMinor      = 0
XGridMajor      = 0
XLabel          = 0
XMinimum        = 0
XMaximum        = 0
XRange          = 0
YScale          = 0x00000009
YMode           = 0x00000000
YTypeMinor      = 0x00000200
YTypeMajor      = 0x00000200
YGridMinor      = 0
YGridMajor      = 0
YLabel          = 0
YMinimum        = 0
YMaximum        = 0
YRange          = 0
Color0          = 0x00FF00
Color1          = 0x66CC66
Color2          = 0x00A000
Color3          = 0x008070
Color4          = 0x00FFFF
Color5          = 0x0099FF
Color6          = 0x0000FF
Color7          = 0x0000A0
Color8          = 0xFFFF7F
Color9          = 0xFFFF00
Color10         = 0xFE8001
Color11         = 0xFE5555
Color12         = 0xFF0000
Color13         = 0xFF00FF
Color14         = 0x8500B6
Color15         = 0x000000
Mode            = 0
```

# X vs. Y Display Table, (xvy.300)

## Overview

This display consists of the typical X vs. Y plot where two values are used to pick a display point. The X vs. Y display has the capability to connect all the points together including from one time period to the next. This could be used to construct a sounding. Other uses for the X vs. Y display include spectrum plots of particles and scatter plots of various variables.

## Parameters

**Name**
The name is the identifier for the X vs. Y entry. For example, “2DC Conc”.

**Number**
A unique integer (Note that multiple X vs. Y displays can have the same integer) used to identify this display to the M300. If the user has multiple X vs. Y displays, they can assign different and/or the same integers to each display based on the intended usage of the M300 command manager. Note that these integers are unique to the display type only, they are not global to the M300. For instance, if an HVPS display has a one assigned to it and a X vs. Y display does also, then a command set up to change the color of the X vs. Y display will not affect the HVPS display.

**Window**
Each entry in the X vs. Y display table needs to belong to a window. This parameter is the name of the window where the X vs. Y display will be done. The type of the window must be X vs. Y display. For example, “xvy” (see also, "Window" on page 537).

**Color**
The user selectable color for the X vs. Y entry (see also, "Color" on page 538).

**Type**
This parameter is used to select what gets drawn once a new point is found.

| Name | Type |
|---|---|
| Point | 0 |
| Line | 1 |
| Bullet | 2 |
| Line with Bullet | 3 |

**Type**

**Width**

Line width for the X vs Y entry. This is normally 1 pixel wide. Larger value for line width will require more drawing and slow down the display. You should keep this in mind when changing the line width.

**State**

The state variable is used to control when a X vs Y entry is visible and active (1) or not visible but active (0) (see also, "State" on page 538).

**Group**

X vs Y entries can be placed in groups. This field is a string with the X vs Y group name. When running the M300 the user can easily select different groups of X vs Y entries for display.

**Entries**

The number of entries kept in memory for each X, Y pair. If the number of entries is '0', then the X vs. Y plot has no memory.

**xFormula, yFormula**

Data sources for X vs Y plot. The x data source will be used to pick a point along the x-axis, while the y data source will be used to pick a point along the y-axis.

**xMin, xMax**

These specifies the minimum and maximum limits for the x-axis.

**yMin, yMax**

These specifies the minimum and maximum limits for the y-axis.

## Example

```
; Version = 3
; xvy.300
; Name Number Window Color Type Width State Group Entries xFml  yFml xMin xMax yMin yMax
"Amb RH%"    0  mainxvy   Red  1    1    1   ""   0     F3402 F3417   0   100    0 20000
"Dewpoint"   1  mainxvy Green  1    1    1   ""   0     F3409 F3417 -30    20    0 20000
```