

# RETDAT Waveform Variations

*Logic changes made*

Fri, Jun 23, 2000

To support several variations of desired data relating to an analog signal specified via the SSDN, modifications have been made to the RETDAT logic. Examination of the user program-specified parameters of period, offset, and length determines what form of data reply is sought. This note defines the interpretation on these parameters made by RETDAT.

## *Forms of data*

There are two basic forms of data a user may request that relate to an analog signal, plus time-stamped variations of each, for a total of 4 forms.

<i>Simple data format</i>	<i>Period</i>	<i>Offset</i>	<i>Length</i>
Reading	any	0	2
Waveform segment	any<>2	any	points*2

The time-stamped structures, designed to return 15 Hz data, are variations of the above.

<i>Time-stamped data format</i>	<i>Period</i>	<i>Offset</i>	<i>Length</i>
Reading	2	0	8
Waveform segment	2	any	4 + 2*points*2

Since the purpose of providing time-stamped data is to give reliable access to correlated 15 Hz data, the period is used to determine the number of sets of 15 Hz data that are contained in the reply. With a 4-byte header, the number of sets is given by period, and the size of each set is therefore  $(\text{length} - 4) / \text{period}$ , where the period is in units of 15 Hz cycles. This logic breaks down if the request is a one-shot or is event-based, so that time-stamped data formats are not returned in those cases. Restricting waveform array access to 7.5 Hz replies helps to resolve ambiguities here. There is no particular reason why 7.5 Hz replies should not be used when trying to obtain 15 Hz time-stamped data.

For all time-stamped formats, a 4-byte header precedes the data sets. The header consists of a word that is the number of sets included in the reply, followed by a time stamp word that is the 15 Hz global cycle number, which is derived from the multicast clock event message. The number of sets is normally equal to the period, which should be 2, except for the first prompt reply in which it is 1. The minimum size of this structure is 8 bytes. The points value in the above formulas is the number of 2-byte data values in a waveform array. Here is the form of a typical 8-byte time-stamped reply at 7.5 Hz.

0002	number of data sets = 2 for 7.5 Hz
time1	low 16 bits of multicast 15 Hz cycle counter
data1	data word for first cycle
data2	data word for second cycle

A waveform segment may specify a zero offset to access the waveform starting at the beginning; otherwise, it can specify another starting point via a suitable byte offset value. The total waveform size is unknown, so the user gets the size requested. The first waveform element cannot be accessed alone, because a zero offset value yields the Reading value.

## *Interpret request parameters*

Initialize

avgFlag = false  
perCycle = false  
waveFlag = false  
nBytes = length

Check the following conditions:

Listype = 0 (analog reading) or 1 (analog setting)  
 Acnet property index = 12 (Reading) or 13 (Setting)  
 length even  
 offset even

If the above conditions are true, then check the following:

If length = 2 and offset = 0 --> reading (or setting), avgFlag = true, perCycle = true  
 else

If period  $\neq$  2 --> waveform segment, waveFlag = true, nBytes = length  
 else

If length = 8 and offset = 0 --> t-s reading or setting, perCycle = true  
 else --> t-s waveform segment, waveFlag = true, perCycle = true,  
 nBytes = (length - 4) / 2

If perCycle = true, set bit 15 of RDI field, and if avgFlag = false, set bit 14 also.  
 If waveFlag = true, set bit 13 of RDI field.

During DOPTRS processing, which initializes the internal pointer structures, if RDI bit 13 is set, check for a waveform defined for the channel by searching the CINFO table; if it is, change the internal pointer to the address of the waveform segment. If perCycle = true, set nBytes = (length - 4) / 2, and clear bit 13. (If no waveform is defined for the channel, leave internal pointer as address of ADATA entry.) If bit 15 set, it means internal pointer is two long words. If bit 15 is set, and bit 14 is set, nBytes is placed in hi word of second long word of internal pointer structure.

### *Averaging logic*

There is some averaging logic used for RETDAT requests, in the case that the Reading property is used with a listype of 0 specified in the SSDN and a length of 2 bytes and period of more than two cycles. The averaging is meant to preferentially average over beam pulses, returning a non-beam average if no beam pulses occurred within the averaging interval, which is the same as the reply period. Because this logic provides a "hook" that permits some kind of updating every 15 Hz cycle, if only to accumulate the latest reading values into a sum, it can help to support the time-stamped options listed above. The averaging option is marked by setting the sign bit in the 16-bit RDI field, which is a small ( $\leq 25$ ) read routine index number. A status bit is checked to determine whether beam is present; if the status bit never changes, the averaging occurs over all cycles in a period.

According to the above tables, the time-stamp options come into play if the period is 2 and the length is at least 8. For each such case, there is work to be done every 15 Hz cycle.

For handling the averaging case, two long words serve as the internal pointer structure rather than only one long word. Bit 15 of the RDI field in the DRB structure signals special per-cycle processing that now includes the new options as well. Bit 14 of the RDI field signals not averaging, but other per-cycle special processing. The first long word is the usual internal pointer value returned by the pointer type routine for listype 0 or 1, which simply points to the appropriate ADATA field. The second long word is used to house two words: the number of bytes of data to copy per cycle and the current offset into the answers structure. On the first cycle of the period, the answers area is prepared by clearing the Acnet status word, clearing the count of data sets, setting the time stamp word to the current cycle number, and setting the current offset to 6 to skip past the Acnet status word and 4-byte header. On each cycle of the period, the number of bytes of data are copied into the answers area at the offset indicated, the offset is advanced, and the count of data sets is incremented. On each successive cycle, then, more data is copied into the area to build up the answer buffer. On the cycle when the reply is due, which is the second cycle, no further processing is needed.

### *Floating point option*

A new system table called FDATA, which is parallel to ADATA, will be available for holding floating point readings, settings, nominal and tolerance values. The above logic also recognizes the

floating point option for a new listype that targets the single precision reading field. In this case, a request for 4 bytes of data, using listype 90 in the SSDN, is recognized as a request for floating point readings from the FDATA table. If the number of bytes requested is 12, and the period is 2 cycles, then time-stamped values of such readings will be returned. If the period is something slower than 2 cycles, then averaging logic is enabled that is a floating point computation.

### *SSDN variations*

There are two variations currently used in interpreting support of SSDNs. One relates to use of the offset word to modify the third word of the 4-word structure. It can be used to collect data from a channel other than that specified in the SSDN by adding the offset word to it. It is most often used to access analog descriptor records from a channel specified by the offset word. (In these cases, the SSDN specifies channel 0 so that the value of the offset word is the channel number.) The reason for this interpretation is partly to make up for not being able to alter the SSDN that is sent to the front end. This feature has also been used to select the event number used as a time base reference for the 1 KHz digitizer data. (In this case, the offset is applied to the long word composed of the 3rd and 4th words of the SSDN.)

A second variation is used when the low byte of the fourth word of the SSDN is non zero. In that case, it is interpreted as the size of a single item of data to be retrieved from each channel. For example, when the low byte is 2, it allows a request for  $N*2$  bytes to return an array of  $N$  2-byte readings of consecutive analog channels. It has been used for access to HLRF trip logs that are treated as an array of words.

The use of the offset word to modify the channel number as described above is enabled by setting the offset flag nibble to 1. In order to add support for waveform access, this nibble must be 0. So, one can specify whether the channel number is modified by the offset or that a byte offset is interpreted as a means of indexing into a waveform, but not both. If the offset flag nibble is 1, it modifies the channel number field of the SSDN before it is interpreted, after which the offset word is cleared, so that it cannot be used for indexing into a waveform.

The use of access to consecutive channel data by specifying a sufficient length is not compatible with supporting waveform access, but it can permit access to time-stamped arrays of consecutive channel readings, in which the low byte of the 4th word of the SSDN is 2. Here is the usual form of an SSDN for doing the special waveform and time-stamped stuff:

```
0001 node chan 0000
```

Note that the offset flag nibble is zero, and the item size byte is zero.

### *Application data pool management for correlated 15 Hz data*

Time-stamped data is delivered every 7.5 Hz from the front ends, because the application made 7.5 Hz requests for each set of data to be correlated in this way. For each data set, the application manages an array to hold multiple sets of the return values, each including a “new data” flag.

On each 15 Hz cycle execution of the application, the console data pool must be polled for each data set. If the return status indicates that the data changed while it was being collected, then repeat the polling call. If the return status indicates that the data is new, as opposed to the same received last cycle, then copy the data into the array of data sets and set the flag for that set. The array element that holds the data set is determined by the cycle number gleaned from the time-stamped data structure received. For example, if 4 data sets are housed in the array, then use the least significant 2 bits of the time stamp cycle number as an array index. Before copying, check the new data flag; if it is set, it means that this is a repeat of data already collected, so it doesn't need to be copied again.

After polling for data set updates, the application should check all new data flags for the “current” element, which was initialized to the first encountered data set since the requests were issued. If all data sets for that element are new, then process the correlated data. Advance the current element,

and check for all data sets new again, and if they are process that set of correlated data, too. (It will not be surprising to find two complete sets of data available at the same time, since two sets arrive from a front end in each reply.) Whenever a set of correlated data is processed, clear the new data flags for all sets of that element, so they can be discovered the next time around.

It would be useful to include some diagnostics that can detect missing data sets. There may need to be a time-out on a “stuck” current element. Maybe one can check all new data flags for each element, beginning with the current element. If an element is found with all its new data flags set, then advance the current element to that one, noting how many are skipped as a diagnostic.

This may seem confusing, but it is designed to correlate 15 Hz data from multiple front ends, which is not an easy problem. Maybe someone will think of an easier application data pool management scheme.

### Summary

In summary, then, what does the new support provide? With a single SSDN that specifies analog channel readings, one can make a request for the following:

<i>Choice</i>	<i>How</i>
16-bit integer readings	length=2, period<>2, offset=0
beam-preferred averaged readings	length=2, period>2,
array of N channel readings	length=N*2, offset=0
time-stamped readings	length=8, period=2, offset=0
time-stamped array of N readings	length=N*2*2+4, period=2, offset=0
waveform element[M]	length=2, period<>2, offset=M*2
array of N waveform elements at [M]	length=N*2, period<>2, offset=M*2
entire waveform of N elements (N>1)	length=N*2, period<>2, offset=0
time-stamped waveform element[M]	length=8, period=2, offset=M*2
t-s array of N waveform elements at [M]	length=N*2*2+4, period=2, offset=M*2
t-s waveform of N elements (N>1)	length=N*2, period=2, offset=0

Note that, as in the earlier descriptions, the period is here specified in 15 Hz accelerator cycles, although the actual FTD (Frequency Time Descriptor) passed in the RETDAT request is expressed in units of 60 Hz.

Note that to enable the array of N channels options, one must have specified the size of the data item (2 or 4) in the low byte of the SSDN 4th word. Without doing so, one will only get an array of data that begins at the ADATA reading but includes less desirable other stuff.

It is possible to set the offset flag nibble in the SSDN to enable the use of the offset word to (internally) modify the channel number in the SSDN, in which case the offset word is the delay channel number to be added (not twice that). In this case, however, the offset cannot also be used in the ways described above. (It will internally be set to zero after it has been used to modify the channel number.) This is unlikely to be useful.

As for the floating point data options, in which the hi byte of the first word of the SSDN is 90, one has the following choices:

<i>Choice</i>	<i>How</i>
32-bit single precision readings	length=4, period<>2, offset=0
beam-preferred averaged readings	length=4, period>2, offset=0
array of N channel readings	length=N*4, offset=0
time-stamped readings	length=12, period=2, offset=0
time-stamped array of N readings	length=N*4*2+4, period=2, offset=0

No waveform options have yet been defined for floating point raw data. Nonzero offset values

modify the data that is returned, but this feature is not known to be useful.

In cases where floating point raw data is used, the PDB should specify the null transform; *i.e.*, the raw floating point values returned are already in engineering units. Such data is “born” as a result of computations made by local applications.