

# SSDN Nuances

## *Waveform variations*

Tue, May 30, 2000

Acnet device names often refer to a particular signal. But it is increasingly possible with modern control system hardware to support digitization of entire waveforms of the same signals. Users often want to use the same name to refer either to a single reading of the signal or to a waveform. For the Acnet Reading property, this means that all variations of data related to the signal value must share the same 8-byte SSDN structure that is stored in the central database. An application program can communicate its interest in data for a device by using this SSDN, which it cannot modify, by the requested period, and by a 16-bit offset value and a 16-bit length value, which is the number of bytes of data requested. This note discusses how to support various types of request for an analog signal by using a fixed SSDN and suitable values for the offset and length words.

### *Single reading*

An analog device has a simple 2-byte (or 4-byte) reading value. This was the original meaning of the Reading property. Entering the Acnet device name on a Parameter Page sends a request for 2 bytes of data using a zero offset value.

Using a non zero offset value can be done via the parameter page by using C language array element syntax, such as [5] following a device name. The parameter page sends a request for two bytes, specifying an offset value that is twice the indicated array element index. The expectation is that an element of an array is being selected, perhaps an element of a waveform array.

One expects to be able to plot waveforms, too, for which the FTPMAN protocol is used. It still uses the same SSDN that is associated with the Reading property. Of course, one could define a different device name and thereby specify a different SSDN, but that would require a user to keep track of two device names for a given signal, one that is used to get one reading value and a different one to plot the waveform. If the Acnet database included a Waveform property, the FTPMAN client might be able to use a separate SSDN if that property exists, else it could default to the Reading property SSDN. This note assumes no such option exists.

### *Waveforms*

As one can specify an array element via RETDAT, an application can also access a waveform array by requesting a suitably large number of bytes. The offset scheme can also be used with it, so that the application can ask for an arbitrary piece of the array. The byte offset value will indicate where to start, and the requested number of bytes will indicate how much of the waveform is requested. A simple case would specify a zero offset value and request the size of the entire waveform array.

### *15 Hz data*

Another kind of request for multiple values of data from a device comes about because the Acnet application interface does not support reliable access to 15 Hz data returns. The application executes at an accelerator-asynchronous approximate 15 Hz rate, and it cannot be sure to be able to catch data from each 15 Hz cycle, even if the front end accurately delivers the data at 15 Hz.

An approach to get around this problem is to ask the front end to return data at 7.5 Hz—every two 15 Hz cycles—but to include two data values within each reply. The application would check for new data at each “15 Hz” execution in order to be sure it will not miss any 7.5 Hz reply.

But this scheme, although it allows collection of 15 Hz data, will not allow correlation of data returned from multiple front ends in this way. In order to do that, it is necessary to time stamp each data point, so that it is known on which 15 Hz cycle each data value was measured. To do this, one can use a 15 Hz cycle counter that is known by all front ends. A means of knowing such a

cycle counter is via the multicast clock event message that is sent at 15 Hz following the 0x0F clock event. This clock event occurs at 15 Hz at a time about 19 ms ahead of the time of the Booster minimum magnetic field (BMIN), when the Linac 400 Mev beam is injected. A 32-bit cycle counter is included in this multicast message. The low 16 bits is more than enough to serve as a short term time stamp for correlating 15 Hz data from multiple front ends.

This is a suggested means of adopting a reliable 15 Hz cycle counter time stamp across all front ends. The front end must listen for the multicast message, of course, and it must also receive clock events. As an example, let us examine a front end that begins its 15 Hz cycle execution at 1 ms after BMIN. (Linac front ends operate this way so that they digitize their signals at a time when the gallery is relatively quiet, electrically speaking.) The 0x0F clock event occurs about 20 ms prior to that. Let the low 16 bits of its 32-bit cycle counter be used as the 15 Hz time stamp for data that is measured on the *following* 15 Hz cycle. Recognizing that there can be some variation in delivery time of the multicast message across the complex network architecture, assume that if the front end receives the multicast message within , say, 10 ms of the 0x0F event, it will be valid for use on the following cycle; otherwise, it is assumed invalid. If a valid message is received, its cycle number will be used to time stamp data on the following cycle; otherwise, a cycle number that is one more than the one last used will be used on the following cycle. In the absence of valid multicast messages, a predictable incrementing cycle number will be used as a time stamp. Any front end that is synchronous with the 15 Hz accelerator clock will know how to generate this sequence, given only that it receives a valid multicast message at least occasionally. In reality, valid multicast messages occur almost always.

### ***15 Hz time-stamped data***

In what format should time-stamped data be returned in response to a RETDAT request for such data? Here is one possibility for returning two data values at 7.5 Hz:

```
Count
Time
Data-1
Data-2
```

This format allows for more than two data values to be returned with the `Count` indicating how many are included in the reply, in case one wants to consider slower replies than 7.5 Hz. The `Time` word is the 16-bit cycle number as described above. The value of `Time` is associated with the first data value. The value of `Time` plus one is associated with the second data value.

What can an application do with time-stamped data? It can be used to correlate data returned from multiple front ends. The logic needed to do this may require some ingenuity, but at least it is possible. A cache of recently-received data values from each device to be granted correlated data support will need to be maintained in a kind of application data pool. During a 15 Hz execution in which the application finds that it has values for all devices with equal time stamp values, the application can do its work with those correlated values; otherwise, it must await the next cycle when it can hope that all devices will have matching time stamps. After it uses the correlated data for a given cycle, it can forget about them as it moves on to the next cycle. On some cycles, it may find that two sets of correlated data are available, so that it can do its work with both sets.

Note that even if the Acnet application interface supported callbacks, one would still have to deal with the same kind of complexity to achieve correlated data. The only way to deal with correlated data in a simple way is to “keep up” with 15 Hz by using synchronous operation across front ends and clients. But even with the high clock speeds available with today’s CPUs, modern control system architectures are too complex to be able to do this.

### *Time-stamped data access*

How can the above structure be specified via the RETDAT protocol? A special application is needed to interpret the structure contents, passing raw data values to the PDB support routines to perform scaling to engineering units. Such a non-generic application might use different device names, which would permit use of different SSDNs, but it would be easier to use the usual names. As stated above, accessing single values can be done using a short length and zero offset. Accessing waveform values can be done using long lengths and any offset. Accessing time-stamped data may be done using medium lengths with zero offsets. In the above example, accessing two time-stamped values was done with a request for 8 bytes to be returned at 7.5 Hz. One may want to support slower reply rates, in which case the requested length would be somewhat longer. Suppose a front end maintained the most recent 16 copies of its 15 Hz readings, so that a requester might receive replies at a leisurely pace of 1 Hz. In this case, the request might be for 34 bytes ( $= 4 + 15 \cdot 2$ ) to collect all 15 values.

The scheme alluded to here uses “medium-size” length values to request the special time-stamped 15 Hz data, with zero offset values. The assumption here is that waveform access would specify larger values than, say, 36 bytes. (Using 4-byte data values, even 68 bytes could be considered medium-size.) If a piece of a waveform were requested that is not at the start of the waveform, so that the offset value is non zero, there is still no ambiguity with a time-stamped data request.

### *Time-stamped waveforms*

If a user needed to collect time-stamped waveform data, we have another problem. Collecting 15 Hz waveforms is difficult, and it may require a similar approach to that described above for single 15 Hz values. The front end would have to maintain at least two copies of each waveform, one from the previous 15 Hz cycle and one from the present. Both replies would need to be time-stamped and delivered in a single reply.

Rather than inventing a new term of extra-long-size length, suppose we use an offset value of 2 to indicate to the RETDAT server that time-stamped waveform data is sought. Normally, a large-size length would be used with a zero offset when asking for the waveform data. Using an offset value of 2 to read a waveform except for the first point would seem pointless, so we can ascribe to it the special meaning of returning two (or more) waveforms in a similar structure as described above for returning multiple 15 Hz reading values.

The number of waveforms to return in the reply can be found from the return rate. If the request is a 7.5 Hz request, then 2 waveforms are expected. The specified number of bytes determines how many points are returned for each waveform.

### *Inside the front end*

Having described how to sort out several variations on what forms of data are implied by a RETDAT request, how can the front end support them? The following remarks apply only to the IRM front ends, whose architecture is well known to the writer.

Analog signals are supported in IRMs by a channel-indexed data pool and mini data base that includes scale factors and parameters specifying related analog control and digital status and control. A special nonvolatile table houses channel-related information that is only needed by a few channels. This is how the IRM knows whether a waveform is available for a given channel.

Support for 15 Hz time-stamped data is new. The RETDAT support includes logic for providing averages of 15 Hz data with preference granted to beam cycles, assuming that the request is made for replies slower than 15 Hz. This means that certain logic is invoked at 15 Hz to capture and accumulate the 15 Hz data values that will be averaged. This support may be used to also support access to time-stamped data. In the present averaging support, two long words are used for the

internal pointer structure, rather than the usual one. If the request is for time-stamped data, this 15 Hz opportunity can be used to capture and build up the reply. The first time, the header can be installed in the reply buffer than includes the count, initialized to 1, and the current cycle number as the initial time stamp, followed by the first data value. On each additional invocation, another data point can be added to build up the structure that will be returned on the final cycle. For the case of time-stamped waveforms, a similar approach can be used. On each 15 Hz invocation, the latest waveform can be copied into the reply buffer and the header updated.

### Summary

Here is a list of some possibilities of data accessible via RETDAT using the above scheme. The SSDN only serves to identify the signal about which data is desired.

<i>Period</i>	<i>Offset</i>	<i>Length</i>	<i>Meaning</i>
any	0	2	Reading field from data pool
7.5 Hz	0	8	Time-stamped Readings from two cycles
1 Hz	0	34	Time-stamped Readings from 15 cycles
any	>0	2	Waveform element using byte offset
7.5 Hz	>0	8	Time-stamped waveform elements from two cycles
any	0, any>2	large	Waveform
7.5Hz	2	large	Time-stamped waveforms from two cycles

In addition, one can request snapshot data of a waveform via FTPMAN, where the server will await a selected clock event before capturing the waveform for the client.

### Philosophy

One may determine that the above methods of interpretation of the SSDN is somehow not "pure" enough. But the schemes described here are oriented toward finding a means of providing the support for data access that users require, using the currently-supported protocols. If that means making what may seem to be strange conventions, so be it. The bottom line is to provide support for users. The current Acnet application interface is limited in certain ways, especially with providing reliable support for 15 Hz data that will be increasingly important as accelerator studies are made to improve Booster beam loss, for example. It also is not presently equipped to provide support for data that is correlated across 15 Hz accelerator cycles, even if the data is returned during the same 15 Hz cycle. Although a new Acnet system that is based upon the Java source language is currently under development, it will not be ready any time soon. The current RETDAT support will have to "make do" for some time to come. Issues of providing correlated 15 Hz data in the new system are still under development, so that we may gain some useful experience in trying to solve the problem using the current protocols.