

# GETS32 Protocol Support

*Methodology adopted*

Mon, Oct 30, 2006

The GETS32 protocol is analyzed and discussed in a previous note *GETS32 Protocol Notes*. This note describes how the front end support was built for that protocol.

Rather than developing a local application to handle GETS32/SETS32 protocols, such support was added to that which already exists for RETDAT/SETDAT, so that both protocol sets are supported via the ACReq task. All the old support must also apply to the new protocols; the RETDAT/SETDAT support is not going away.

## *Implementation*

New ACReq task stack variables were defined. New fields were added to the request block header, although its overall layout remains unchanged. The reply message block now includes space for the new reply header. New global low memory variables were defined to hold the cycle time stamp and related variables to aid in constructing the time stamps that are specified for GETS32 replies. The logic allows for a new version of ACLK to be written to supply the correct cycle time stamp as obtained from the multicast event message. In case the cycle time stamp has not been updated, however, the code attempts to create one itself, building on the previous GMT0C support that provides calendar time for any clock event accurate to within 100  $\mu$ s, say.

## *Details of GETS32 support*

Many changes, both large and small, were made in the ACReq task code itself, where RETDAT/SETDAT support is found. Special new functions are found in the module ScanDES, named for the code that scans the data event string used in GETS32 requests, reducing it to a more manageable data structure. New low memory variables were defined as follows:

<i>Name</i>	<i>Size</i>	<i>Meaning</i>
CYCLETSR	8	GETS32 cycle time stamp reference
CYCLETS	8	GETS32 cycle time stamp for current cycle
CYCLEMIC	4	0x0F event time, from cpu $\mu$ s counter
COLLTSO	4	time from 0x0F event until GS32TS is called each cycle, in $\mu$ s

If the ACLK local application is running, it extracts a cycle time stamp, during its processing of the multicast event message, and deposits it into CYCLETSR, where it is seen by the code in GS32TS that is called by the Update task each 15 Hz cycle. Whether CYCLETSR is updated or not, GS32TS leaves the operational cycle time stamp in CYCLETS for use by GETS32 request handling. The CYCLEMIC variable holds a copy of the cpu  $\mu$ s counter captured for the 0x0F event, used for event time stamping. It is used by the function GS32NOW to derive a current time stamp. The COLLTSO variable is used to help derive a collection time stamp for periodic request cases.

New functions in the ScanDES module:

<i>Name</i>	<i>Meaning</i>
CopyDES	Copy data event string from request message into buffer, byte swapped.
ScanN	Convert numeric string into 4-byte long integer value
ConvCyc	Convert from msec units into operating cycles units
ScanDES	Scan data event string, build DESVARS 8-byte structure
ScanFTD	Build DESVARS structure from a RETDAT FTD.
DES2FTD	Build analogous FTD from DESVARS structure.

ReplyDue	Given the request block, determine whether a reply is due on this cycle.
GS32TS	Update the low memory fields relating to the cycle time stamp.
GS32NOW	Get the GMT time stamp for “now”.
GS32FMT	Given a GMT time, compute the time stamp used for GETS32 replies.
EVTGMT	Given a clock event, return the GMT time stamp.

The functions `ScanN` and `ConvCyc` are only used by `ScanDES`. The `GS32TS` function is called by the `Update` task just after the call to `GMTTIME`. (This is the only code change for GETS32 support outside of the `ACReq` module and the functions it calls in the `ScanDES` module.)

### *DESVARS structure*

The data event string included in a GETS32 request message header is interpreted by `ScanDES` and converted into a structure consisting of two 32-bit long words. This structure is found in the request block (type #12) that supports both `RETDAT` and `GETS32`. The `ACReq` code was modified so that it refers to this structure rather than the original `FTD`. To that end, `ScanFTD`, during `RETDAT` request initialization, forms the required structure based upon the given `FTD`. This is analogous to what `ScanDES` does during `GETS32` request initialization. The `ReplyDue` function is called by `ACUpdChk` to determine, for either protocol, whether a reply is due on the current cycle. At points in the code where it is necessary to distinguish which protocol applies to a given request, the sign bit of the first long word is set for `GETS32`.

The first long word field is `DESTYPD`. It has flags in the uppermost 4 bits, a type# in the next 4 bits, and a delay in the low 24 bits. The three type#s currently supported are immediate (0), periodic (1), and clock event (2). For the periodic case, the 24 bit delay is in units of operating cycles, such as 15 Hz. For the clock event case, the 24 bit delay is in units of ms. The only flag bit used is set when a periodic request specifies that an immediate reply is to be returned. (For `RETDAT`, an immediate reply for a periodic request is always returned.) Although other flag bits are used to mark the hard/soft/either option in an event request, they are so far ignored.

The second long word is `DESCNTR`. It contains the clock event# in the upper byte, if it applies, and a “counter” in the low 24 bits. For the periodic case, there is no event#; the low 24 bits are used to hold a cycle countdown to measure the specified period between replies. For the clock event case, the event# is maintained in the hi byte, and the low 24 bits are zero until the specified clock event has been seen, then set to the specified delay in units of  $\mu\text{s}/256$ , or roughly, quarter milliseconds. (The determination of when the specified delay has been reached after the event is done in  $\mu\text{s}$  units, which is why this is useful.) Once the delay has been reached, the low 24 bits are cleared to enable waiting for the next occurrence of the event. If the specified delay is long enough, so that another occurrence of the same event takes place before the delay is reached, no reply will ensue. The test is always based on whether the current time exceeds the most recent event time by the specified delay.

### *Time stamps*

The time stamps defined for use in the reply header for the GETS32 protocol are 64-bit integer milliseconds since the start of the year 1970. Conforming to the endian-ness used by GETS32 clients, this 4-word integer must be returned in reverse word order. To restrict the prevalence of this new format within the system code, the function `GS32FMT`, given a GMT time stamp, computes the required number of milliseconds and deposits the result into the reply message header—the only place it is needed—in the required word order. Recall that the GMT time stamp consists of two 32-bit long words; the first is the number of seconds since the start of the year 1900, and the second is the number of  $\mu\text{s}$  within that second. All GMT times are maintained in this format internally. For the record, the number of seconds between 1900 and 1970 is 2208988800, or `0x83AA7E80`.

Three time stamps are included in a GETS32 reply message header. The first is the “cycle” time stamp that gives the time of the most recent 0x0F clock event, occurring about one 60 Hz cycle ahead of the Booster reset clock event. This time stamp is kept in low memory as CYCLETS and is updated by the call to GS32TS.

The second time stamp is the “collection” time stamp. It normally refers to the time that the data is sampled, which in the present implementation, for all but event-based requests, marks the time that the Update task begins updating the local data pool. For event-based requests, however, this time stamp is built from the event plus delay time, which should occur within one 15 Hz cycle (66 ms) before the final time stamp, which is the reply time stamp, denoting when the reply message is built for queuing to the network. Note that this special version of the collection time stamp may not be bracketed between the other two time stamps, as they are found for the periodic case, say. In particular, it will be found outside this window if the event plus delay is reached before the most recent 0x0F clock event.

The third time stamp is the “reply” time stamp that merely marks the time when the reply message is ready to be sent to the requesting node. It may have some diagnostic value.

### *Server node support*

Server node support means that a request message received that calls for data from more than just the receiving node will require that node to function as a “server node,” which means it will seek to gather all the requested data itself, from however many contributing nodes, and after receiving the replies from each, it will return a composite reply to the requesting node. Since such support is in place for RETDAT, it must also work for GETS32. This is yet another reason why it was decided to add GETS32 protocol support to the existing code that supports RETDAT.

When the replies come back to the server node, from each of the various contributing nodes, the reply data is sprinkled into the appropriate parts of the composite reply buffer. As for the time stamps, the first two time stamps are copied from every contributing node reply header. This means that the first two of the final composite time stamps are the same as the last-received contributing node reply. The reply time stamp is assigned by the server node when it is ready to deliver the composite reply to the original requesting node.

In the case that a contributing node stops returning replies, the server node will remind it, every two seconds, what data it should be replying to any active request. If that node had rebooted, say, then it will “rejoin the fray” promptly after it comes back up.

### *Testing*

Kevin Cahill provided valuable help in testing this new protocol support. Once it was working to return reply messages to simple requests, we examined the time stamps for accuracy. The protocol definition asked for units of milliseconds since 1970, as stated earlier. The front end systems already developed support for GMT time several years ago, so that support was used as a basis to develop the GETS32 times. (This may be replaced by nodes that listen to the multicast events message, once that protocol includes a properly-formatted 0x0F event time stamp.) Most clock events are synchronized to accelerator 15 Hz operation, which is synchronized to the 60 Hz power line frequency. But clock event 0x8F occurs on calendar seconds. By asking for a request specifying the 0x8F event, the collection time stamp should reflect the time of that event, which should fall on exact seconds. After bugs were found and removed, it did. Milliseconds expressed in decimal ended in “000”. For a request specifying the 0x8F event plus 500 ms delay, the time stamp ended in “500”, as expected.