

# SETNSERV Logic Flow

Robert Goodwin

Tue, Nov 4, 2008

Support for the SETDAT protocol in front ends is not so easy to assimilate at first glance. This note is an attempt to give a guided tour through its logic. Basically, SETNSERV is to handle a non-server SETDAT message that is to be executed here in the local node, rather than sent to another node for its execution. The code reviewed here is 68K code, but the corresponding PowerPC code should be very similar.

At first, SETNSERV calls SAALLOC to allocate a setting acknowledgment message block that will eventually be used to deliver reply status on the success of the setting. The data part of the reply message is simply an array of Acnet status words, one for each device being set.

The main part of the code is a loop over all devices to be set, each specified by a setting packet of 16 bytes (SETDAT) or 20 bytes (SETS32) followed by setting data. Peeking at the end of the loop, assuming that we have processed a REQ SETDAT message rather than a USM, the setting acknowledgment message is queued to the network. Finally, just before exiting, the SAFREE routine is called to free the message block in the case that it was not actually used. (If it were successfully queued to the network, the QMonitor task will later free the message block after its reply message been transmitted on the network.)

Now review what happens inside the loop for each device that is to be set. The node# field of the SSDN (second word) is checked for matching the local node#. If it does *not* match, skip to the end of the loop, advance to the next setting packet and check whether it is still within the message size. Assuming it is, then loop back to process the next device. As for why the node# would not match the local node#, it may be that the message was received from a server node that forwarded the original SETDAT message, in which case devices may be included that reside in a number of different nodes. Such packets should be ignored by the local node.

For a local setting, call SFSAMPLE to prepare a record of the (eventual) setting for delivery to a special node for updating the Acnet central database. (It is a special message sent to task "DBM. . ." at Acnet node 0x09EB.) The record needs information from the setting packet, so that information is cached for possible later use, after a successful setting.

The next part of the logic deals with support of the offset field in the SETDAT message. Early in the development of Acnet support, it was found that we needed to be able to effectively modify a device's SSDN, so that one device could provide more flexible access to local data pool or local database information. But modifying an SSDN by an Acnet application is not permitted, since the format of an SSDN is determined by the front end design. So, we opted for a special interpretation of the offset field, which the Acnet user *can* specify. To be cautious, however, an offset options flag must be set in the SSDN to permit making this special interpretation. This logic is handled by the routine ADJOFFS when the options option nibble (mask 0x00F0 on the SSDN first word) is nonzero and the offset field is nonzero.

After checking that the current setting packet plus data fits within the entire setting message, the routine CHKABLK is called to handle cases of the ordinary nonzero offset field for the case that the offset option nibble is zero. This only applies for the listype# 0x02 (in the high byte of the SSDN first word), which is only used for the analog alarm block property. The CHKABLK logic checks that the specified offset field plus the length of the setting data fits within the 20-byte alarm block structure. Then it builds the entire current 20-byte alarm block via a call to PPANABL, then modifies it by copying from the setting data. Then it returns a pointer to the entire 20-byte alarm block, and it sets the setting data length to 20.

It should be noted here that no Acnet alarm block resides in front end memory; it is only used for communication with an Acnet client. A different data structure is used for operation within the front end. It consists of a nominal, tolerance, flags, and trip count word. To make a setting to an analog alarm block, the front end must translate that structure into the 8-byte data structure used internally before it can accept the new data.

After the call to `CHKABLK`, the routine `ADJDATA` is called to prepare the local data structure to be used for making the setting locally, for the cases of alarm block properties only. It calls `MDANABL` for most of this, resulting in a temporary buffer being filled locally, which will be used for executing the actual setting.

As the last step before executing the setting, the routine `IPSETOK` is called to be sure that the source node that sent the setting message is allowed to make settings. That is determined by a special "IP Security" table housed in each front end that consists of entries specifying source IP addresses and relevant masks.

Assuming it is allowed, the setting is finally executed by calling the routine `SETLOCAL`. The only error code that can be returned from that call is a bus error; all other errors have already been checked. The special case of floating point alarm block requires a second call to `SETLOCAL` to install the floating point nominal and tolerance values (using a different `listype#`) into the `FDATA` table fields. This is only used for raw floating point channels.

With no errors, the next step is a call to `SFENTER`, which places the cached setting packet information into a setting acknowledgment message being built for forwarding settings to the central database. Then, for the cases of setting alarm blocks, the Acnet device index (`DI`) from the setting packets is stored into the channel-indexed `ADEVX` system table. The purpose for doing this is to make alarm message delivery more efficient for the Acnet alarm handler `AEOLUS`. If a device index can be specified rather than an `EMC`, it saves `AEOLUS` an extra step. The local application `AERS`, which accepts alarm messages from the `Alarm` task via a message queue, shepherds alarm messages for Acnet. It checks for an existing entry in the `ADEVX` table for the given analog channel. If it finds one, it uses a `DI` rather than an `EMC` code.

Next, following a successful setting, the routine `SETLOG` is called to write an entry to the data stream `SETDLOG`. This provides a diagnostic of recent `SETDAT`/`SETS32` settings. It may be noted that deep inside `SETLOCAL`, another diagnostic data stream queue is written to mark that a setting has been made. It is a data stream called `SETLOG` (without the `D`). This includes settings made by local applications or those made by Classic protocol clients, in addition to those made via `SETDAT`/`SETS32`.

The last step in the loop over devices is a call to `SASTAT`, merely to record the setting status into the `SETDAT` acknowledgment reply message.

If we have not reached the end of the devices to be set, repeat the loop with the next device. Once the end is reached, as stated earlier, `SETACK` is called to deliver the reply message.