

# Debugging Episode

*What about the SSDN?*

Sat, Mar 23, 2002

A PowerPC/vxWorks front end node that had run successfully for 30 days suddenly quit working. So it needed to be reset. A while later it stopped working again. The ACReq task, which supports all Acnet protocols, had been suspended, meaning that the kernel didn't like something it did. We can reset these nodes from afar, using the Classic protocol, so it doesn't necessarily mean someone has to take a walk and physically press the reset button on the CPU board. But it is a disruption, and it certainly looks bad from the Main Control Room.

So, what happened that day that may have contributed to the problem? A new local application was under development. But that did not seem to be the problem, since disabling execution of that LA didn't stop the hang-ups. Removing the program from the system made no difference. Could there be some persistent bug left by executing this new program?

Asking what survives a node reset, the first thing we think of is the nonvolatile memory. Much system configuration data is maintained in nonvolatile (but eminently writable) system tables. Each node runs the same system program, but its particular behavior and personality rely strongly on its configuration parameters that are housed in a set of system tables. An examination of the contents of these tables was made, but nothing appeared to be unusual; there was no evidence that an LA had destroyed something important in the tables.

The LAs that remained in this particular front end, node062E, were the usual suite of LAs that reside in all Linac nodes. But the code for these LAs is maintained by vxWorks as a DOS-format memory file system. If a program was hurt slightly by a bug in the new LA, could it still be allowed to run; does the kernel make use of checksums for verification of file contents? We disabled each of these LAs, deleted each from the file system, downloaded new copies from the library node0619, and re-enabled each one. We hoped that the problem was now eliminated.

Alas, after a time, the system stopped working again. Too bad; it seems that there is no reason to blame nonvolatile memory. But what else could it be?

When the new LA was being tested, two devices were DABBELed into the Acnet database. We wondered whether something might be wrong there. But since that act of entering devices into the database merely causes changes to entries in the nonvolatile ADESC (analog descriptor) table, these had already been looked at, and nothing seemed at all strange in the fields of the affected entries. The dates-of-last-change were checked for all ADESC entries, and only those two channels showed recent modification. Just in case some other entry was hurt by an errant memory write, which would not alter the date-of-last-change, a listing of all such fields was examined, and again, nothing seemed untoward.

Since the two devices were DABBELed, we checked what had been downloaded to the front end, and nothing seemed to be in error, as noted above. But what about the SSDN values associated with each of the device properties?

The SSDN is an 8-byte structure that is stored in the Acnet database for use by Acnet data pool software to request (or set) the property of a device from an Acnet console. The significance of the fields in an SSDN is chosen by each front end programmer to include what is necessary to identify what is being referenced in an Acnet data request. If it is wrong, invalid data

might be obtained from the wrong node. The SSDN values were checked. The target node number (always in the second word of the SSDN for these front ends) was correct—0x062E. Each SSDN looked superficially ok.

But then it was noticed that the SSDN corresponding to the analog alarm block property for one of the two devices was in error. The node number field was ok, but the listype number was wrong. For this property, the listype number is always 0x02. That listype references the nominal value field in the ADATA entry for the device/channel. In that table's entries, the fields following the nominal value are the tolerance value, the alarm flags and alarm trip counter. All alarm fields can be accessed by this listype, so that the Acnet 20-byte alarm block property can be composed in response to a request. The actual (erroneous) SSDN used was

```
0011 062E 0060 0002
```

The correct SSDN for this channel should have been

```
0201 062E 0060 0000
```

In order to explain why an incorrect SSDN could cause difficulty for the front end, we may have to postulate that there is a bug in the handling of an SSDN that was triggered by this particular errant SSDN. Ideally, the software should not rely so strongly on a correct SSDN that the health of the front end is affected when junk is used by mistake. Perhaps we are not careful enough in screening out bogus SSDN fields.

In order to make the system more robust, since the listype number must be 0x02 for an analog alarm block property, we could check for it, and if it does not match the expected value, we could return an error. Examination of such errors, which are logged on the web every day, might reveal that something needs to be re-DABBELed. (An alternative could be to ignore the listype number given, but this approach would not help to get such database errors corrected.) Of course, not all SSDN errors can be detected, but we need to be careful about those that are patently erroneous.

As to what lessons can be learnt by this experience, we now must understand that invalid Acnet database errors can have effects beyond merely downloading incorrect ADESC entry fields. Even when the ADESC entry is ok, the SSDN must still be checked, perhaps carefully, since the SSDN will be acted upon when an Acnet request is made. (For an alarm block, the same SSDN is used when setting the alarm block.) From this particular experience, we need to review the processing of an SSDN to prevent invalid values from causing system-wide problems. A DABBEL user cannot be expected to never make a mistake, as this area of expertise is a bit parochial. But we should try as far as possible to prevent such mistakes from causing errors extending beyond merely returning invalid data values.

Another result from this experience might be improved diagnostics so that unusual activity might be detected at the front-end level. This will take more thought.