Four Mark-5A Tutorials


(1) Record short scans from the test vector generator (TVG) and check them

On the Mark-5 machine, put a disk pack (not write protected) into bank A, and turn on the bank-A keyswitch. We will erase and write on this disk pack, so be sure that there is no data that you want to save on this disk pack in bank A. Put another disk pack (not write protected) into bank B and turn on its keyswitch. We will write on (but not erase) this disk pack in bank B. Then start up the Mark5A program:

```
Mark5A -m 0 &
```

That -m 0 gives debuggery, which is helpful if something goes wrong, but which we’ll ignore in this tutorial, and the & puts Mark5A in the background to allow this terminal to be used also for other things. After a minute or so, when Mark5A is ready, see (the last line):

```
Mark5A Ready. End with EndM5, please
```

Start the tstMark5A program optionally, and for this tutorial preferably, in a different terminal:

```
tstMark5A
  tstMark5A Ready (end with ^C)
>
```

If tstMark5A is run on a different machine, then the tstMark5A command line needs to have the name of the Mark-5 machine (it defaults to localhost). Lines typed by the operator into tstMark5A are preceded by the prompt > and are sent through a socket to Mark5A. The Field System can also do this; use a 'mk5=' prefix.

Check status:

```
> status?
!status? 0 : 0x02300001 ;
```

A ? ends a query. The line beginning with ! is the response to the status query from Mark5A. The first 0 in the response signifies that this query completed OK. The 1 in the far right means ready. The embedded 2 means that disk bank B is ready, and the 3 (that’s 2 + 1) means that disk bank A is ready and selected. (This happens automatically, if possible, on Mark5A startup.) If there had been a pending error, then the ‘!status?’ query response would have printed an error message and cleared the error, if possible, for next time. More information on the ‘!status?’ response is in Appendix A, and detailed information on all the Mark-5A commands and queries is in “Mark 5A command set” cited above.

Which version of the software and hardware do we have?

```
> DTS_id?
!DTS_id? 0 : Mark5A : 2003y202d10h : 1 : mark5-21 : 1 : 1 : 2.5x : 0xb4 : 0x17 ;
```
This says that the Mark-5A software was last diddled on year 2003, day 202, at about 10 o’clock. This machine’s serial number is mark5-21. The corresponding writeup is version 2.5 but with extensions (x). The I/O board has input design revision 0x24 and output design revision 0x17. Certain features depend on these software and hardware versions.

The status query above shows that bank A is selected, but we can also check with:

```plaintext
> bank_set?
```

This says that bank A is selected and has a disk module with a VSN of USN-0052, and bank B is ready but not selected and has a disk module with VSN BKG-0036.

Next let’s erase whatever is on the disk pack in bank A. Be sure that you don’t want to save any data now on this disk pack. Then:

```plaintext
> protect = off
!protect = 0 ;
> reset = erase
!reset = 0 ;
```

Since erase is such a drastic command, a protect=off is needed just before reset=erase (even if protect was already off) to ensure that you really want to erase whatever data are on these disks. That = sign (instead of a ?) after protect and reset means a command (instead of a query). Spaces in command and query lines are optional; I’ve put spaces to be more readable. This erase operation takes only a fraction of a second, and the 0 return means that the command has completed successfully.

Set the mode to test-vector generator (TVG), to prepare to make a TVG recording:

```plaintext
> mode = TVG
!mode = 0 ;
> mode?
!mode? 0 : tvg : : st : : - : 0 ;
```

Many keywords, such as mode, can be either queries (with ?) or a commands (with =). The Mark5A program is not case sensitive (except for file names), so TVG and tvg are equivalent. In the TVG case, only the first and second parameters returned by the mode query are significant.

The internal TVG is on the Mark-5 I/O board, and it’s clock is set by the play_rate command (even though we will be recording rather than playing). The maximum clock rate is determined by the number of disks and the mode; here let’s try 8 MHz (8 Msamples/sec or 8 Mbaud per track):

```plaintext
> play_rate = data : 8.0
!play_rate = 0 ;
> play_rate?
!play_rate? 0 : 8.000 : 8.000 : 8.000 ;
```

In the TVG case, the three numbers from the play_rate query are the same.
Start recording (the scan name, TVG-1, is arbitrary):
> record = on : TVG-1
!record = 1 ;
That 1 in the response means delayed completion; in this case that is recording in progress. Mark5A will continue recording this scan until recording is commanded off (or until space runs out on the disks). Some or all of the lights on the front of the disk pack should now be on, but, since we're recording at only a modest rate, maybe not fully bright.

Some queries work even while recording is in progress:
> status?
!status? 0 : 0x02300049 ;
That 49 is 40 (recording in progress) plus 8 (delayed-completion action in progress) plus 1 (ready) and no known errors. More information on the 'status?' response is in Appendix A.

The position query can be used to track the progress of recording:
> position?
!position? 0 : 320024576 : 0 ;
A little later:
> position?
!position? 0 : 672067584 : 0 ;
That second (big) number is the record pointer; if it does not increase with time during recording, then something is wrong with recording.

Let it record for a minute or so, then end this scan:
> record = off
!record = 0 ;
> record?
!record? 0 : off : 1 : TVG-1 ;
So recording is now off, and we've just made scan number 1, whose name is TVG-1.

At record=off, the play pointer is set to the start of the scan just recorded, so we can check a snippet of this scan with:
> data_check?
!data_check? 0 : tvg : 0 : 249856 : 249856 ;
A data_check?, track_check?, or scan_check? on a TVG scan all give the same result. The three numbers after tvg are the starting word number where Mark5A first found TVG pattern, the word number where TVG pattern ended, and the size of the buffer, all in units of 32-bit words. So the correct answers for these three numbers are 0 for the first, and the second and third should be equal. Although we've tested only a small snippet of it, this looks to be a good TVG scan. We could play this scan, but without a test-vector receiver (TVR), we can't test playback.
Now let's change to bank B:

```
> bank_set = B
!bank_set = 1 ;
Bank_set=inc (meaning increment or, in this case, just change) would have done the same. The 1 return means delayed completion; expect up to a 3-second delay while the bank is switching. During this time, many commands and queries return errors. You can also change banks by turning off the keyswitch on the bank you want and turning off the keyswitch on the bank you don't want. Then check this changeover:
> bank_set?
```

The status query also shows that bank B is now active:

```
> status?
!status? 0 : 0x03200001 ;
```

More information about the status-query response is in Appendix A.

The mode and play_rate settings have not changed, so let's make another short TVG recording, this one onto the disks in bank B:

```
> record = on : TVG-2
!record = 1 ;
```

And after recording for at least a few seconds, then end this scan:

```
> record = off
!record = 0 ;
> record?
!record? 0 : off : 35 : TVG-2 ;
```

So we've just recorded scan number 35, whose name is TVG-2. On record=off, the scan number (scan pointer) defaults to the scan just completed, so:

```
> scan_check?
!scan_check? 0 : 35 : TVG-2 : tvg : 0 : 249856 : 249856 ;
```

This also looks to be a good TVG scan. More about scan_check is in the following section.

If this is a test scan that we do not want to keep, then we could reset=erase_last_scan to delete just this one scan, or we could reset=erase, as above, to delete all scans on the disks in this disk pack.
(2) Record a VLBI scan and copy a prescribed one minute of it to another machine

Mark5A and testMark5A should be running from the previous tutorial. We can use the disks in either bank. Now we want to make a VLBI recording from a formatter, so the formatter needs to be set up, checked, and connected to this Mark-5’s I/O board. We assume that this has been done, but we can also check some parts of this setup and connection. Let’s set Mark5A to record 16 even-numbered tracks from a Mark-4 formatter:

```plaintext
> mode = mark4 : 16
!mode = 0;
> mode?
!mode? 0 : mark4 : 16 : mark4 : 16 : S : 1;
```

In the response to the mode query, the first `mark4:16` is for the input section, the second for the output section of the Mark-5A I/O board. That ‘S’ means that the output section sees the correct format of VLBI data and has Synchronized to it; this is an important check for recording VLBI data. If you see ‘-‘ instead of ‘S’ from a `mode?` query, then whatever is wrong needs to be fixed before recording can be done.

Start recording (the scan name, TestMark4-16, is arbitrary):

```plaintext
> record = on : TestMark4-16
!record = 1;
```

Use the position query to track the progress of the record pointer:

```plaintext
> position?
!position? 0 : 43909526022 : 0;
```

A little later:

```plaintext
> position?
!position? 0 : 44012206180 : 0;
```

If the record pointer stops increasing with time, then something is wrong with recording. And a status query will show certain errors:

```plaintext
> status?
!status? 0 : 0x03200049;
```

This all seems to be OK.

Let it record for a minute or two, then end the recording:

```plaintext
> record = off
!record = 0;
```

Check the scan that we’ve just recorded:

```plaintext
> scan_check?
!scan_check? 0 : 36 : TestMark4-16 : mark4 : 16 : 2003y199d14h35m26.590s : 159.6s : 4.000 : 0;
```
At record=off, the scan number (scan pointer) defaults to the scan just recorded. From left to right: The 0 means that scan_check completed OK; this is scan number 36, whose name is TestMark4-16; its mode is mark4, 16 tracks; it started at year 2003, day of the year 199, 14 hours, 35 minutes, 26.590 seconds UT; it contains 159.6 seconds of data, at 4.000 Megabytes/sec (nominal track data rate); and that last 0 means that no data were lost or added during this scan.

We can also do a data_check starting at the current play pointer, which, at record=off, defaults to the start of the scan just completed:

> data_check?
  !data_check? 0 : mark4 : 16 : 2003y199d14h35m26.590s : 19632 : 0.00500s : 40000 : 0 ;

This data_check response repeats some of the information above. Also the first frame header was found 19632 bytes into the scan, and there are 0.00500 seconds of time and 40000 bytes (on the SS disks) from one frame header to the next.

Since this was a 16-track recording, only even-numbered tracks from 2 to 32 were recorded. If a decoder is available, then we can play this scan and check tracks on the decoder, two at a time. Or we can check tracks on the disks using track_check. Let's first point to track 2:

> track_set = 2
  !track_set = 0 ;

Now a track_check will check this track:

> track_check?
  !track_check? 0 : mark4 : 16 : 2003y199d14h35m26.590s : 19632 : 0.00500s : 4.000 : 2 : 0 ;

This track_check response repeats some of the information above, but the 2 in the next-to-last position shows that track 2 was found correctly. Let's try track 3, which we can get by an inc (for increment), which adds one to the previous value:

> track_set = inc
  !track_set = 0 ;
> track_check?
  !track_check? 0 : mark4 : 16 : 2003y199d14h35m26.590s : 19632 : 0.00500s : 4.000 : 2 D : 0 ;

Instead of track 3, track_check reports, correctly, a duplicate of track 2 (2 D). If the wrong track had been found, then that 'D' would have been a '?'.

Let's try track 32:

> track_set = 32
  !track_set = 0 ;
> track_check?
  !track_check? 0 : ? ;

Oops! That '?' means that track 32 is faulty or was not correctly recorded. In general, a returned blank means unknown; a '?' alone means unknown and also probably an error. Further checking shows that tracks numbered above 16 were not correctly recorded. Something must be done about this!
To use rtime (below), we need to set the play_rate to agree with the formatter’s sample rate (clock rate) divided by the fanout to get a track data rate. (As a check, the formatter’s track data rate is displayed by the decoder, and the scan_check above has also calculated it.) In this case, the formatter was set to 4 MHz with no fanout:

```plaintext
> play_rate = data : 4.0
!play_rate = 0 ;
```

Now how much disk space is left for future recordings?

```plaintext
> rtime?
!rtime? 0 : 54415.0 : 435.321480064 : 90.7 : mark4 : 16 : 4.000 : 64.0 ;
```

This means that we can record approximately 54415 more seconds of data onto the 435.3 Gbytes (which is 90.7% of the total) remaining on these disks provided that we don’t change mode or clock rate. The mode used in these calculations is mark4:16, the track data rate is 4.000 MHz (Msamples/second or Mbaud), and this corresponds to 64.0 Mbaud total recording rate.

Next let’s copy part of this scan to a file on another machine (which might not be a Mark-5 machine). First we need to set up to receive a file on this other machine (fritz, in this case) using the stand-alone program Net2file. To a Linux shell on fritz, try:

```plaintext
Net2file
```

Here we’ve taken all the defaults: The data will be saved in a file named save.data in the current directory. Note that we set up the receiving machine first.

Then, back in the terminal with testMark5A:

```plaintext
> scan_set = : 36m : +1m
!scan_set = 0 ;
```

After the = sign in this command, we could abbreviate the scan name, provided that there is no preceding scan with the same abbreviation in its name, or, in this case, leave out the scan name because it defaults to the last scan recorded. The second parameter in this command sets the start byte to the first (and in this case only) occurrence of 36m0s in this scan. (Otherwise we might have to calculate and type those eleven- or twelve-digit numbers.) We can verify this start-byte calculation by another scan_check, which reads data starting at the play pointer (same as start byte in this case):

```plaintext
> scan_check?
!scan_check? 0 : 36 : TestMark4-16 : mark4 : 16 : 2003y199d14h36m00.000s : 60.0s : 4.000 : 0 ;
```

The third parameter in the scan_set command above (+1m) also sets the end byte to one minute (the 60.0s above) later in time than the start byte. Thus we are ready to write one minute of data from 14h36m to 14h37m to, in this example, a Linux file on the other machine. To send this off, we first connect to this other machine, whose name is fritz:

```plaintext
> disk2net = connect : fritz
!disk2net = 0 ;
```

And verify that we’re ready to send:

```plaintext
> disk2net?
!disk2net? 0 : connected : fritz : 43619737088 : 0 : 44099737088 ;
```
Those two big numbers are the starting and ending byte numbers as calculated and set by scan_set above. Start the transfer:

```bash
> disk2net = on
!disk2net = 1 ;
```

The 1 response means delayed completion; in this case, data transfer is in progress. As time passes, we can check how we're doing:

```bash
> disk2net?
!disk2net? 0 : active : fritz : 43619737088 : 43907571199 : 44099737088 ;
```

That middle number, the so-called now byte, increases and will equal the end byte when the transfer is complete:

```bash
> disk2net?
```

Then, since this is all that we want to send just now, end the transfer:

```bash
> disk2net = disconnect
!disk2net = 0 ;
```

As a result of this disconnect, on the target machine (fritz), you should now see:

```
Net2file: The End
```

On fritz, try:

```bash
ls -l save.data
```

And see:

```
-rw-rw-r-- 1 jball users 480000000 Jul 18 12:01 save.data
```

File save.data contains the prescribed one minute of data. As an exercise for the reader, transfer this file to the SS disk(s) on a Mark-5 machine using File2net on fritz and net2disk in Mark5A; then use scan_check to verify good data and the prescribed times.
(3) Check a recorded disk pack

Here we check a disk pack already recorded and either ready to be sent to a correlator or already at a correlator. Start up Mark5A and testMark5A and check status as above. The mode doesn’t matter, and we don’t need a formatter or decoder because we don’t intend to either record or play. DirList is a stand-alone program to be run from a Linux shell prompt on the Mark-5 machine. It prints the contents of the directory as read by Mark5A. Here is an example:

DirList
nscans 227, n 226, recpnt 657003363992, plapnt 0, playRate 8.000 MHz

<table>
<thead>
<tr>
<th>start byte</th>
<th>end byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 188-1700a</td>
<td>81952 1603782808</td>
</tr>
<tr>
<td>2 188-1704a</td>
<td>1603864760 3529321832</td>
</tr>
<tr>
<td>3 188-1706b</td>
<td>3529403784 7021613440</td>
</tr>
<tr>
<td>4 188-1709a</td>
<td>7021695392 8979248240</td>
</tr>
<tr>
<td>5 188-1711</td>
<td>8979330192 10584384520</td>
</tr>
<tr>
<td>6 188-1713a</td>
<td>10584466472 15581564320</td>
</tr>
<tr>
<td>7 188-1716</td>
<td>15581646272 17186440760</td>
</tr>
</tbody>
</table>

Note that we ran DirList after starting Mark5A and after the correct disk pack is active; otherwise we might have seen the directory from a different disk pack previously in this machine. This disk pack has 227 scans, numbered from 1, with names and byte positions as listed. This tabulation can be compared with the schedule (or over) to check for missing scans, and the starting and ending byte numbers can be checked against the log (or hveX) or even used to replace incorrect numbers.
Let's check the first scan:

> scan_set = 1
!scan_set = 0 ; > scan_check?
!scan_check? 0 : 1 : 188-1700a : mark4 : 32 : 2003y188d16h59m50.097s : 50.1s : 8.000 : 0 ;

As in the tutorial above, reading from left to right, this scan check is OK, on scan number 1, whose name is 188-1700a, the mode is mark4:32, and this scan started at year 2003, day of the year 188, 16 hours, 59 minutes, and 50.097 seconds UT, the scan lasted 50.1 seconds, the nominal track data rate was 8,000 Msamples/second, and 0 data were lost or added. Since this scan's name is 188-1700a, we expect it to start on day 188 at about 17:00 UT, and it actually started about 10 seconds early. You might want to verify these parameters; for example, was this scan really intended to last only 50 seconds?

We can check a sequence of these scans as follows:

> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 2 : 188-1704a : mark4 : 32 : 2003y188d17h03m56.062s : 60.2s : 8.000 : 0 ;
> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 3 : 188-1706b : mark4 : 32 : 2003y188d17h06m21.080s : 109.1s : 8.000 : 0 ;
> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 4 : 188-1709a : mark4 : 32 : 2003y188d17h08m50.057s : 61.2s : 8.000 : 0 ;
> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 5 : 188-1711 : mark4 : 32 : 2003y188d17h10m57.062s : 50.2s : 8.000 : 0 ;

What's done here is to type a command, scan_set=inc, which increments the scan number, and a query, scan_check?, on the same line with a semicolon as separator. Then this dual-purpose line can be repeated, as shown, by typing just an up-arrow and Enter for each scan to be checked: An up-arrow copies the previous line and Enter sends it again through tsMark5A to Mark5A. Thus we have quickly checked scans 2 through 5 with only two keystrokes per scan.

We can spot check also some scans from near the end of this disk pack:

> scan_set = 224 ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 224 : 189-0454 : mark4 : 32 : 2003y189d04h54m42.095s : 50.2s : 8.000 : 0 ;
> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 225 : 189-0456 : mark4 : 32 : 2003y189d04h56m46.097s : 50.2s : 8.000 : 0 ;
> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 226 : 189-0502 : mark4 : 32 : 2003y189d05h02m17.105s : 50.1s : 8.000 : 0 ;
> scan_set = inc ; scan_check?
!scan_set = 0 ; !scan_check? 0 : 227 : 189-0504 : mark4 : 32 : 2003y189d05h04m05.098s : 94.2s : 8.000 : 0 ;

Note that we first put scan_set=224 to point to somewhere near the end, then repeat the dual-purpose line as above. These all seem to be OK.
Let's also do at least spot checks of the tracks in a scan (the last scan in this case) using a similar scheme. Tracks are numbered from 2, and all tracks from 2 to 33 should have been recorded. So try:

```plaintext
> track_set = 2 ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 2 : 0 ;
> track_set = inc ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 3 : 0 ;
> track_set = inc ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 4 : 0 ;
> track_set = inc ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 5 : 0 ;
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 30 : 0 ;
> track_set = inc ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 31 : 0 ;
> track_set = inc ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 32 : 0 ;
> track_set = inc ; track_check?
!track_set = 0 ; !track_check? 0 : mark4 : 32 : 2003y189d05h04m05.098s : 41904 : 0.00250s : 8,000 : 33 : 0 ;
```

The track number, as before, is the next-to-last parameter returned by track_check. Rechecking scans and tracks is desirable after any change in mode or in formatter configuration. Here the tracks seem to be OK, and we have not found any problems with the data on this disk pack.

(4) Condition and check the disks in a disk pack

New disks should be conditioned before their first use especially if they are to be used anywhere near their maximum data rates. Conditioning can be done by a three-pass procedure: Write, read, and write, each through the full length of the disk. Or a special procedure inside SSErase can be used to condition up to sixteen disks at a time using a two-pass procedure. This saves lots of time, but it's still slow.

**Important:** Note that any operation with SSErase, as the name implies, erases any data that may be on the SS disk or disks.

For the following procedure, Mark5A should not be running. Shut it down, if need be, using EndM5. Mount the disk pack to be erased and conditioned in bank A and turn on the keyswitch. In the following example, we have a two-pack of 120-Gbyte disks. Start SSErase:

```
SSErase -m 0 -c 1
```

The optional `-m 0` turns on lots of debug printing, the `-c 1` turns on conditioning (otherwise it just erases). (SSErase should not be started in the background (no `%`) because it asks one or more questions of the user.) Then see:

```
SSErase DEBUG: msglev set to 0
SSErase DEBUG: cond set to 1
SSErase DEBUG: XLRDeviceFind() OK
```

SSErase DEBUG: Trying to XLROpen() ...
SSErase DEBUG: XLROpen() OK
SSErase DEBUG: XLRSelectBank() to A OK
SSErase DEBUG: Bank A, Label USN-0040/988/1024Played,
    Length 1573379800, STATE_READY, Selected,
    PowerRequested, PowerEnabled,
    MEDIASTATUS_NOT_EMPTY, NOT WriteProtected, NO error
SSErase DEBUG: Calling checkDisks() for bank A
SSErase checkDisks() DEBUG: Found 2 SS disk(s); OK
SSErase checkDisks() DEBUG: SS directory Length 1573379800,
    AppendLength 1573379800, Full 0
SSErase checkDisks() DEBUG: Setting fill pattern
SSErase checkDisks() DEBUG: XLRSetOption() drive statistics
SSErase checkDisks() DEBUG: OK
SSErase DEBUG: Bank A is not write protected
SSErase DEBUG: Bank B, Label LABEL NOT AVAILABLE,
    Length 0, STATE_NOT_READY, NOT Selected,
    NOT PowerRequested, NOT PowerEnabled,
    MEDIASTATUS_EMPTY, NOT WriteProtected, NO error
SSErase DEBUG: XLRSetOption() drive stats OK
SSErase DEBUG: XLRSetDriveStats() OK
SSErase: Are you sure that you want to erase and condition the disks in bank A? (Y or N) y
SSErase DEBUG: Erase start OK
SSErase NOTE: This will run a long time
SSErase DEBUG: In progress, 120263409664
SSErase DEBUG: In progress, 116897742848
SSErase DEBUG: In progress, 113533059072
SSErase DEBUG: In progress, 110189543424
SSErase DEBUG: In progress, 106890330112
SSErase DEBUG: In progress, 103591575552
SSErase DEBUG: In progress, 100290068480
SSErase DEBUG: In progress, 96988430336
SSErase DEBUG: In progress, 93775331328
SSErase DEBUG: In progress, 90605879296
...
SSErase DEBUG: In progress, 15752429568
With `-m 0`, the "In progress" debug prints occur about once a minute so that you'll know that the program hasn't gone comatose. Here we've skipped most of these prints. Conditioning these two disks took about 92 minutes. The eight "STATS" numbers are stats (see get_stats) for these two disks and are within the normal range. Larger numbers in the right-most (longest-time) three bins would indicate a possible problem. The stats after conditioning usually look worse than the stats for the same disks after normal recording.

If, after conditioning a set of disks, you break them up into a different configuration, then an additional erase step is required. In particular, if you condition two eight-packs at a time (sixteen disks), then do an additional SSErase (without the `-c 1` conditioning) on each pack in bank A separately. A similar SSErase is needed for each part of an eight-pack that is split up into individual disks or two-packs. This extra SSErase (without `-c 1` conditioning) will take only a few seconds in part because SSErase has no reset at its end. When you're all done with SSErase, then we recommend an SSReset.
Appendix A: Summary of return from 'status?' query

0x00000001  Ready
0x00000002  Error (message may be appended)
0x00000004  (Not used)
0x00000008  Delayed completion action in progress
0x00000010  Delayed completion request in progress
0x00000020  Disk-FIFO-only special mode
0x00000040  Recording in progress
0x00000080  Media full and/or recording halted
0x00000100  Playing in progress
0x00000200  End of scan (scan_play) or end of data; playing halted
0x00000400  Recording throttled (can't keep up, some data loss)
0x00000800  (Not used)
0x00001000  Disk2file active
0x00002000  File2disk active
0x00004000  Disk2net active
0x00008000  Net2disk active or waiting
0x00010000  In2net sending (on)
0x00020000  Net2out active or waiting
0x00040000  (Not used)
0x00080000  (Not used)
0x00100000  Bank A selected
0x00200000  Bank A ready
0x00400000  Bank A media full or faulty (not writable)
0x00800000  Bank A write protected (not writable)
0x01000000  Bank B selected
0x02000000  Bank B ready
0x04000000  Bank B media full or faulty (not writable)
0x08000000  Bank B write protected (not writable)

Revised: 2005 May 07, JAB