Frequently Asked Questions

This collection of frequently asked questions (FAQ), with answers, applies to the `Mark5A` program and several associated programs used with the Mark-5 VLBI disk-recording system as designed at MIT Haystack Observatory. Documentation for this system is in: http://web.haystack.edu/mark5/command5a.pdf and http://web.haystack.edu/Mark5/AuxProg.pdf. The release notes (updated frequently) are in: http://web.haystack.edu/Mark5/UpdateMark5.html. The Conduant StreamStor documentation is in: http://www.supportcenteronline.com/dmfiles/639/669/Support%20Downloads/Product%20Documents/StreamStor%20Manuals/StreamStor Manual.pdf and http://www.supportcenteronline.com/dmfiles/639/669/Support%20Downloads/Product%20Documents/StreamStor%20Manuals/Simultaneous StreamingAddendum.pdf. This FAQ repeats some parts of these documents.

Contents

Q: I'm a newbie in the Mark-5 game; could you give me a simple introduction for getting started using Mark-5 machines?

A: Perhaps, but we'll need to assume that you know at least some simple things about Linux operating systems (OSs) and Linux shells.

Mark-5 computers have a login for `oper` (operator), which you should use except when you need root privileges. The password for `oper` should be available from your system administer or from the documentation that came with your machine.

Mark-5 programs can be run from the console, if you have one, or from a terminal emulator perhaps on some other computer such as the Field System (FS) computer at field stations or `ccc` at correlators.

For initial testing after a reboot, with a disk module in bank A, you probably should try the StreamStor (SS) check program:

```
ssopen
```

If this program does not report success after a few seconds, then the problem, whatever it is, will need to be fixed first. (Do *not* use sstest; it writes over VLBI data, if any.) The Mark-5 operational program is:

```
Mark5A  -m 0  &
```
where the `-m 0`, which produces lots of debug printing, might be omitted after you've become familiar with normal operation. And the `&` (background) could be omitted if you have another terminal on the Mark-5 machine to be used for `EndM5` (below). After `Mark5A` is running, it should be able to respond to socket requests from the FS or `suman` on the correlator.

`Mark5A` commands and queries and the responses thereto are documented in http://web.haystack.mit.edu/mark5/command5a.pdf. If you need to send manual commands and queries to `Mark5A`, for example to swap banks or for testing, then you can start:

```
tstMark5A  <machine>
```
which accepts any typed `Mark5A` commands or queries and sends them to `Mark5A` on *<machine>*, which, if omitted, defaults to localhost. Responses to commands and queries from `Mark5A` are returned by `tstMark5A`. (But if you start `tstMark5A` in the same window as `Mark5A` with debuggery on and `&`, then you might become confused between the responses from `tstMark5A` and debuggery from `Mark5A`.)

The StreamStor system is persnickety especially with respect to mechanically swapping disk modules and shutting down properly. Such swapping is best done when there is no activity on either disk module: First turn off the keyswitch. If the "Ready" light was on, then wait a few seconds for it to go out. Carefully pull down the latch and pull out the disk module. Carefully slide in the new disk module and pull up the latch to seat it. Then turn the keyswitch back on. If `Mark5A` is active, then the "Ready" light should come on within ten to fifteen seconds.

The proper shut-down of the `Mark5A` program is as follows:

(1) First close all socket connections by ending any programs that have socket connections to `Mark5A`. For `tstMark5A`, a `<Ctrl>C` (interrupt) works; for the FS, try `mk5close`, which might already be in your FS schedule. For the correlator, you'll need to `endall` (sorry). If you fail to do this *first*, you might find that socket m5drive is in limbo and can't be used again until after a reboot.

(2) Then

```
EndM5
```
This script sends an interrupt to the main thread (only) of `Mark5A`. Do *not* use just `<Ctrl>C` to `Mark5A` because this sends interrupts to all threads and creates a mess.

Q: Which operating systems work with Mark-5 software?

A: We have experience with various Linux versions including RedHat 7.2, 7.3, and 9; these work well. We also have some working Fedora (RedHat), GenToo, and Ubuntu-Debian systems, but these required some minor modifications to the install and update scripts. We plan to try also with SuSe and to make available an Ubuntu-Debian version. The stand-alone programs `tstMark5A`, `Net2disk`,

and `Disk2net` have been tested also on HP-UX and are intended to work on a wide variety of other operating systems with perhaps some modification to the compile lines in `cc5A`.

Q: Ubuntu? What's that?

A: Ubuntu is a new (2004) Linux operating-system (OS) distribution based on the older Debian distribution. Development of Ubuntu is being sponsored by Canonical, Ltd., of South Africa. Canonical was founded and is being funded by Mark Shuttleworth of "space tourist" fame. Ubuntu, with headquarters now on the Isle of Man (between England and Ireland), includes many of the world's best known and talented Linux, Gnome and Python developers scattered in several countries around the world.

So what does "ubuntu" mean? The official version, from the Ubuntu Web site, is "Ubuntu is an ancient African word meaning 'humanity to others' or 'I am what I am because of who we all are.'" There are several unofficial versions including, "United we stand," "I learn from the group; I am supported by the group; and I contribute to the group," "We're all in this together," and "I dig hangin' out with my friends."

Q: How do I update Mark-5 software? What if my Mark-5 machine can't talk to Haystack?

A: Shut down all Mark-5 programs, then execute the script `Mark5Update` as root. You should already have this script, which does the whole job provided that it can download the tar file from Haystack, but watch for error messages. If your Mark-5 machine can't talk with Haystack, then try this three-step process: (1) download the tar file from haystack, http://web.haystack.edu/Mark5/Mark5A.tgz or ftp://web.haystack.edu/dist/mark5/Mark5A.tgz; (2) transfer `Mark5A.tgz` to `/tmp` on your Mark-5 machine; then (3) execute the script `Mark5Update.dev` (the `dev` is important) as root. This should finish the job, but watch for error messages.

Q: How do I know what changes have been made since the previous releases of Mark-5 software?

A: The whole point of the release notes (http://web.haystack.edu/Mark5/UpdateMark5.html) is to allow users to judge whether any of the bug fixes, improvements, and changes therein described impact whatever they're doing with `Mark5A` and its associated programs. Much of our work on `Mark5A` is in response to criticisms and requests from users. We try to document in these release notes all the changes that users might notice. Updates from Conduant (XLR StreamStor library functions) might contain changes that we don't know about but might discover through testing. The release notes also contain as much as we know about Conduant updates.

Q: What is Conduant?

A: Conduant, with headquarters in Colorado, is the company that designed and manufactures the StreamStor (SS) card and many other parts of the Mark-5 system including the library of software (XLR) functions. Conduant now sells complete Mark-5 systems comprising also the Haystack-designed parts.

Q: What's an XLR error?
A: This is an error that was identified in one of the XLR functions from the Conduant XLR library. See "What is Conduant?" above, and see the Conduant documentation in the preface above.

Q: I need to edit text files on Mark-5 machines, but I don't know how to find them nor how to use either the `vi` (`vim`) or `emacs` editors. Help!
A: Some of the operations in the initial installation of Mark-5 machines involve creating or finding and editing text files. If you are not familiar with either of the standard Linux editors, `vi` or `emacs`, and don't care to learn about them, then you might want to look into Midnight Commander (`mc`), which is a Linux imitation of the old Norton Commander from DOS. `mc` is run mostly with function keys and arrow keys in text mode, so it can be used even without X (i.e., you don't need a GUI), and `mc` contains a simple and easy-to-use text editor. Try it first just to look around at what's on the disk; type `mc` to a Linux (shell) prompt. Some Mark-5 machines have `mc`; some don't. Or, for a very simple and easy to use editor, try `nano`, which is an augmentation of `pico`, the editor inside the pine email program. First try `man nano` for help.

Q: There seem to be several variants of `Mark5A` with version 2.6.3. What's the story?
A: "2.6.3" is a *documentation* version number; the *software* version is a date code such as, for example, `2004y364d16h` from `DTS_id?`. Often there are several software updates substantially conforming to each version of the documentation.

Q: My Mark-5 machine doesn't know the correct date and time for my time zone. How can I fix it?
A: Most of the following operations must be done as root. The local time zone is determined by the file `/etc/localtime`, which should be a symbolic link such as

```
localtime -> /usr/share/zoneinfo/UTC
```

You may change this link to any of the similar files found in `/usr/share/zoneinfo` or its many subdirectories. Changes in this link take effect immediately.

To have your Mark-5 computer's clock set to a nearby Network Time Protocol (NTP) server on reboot, look at the file `/etc/rc.d/rc.local`. Near the end, you should find a line such as

```
/usr/bin/ntpdate -b -p 8 -u gauss
```

or

```
/home/jball/bin/ntpdate -b -p 8 -u gauss
```

where *gauss* is the name or IP address of an NTP server. Use your editor to change *gauss* to the name or IP address of a nearby and accessible NTP server. See http://ntp.isc.org/bin/view/Servers/ if in doubt. Check that you can reliably `ping` the chosen NTP server, and execute this `ntpdate` line manually to check that it works. If it doesn't work, check that your firewall isn't blocking NTP traffic. The `rc.local` file is executed on reboot.

To have your Mark-5 computer's clock set by NTP each day, look in the directory `/etc/cron.daily` for a file named `ntpdate` or `timeupdate`. If you have such a file (the name doesn't matter), and it contains an `ntpdate` line such as above, then edit it to replace *gauss* just as you did above. If such a file does not exist, then make a file `ntpdate` and type a line:

```
#!/bin/bash
```

(with that # in column 1) followed by an `ntpdate` line as above. Save this file and change the permissions on this new file:

```
chmod 0755 ntpdate
```

It should be owned by root. It will be executed once a day.

Q: How do I know when recording hits the end of the disk(s) (end of medium)?

A: A `record?` query returns `halted` (instead of `on` or `off`) when the recording has halted because of end of medium. Also the record pointer as reported by the `position?` query stops incrementing.

Q: How do I know how many bytes and how much time remains to be recorded on each of the disk modules.

A: An `rtime?` query returns the estimated recording time left on the active disk module (in seconds) as its first parameter. This time is accurate provided that the `input_mode`, `play_mode`, and `play_rate` parameters are correctly set. An `rtime?` query also returns the bytage (GBytes) remaining to be recorded (on the active bank) and the percent remaining. A `bank_info?` query returns an estimate of the bytes remaining on each of the two banks. Also the module serial numbers (VSNs), as read by either `bank_set?` or `VSN?`, are intended to contain the total bytage for the disk pack.

Q: What should I do if recording hits the end of the disk(s) (end of medium)?

A: Assuming that `bank_switch` is `off`, this gives an obvious problem: The last scan will be short by the time from end of medium to end of scan. Be sure to do a `record=off` to end this last scan. Then check a `DirList` *after* this last scan but *before* switching to or mounting any new disks. See also `bank_switch` in the documentation.

Q: What happens when playing hits the end of a scan?

A: Not much. Think of it as a bump in the road; playing rides over the bump and goes on to the next scan (if there is a next scan).

Q: How do I know when playing hits the end of the recording?
A: The previous answer applies to the end of a scan, but at the end of the last scan (end of recording), playing halts, and a `play?` query returns `halted` (instead of `on` or `off`). After `play=off` in this case, a `position?` query shows (approximately) equal record and play pointers.

Q: What are `ssopen` and `sstest`, and what are they good for?
A: These stand-alone programs from Conduant are two of the sometimes-useful auxiliary programs to accompany the Mark-5 system. ssopen does a quick test of the SS card to verify readability, and sstest *erases* and *writes* a scan of 33554432 bytes of SS pattern onto the SS disks. *Important*: Do *not* use `sstest` on disks that have data to be saved.

Q: What is `tstMark5A`, and what's it good for?
A: This stand-alone program accepts commands and queries as defined in the `Mark5A` software documentation, sends them to `Mark5A`, and prints the responses. Try:

```
tstMark5A [ machine ]
```

where *machine* defaults to localhost. `Mark5A` should already be running on *machine*, but `tstMark5A` can run on some other machine including most Linux and HP-UX machines and probably others.

Q: What is `SSErase`, and what's it good for?
A: This stand-alone program erases and, optionally, conditions up to 16 disks in SS disk modules. `SSErase` is useful to test the write-ability of disks and to prepare disks to be recorded. `Mark5A` must *not* be running. *Important: `SSErase` erases all data on SS disks.* If a disk pack is write protected, then `SSErase` asks permission to remove this protection as is necessary to erase or condition. Try:

```
SSErase [ -m m ] [ -c c ] [ -h ]
```

where *m* (msglev) is the debug message level, range -1 to 3, default 1, *c* sets conditioning (`0` for FALSE, `1` TRUE, default FALSE), and `-h` (alone) prints a help message.

If *c* is 1 (TRUE), then `SSErase` also goes through the long conditioning process on whatever disks it can access, up to 16 at a time in both banks. If *m* is set to `0`, then debug prints about once a minute during conditioning to show what's happening.

Conditioning disks is recommended before recording especially if they are to be recorded at or near their maximum data rate. Conditioning amounts to a write-read-write cycle through the whole set of disks, but `SSErase` does this in two rather than three passes. Printed with debuggery and counting down twice is the number of bytes per bus. With an 8-disk pack, for example, this count starts at a quarter of the total capacity, which is twice the capacity of a single disk.

Conditioning one 120-Gbyte disk takes about 101 minutes; two 120-Gbyte disks (as two masters), 103 minutes; four 120-Gbyte disks (as four masters), 111

minutes; eight 120-Gbyte disks (an eight pack), 157 minutes; and sixteen 120-Gbyte disks (two eight packs), 278 minutes. 200-Gbyte disks take about $^{5}/_{3}^{rds}$ as long—no surprise. Conditioning one 200-Gbyte disk takes about 160 minutes; eight 200-Gbyte disks, 286 minutes; and sixteen 200-Gbyte disks, 465 minutes. These times are approximate and for the ideal case, but, if any of the disks has a problem, then times can be much longer. After conditioning two disk packs at a time (i.e., more than eight disks), or in any case where you change the disks into a different configuration, then you should also `SSErase` (without `-c 1`) each disk pack separately before recording. This takes only a few extra seconds.

Q: Why can't I run `SSErase` in the background?
A: This is a Linux "feature" with any program that asks questions of the operator. Following suggestions from our users, `SSErase` always asks because of the possible error of accidentally erasing valuable data. We recommend running `SSErase` in its own terminal (e.g., an `xterm`), not in the background, and using another terminal if necessary for other business. But, of course, you can't run `Mark5A` at the same time.

Q: At the end of a conditioning (with `SSErase`), the `get_stats` of disk *n* stand out. What should I do?
A: If your disk *n* looks suspicious, and if you can spare the time, then try reconditioning this pack. If disk *n* doesn't improve, then try exchanging it with the disk manufacturer or supplier.

Q: What is `SSReset`, and what's it good for?
A: This stand-alone program performs an `XLRCardReset()`, which often helps extricate the SS system from a no-fair state. `Mark5A` must *not* be running; try `SSReset` *before starting* `Mark5A`.

Q: What is `DirList`, and what's it good for?
A: This stand-alone program prints an ASCII version of the scan directory written by `Mark5A`. Try:

```
DirList [ -m m ] [ -f filename ] [ -h ]
```

where *m* (msglev) is the debug message level, range `-1` to `3`, default `1`, *`filename`* sets the name of the Mark-5 directory file, default `/var/dir/Mark5A`, and `-h` (alone) prints a help message. `DirList` reads the Mark-5 directory and lists the contents including the starting and ending byte numbers of all completed scans. `DirList` can run simultaneously with `Mark5A`, in which case the listing will be up to date except for any scan being recorded at the time. If `Mark5A` is dismounted (i.e., no disk pack active), then the listing will be appropriate for the SS disk(s) that were in place before the dismount. Or if `Mark5A` is not running, then the listing will be appropriate for whatever was happening last time `Mark5A` was running. This is equivalent to saying that

`DirList` does not read SS disks, instead it reads whatever `Mark5A` has written into `/var/dir/Mark5A` or *`filename`*.

Q: What are network (electronic) VLBI data transfers (eVLBI) good for?

A: The most common use of Mark-5 eVLBI data transfers involves moving scans from the SS disks on one Mark-5 machine to the SS disks on another (distant) Mark-5 machine.

The scans on the source machine were probably recorded using the VLBI Field System (FS), and a log of this operation, which includes starting and ending byte numbers for each scan, is kept by the FS. This FS log is converted and used by correlators to process these data. `Mark5A` running in the source Mark-5 machine also keeps such information in a directory both on the SS disks and in a Linux (OS) file, `/var/dir/Mark5A`. This file can be converted to ASCII and viewed using the stand-alone program `DirList`. This directory is useful for cross checking and in case the FS log is flawed or lost.

One mode of eVLBI transfers involves moving a whole group of scans from the source machine, using `disk2net`, to the target machine, using `net2disk`. If the SS disks on the target machine were initially empty (i.e., erased), and if the block of scans to be transferred are continuous and begin at zero byte position, then the starting and ending byte numbers in the FS log and in the source Mark-5 directory will be correct also for the target machine. This is usually very desirable; the correlator setup should be unchanged.

If, instead, the starting byte position on the target machine is not zero, but the transfer is otherwise the same, then all byte positions on the target machine will need to have this starting byte position added as an offset.

Note that, in both these cases, the directory on the source machine (`/var/dir/Mark5A`) is useful and probably should be transferred to the target machine. This transfer is to be done in a separate step, perhaps using `ftp`, and does not occur automatically. Also `Mark5A` running on the target machine does not know about this directory, so operations such as `scan_set?` and `scan_check?` cannot work.

Another common use for eVLBI data transfers is to move just a few selected scans or perhaps only parts of scans from field stations to correlator machines. This operation is often recommended to test for problems at the start of VLBI experiments. In this case, creating a new scan on the target machine for each transferred scan or part of a scan is desirable. One scheme would be to use, on the target machine, `net2file=open`, with a scan name, followed, after a transfer, by `net2file=close` for each scan or part of a scan transferred. Then the newly created Mark-5 directory on the target machine would show this scan name and its starting and ending byte numbers, so that `scan_set=...` and `scan_check?` should work on such scans or parts of scans.

Similar possibilities apply in case either the source or target machine have or will have VLBI data in files on OS disks. In this case, try the stand-alone programs `File2net`, on the source machine, or `Net2file` on the target

machine. The files on the OS disk can be made to contain multiple scans, single scans, or parts of scans. The same notes apply with respect to byte offsets into these files.

Q: What about the rumor that there are undocumented features available on the `in2net` and `net2out` command lines? Why aren't these documented?

A: The undocumented additional buffer-size parameters on the `in2net` and `net2out` command lines have been or will be removed. Don't try to use them. This functionality has been augmented and transferred to `net_protocol`; use `net_protocol` instead.

Q: How should I determine good values to set the `net_protocol` parameters?

A: We're working on documentation with suggestions for optimizing these buffer sizes (network tuning parameters) as a function of measured network speeds and round-trip delays. As a heuristic trial, set the "socbuf size" in `net_protocol` to the delay-rate (delay-bandwidth) product, that is the round-trip delay times the data rate for this network connection, converted to bytes. Use `ping` to approximate this delay; use just trials to approximate a data rate. This product is the number of bytes that fit into the round-trip "pipe" from sender to receiver. As an numerical example, for a network link with an effective data rate of 8 Mbaud (10-Mbaud ethernet) or 1 Mbytes/second and a ping time of 100 ms, try a "socbuf size" of 100000 bytes.

But there's more: There seems to be a significant speed advantage to picking the largest possible MTU size (typical maximum: 1500 bytes or 9000 bytes for so-called jumbo frames) consistent with a data-payload size that is an integer multiple of 8 (to please StreamStor). The data-payload size is the MTU minus the IP header with its options (typically 20 bytes) minus the TCP header with its options (typically 32 more bytes for a total of 52 header and option bytes). Set the MTU size in bytes (as root) using `/sbin/ifconfig eth0` *MTU*, where `eth0` might be `eth1` if you're working with Gbaud ethernet.

Then also try picking both buffer sizes in `net_protocol` to be integer multiples of this data-payload size and also integer multiples of 4096 bytes (memory page boundaries) for efficient memory-to-memory transfers.

Here is a numerical example: Assume jumbo frames, maximum MTU 9000, and choose MTU=8996 bytes, which (usually) gives a data-payload size of $8996-52=8944=1118\times8$. And suppose a ping time of 100 ms and a data rate of about 900 Mbaud (Gbaud ethernet) or 112.5 Mbytes/second for a delay-rate product of about 11 Mbytes.

Then $2289664=256\times8944=559\times4096$ bytes is an interesting number. Try setting "socbuf size" to $5\times2289664=11448320$ (a little more than to 11 Mbytes) in `net_protocol`. If "nbuf" stays at 8, then the maximum "workbuf size" is 134217728/8=16777216, so we might choose "workbuf size" to be the same as "socbuf size" in `net_protocol` in this case. An alternative would be to reduce "nbuf" to 4 and set "workbuf size" to twice "socbuf size".

Q: What are `Net2file` and `File2net`, and what are they good for?

A: These are two stand-alone programs intended to exchange data with Mark-5 machines but primarily intended to run on computers other than Mark-5 machines. The source code contains hints for compiling on Linux and HP-UX. Try:

```
Net2file [ filename ]
```

which accepts a connection from a Mark-5 machine and writes the received data to *`filename`* or to save.data if *`filename`* is blank. This file will be created if necessary or appended. Start `Net2file` *first*, then command `disc2net` or `in2net` in `Mark5A` in the Mark-5 machine. Monitor progress with

```
ls -l filename
```

but this will lag the actual progress because of buffering. `Net2file` will end when the Mark-5 machine disconnects or with `<Ctrl>C`, after which *`filename`* will be ready to use.

Or try:

```
File2net machine [ filename [ startbyte [ endbyte ] ] ]
```

which sends a file or part of a file to a Mark-5 machine. Command `net2disc` or `net2out` in `Mark5A` in the Mark-5 machine *first*, then start `File2net`. *`Filename`* defaults to `save.data`, *`startbyte`* defaults to `0`, and *`endbyte`* defaults to the end of the file. `File2net` ends when the prescribed transfer is done.

Q: Can `Net2file` and `File2net` run on machines other than Mark-5 Linux boxes?

A: Yes. The latest versions of `Net2file.c` (2004 August 19) and `File2net.c` (2004 March 9) compile with `gcc` on Linux or HP-UX and also with `cc -Ae` on HP-UX and run properly. `Net2file.c` will not compile with `g++` without some mods. If you are using some other compiler or operating system, then you might need some modifications. We have never tried on Windows or Macintosh. Sorry.

Q: Just how does `skip=...` work during `net2out`?

A: This does not seem to be documented, but here's how we understand it: Where in the FIFO the write and play pointers are is determined by how much delay there is between starting writing and starting playing and also by any speed difference between writing and playing. In the usual case, the writing and playing speeds are the same, the network can keep up, and reading (over the network) starts as soon as possible, so the offset between write and play is very small—maybe just milliseconds. This means, for example, that a significant forward skip is not possible. But you can ask for negative skips, which will allow the pointers to separate and the output to delay and come later with respect to clock time. Temporarily slowing the play rate would do the same. We typically run our "real-time" correlations a few seconds behind clock time in order to allow at least

some possibility for both forward and backward skips to align stations. If, for some reason, you need to have a significant part of the FIFO buffer between write and play pointers, for example to accommodate big skips or variable network speeds or delays, then you'll need to start the correlations that much delayed with respect to clock time.

Q: Must a disk module be in place for `in2net`? Why?
A: The release notes (http://web.haystack.edu/Mark5/UpdateMark5.html), under "2004 March 23" say, "Many of the problems with `in2net` and `net2out` seem to be fixed with this Conduant update, but you'll still need a scratch-disk module." But beginning with Conduant "FirmwareVersion 10.79, FirmDateCode Nov 09", the scratch disks are no longer needed.

Q: What is disk FIFO, and what's it good for?
A: **Summary:** Disk FIFO is a special operating mode of `Mark5A`, selected at startup by `-d 1` on the command line, and which allows a scratch disk module to be used to augment the SS RAM FIFO buffer during `in2net`. Since a disk FIFO can typically store many hours of VLBI data, using `in2net` with disk FIFO can allow the network transfer to lag by up to many hours from real time. This is useful at Field Stations with slow network connections but who wish to transfer data to correlators as quickly as possible. The network transfer in this case continues regardless of new data flowing (i.e., `in2net on` or `off`) from I/O board to FIFO.

In more detail: Use disk FIFO with `in2net` to transfer to disks (OS or SS) on remote (target) machines especially when network connections are slow compared to the selected data rates as set by formatter configurations and input-board modes. Use ordinary (default) RAM FIFO with `in2net` for transfers directly to correlators (using `net2out`) with network connections that are fast enough to keep up with almost-real-time at the selected data rates.

Disk-FIFO-only mode requires a scratch disk pack in bank A. If this is an 8-pack, then up to half-speed formatter output (512 Mbaud) will be supported. Erase this disk pack (using `SSErase` or `Mark5A`'s `reset=erase`) *before* restarting `Mark5A` in disk-FIFO-only mode as follows:

```
Mark5A -m 0 -d 1 &
```

The `-m 0` and the `&` are optional, as usual. The `-d 1` causes disk-FIFO-only mode as required for this operation. When `Mark5A` is running in this mode, a `status?` query will show (among other things) `0x20`, "Disk-FIFO-only special mode," and many of the normal `Mark5A` commands and queries will return errors and not function. In this mode, `in2net` will use the disk pack as (most of) its FIFO.

You can run `Mark5A` from `tstMark5A` or from the Field System, as usual. Except for the FIFO size, `in2net` commands and queries operate as usual. A typical sequence might be:

(1) Start the receiving program, for example `Net2file`, on the target machine.

(2) Set and check the formatter configuration and the input-board data mode with the `mode=...` command and `mode?` query especially to verify input board connected and data synchronized.

(3) Connect to the target machine using `net_protocol=...`, if necessary, and then `in2net=connect:...` commands.

(4) Start and stop each scan with `in2net=on` and `in2net=off` commands.

(5) Log the byte position before the start of each scan (or at the end of each scan) using an `in2net?` query (instead of a `position?` query as in ordinary recording).

This log will be less precise than `in2net?` in normal (RAM FIFO) mode because there is up to one SS block (65528 bytes) lost. But `data_check?` or `track_check?` at the correlator starting near the logged positions will give exact answers.

To go back to normal operation after you've finished with disk-FIFO-only mode, you'll need to shut down and restart `Mark5A` without `-d 1`. The scratch disk pack will also need to be erased (again) for its normal operation.

Q: What about this rumor of a "development" or alpha-test version?

A: The so-called developmental version (alias dev or sandbox version) of `Mark5A` and its helper programs has been offered only to selected sophisticated users. These modifications have not yet been officially released, but sometimes there is a tarball of a dev version at ftp://web.haystack.edu/dist/mark5/Mark5A.dev.tgz. Changes from the last published version are briefly described in http://web.haystack.edu/Mark5/Mark5AUpdateNotes.html. If you intend to make any use of `Mark5A.dev.tgz`, then please read `Mark5AUpdateNotes.html` first. If you are intrepid and willing to help us with testing, then to update a Mark-5 machine to this dev version, first download `Mark5A.dev.tgz`, put it into `/tmp` on the target Mark-5 machine, change its name to `/tmp/Mark5A.tgz`, and then execute the script `Mark5Update.dev` (February-27th version; the `dev` suffix is important), which you should already have. If not, get also ftp://web.haystack.edu/dist/mark5/Mark5Update.dev. Make this script executable, if it isn't already, and execute it as root. Comments are solicited. Note that both `Mark5A.dev.tgz` and `Mark5AUpdateNotes.html` might change from day-to-day as we work on them; check the dates on these files to see whether you have the latest versions. If you don't like this Mark-5 development version, then you can return to the last publicly released version using the standard `Mark5Update` script.

Q: What is this strange `^^Played` at the end of a `VSN?`

A: Under "2004 July 13 (day 195)" in the release notes, see: "WARNING: If a module is processed through recording, playing or erasing with this version (or

later) of Mark5A or SSErase and later read into an earlier version of Mark5A, then the DMS code will appear to be appended onto the VSN of the module in the results from the bank_set? and VSN? queries. (There will be no disk_state? query in this case.) If you are processing these results in a script or program, then allow buffer space for the extra characters." The ^^ is ASCII 30 (decimal) also known as RS or Control-^ and is set and read as '\036' (36 octal).

Q: I've lost information about what was recorded on a Mark-5 disk pack. How can I recover such information?

A: There is a directory file, written by Mark5A as soon as it selects a new or changed disk module, in /var/dir/Mark5A. This file can be read and converted to ASCII by the stand-alone program DirList (directory list). The output of DirList comprises a line for each scan with scan number, scan label, and the starting and ending byte numbers (or *bytage* by analogy with *footage*). At Haystack-designed correlators, the output of DirList is needed only if a station's experiment logs are flawed or missing. DirList is not, of course, a substitute for a scan_check? on each scan, but it shows the bytage to be used for such queries. DirList shows only a few special kinds of problems such as missing scans or zero-length scans.

Within a scan, you can calculate the bytage from the time or the time from the bytage from just one log entry showing both bytage and time (plus knowing, of course, the prescribed byte rate). This is done routinely at Haystack-designed correlators. We recommend, nevertheless, that field stations log redundant information: the starting time and bytage and ending time and bytage for each scan. This logging can be done, even during continuous recording, using the position? query and also noting the time. A similar log can be made with in2net sending and net2disk receiving using the results of an in2net? query (instead of a position? query). If the receiving disks were not initially empty in this case, then a starting byte offset needs to be added to all the bytages. At Haystack-designed correlators, this information should be transferred to the lvex file for the experiment, perhaps using vlogx. Thus disk bytage corresponds to tape footage (and is somewhat more precise); the logging recommendations are the same. The log should also, of course, contain the scan labels.

Q: Why can't I make up a disk pack with five disks?

A: You can make up a disk pack with one disk (in position 0), two disks (preferably in positions 0 and 2, i.e., two masters), three disks (in positions 0, 2, and 4), four disks (preferably in positions 0, 2, 4, and 6, i.e., all masters), six disks (in positions 0, 1, 2, 3, 4, and 5), and, of course, eight disks (filling all positions). But five or seven disks in a disk pack is an illegal configuration, which Mark5A checks for and, if so, goes into a read- or play-only salvage mode on the assumption that there is a bad disk. SS refuses to record in this case. One of the

rules is: If any master has a slave, then all masters must have slaves. Thus there is no way to make a legal five-disk module.

Q: What about this rumor that Conduant shipped some Mark-5 machines with 110-MHz oscillators on the I/O board instead of the correct 100 MHz? How can I tell which oscillator I have?

A: You could, of course, look at the I/O board and read the label off the oscillator. There is no direct way to do this with software, but the following indirect methods might work. When you make a VLBI recording, the formatter clock determines the data rate; but when you play it back, the I/O-board clock, derived from this I/O-board oscillator as set by `play_rate=...`, determines the data rate. By comparing the time for recording with the time for playing (look for 10% less), you could check the I/O-board oscillator against the formatter clock. This is awkward but should work.

The data rate of a TVG recording, however, is determined by this I/O-board clock as set by `play_rate=...`. So the number of bytes recorded in a timed test should show the rate set by `play_rate=...`, and a 10% increase in byte count should be easy to see. Try this: Set `play_rate=...` to a standard value (or use the default, 8 MHz). Set `mode=TVG`, and record a carefully timed several-minute-long scan. (A shorter scan will make accurate timing more difficult if you're using just a wristwatch.) Then `scan_set?` or `DirList` will show the starting and ending byte numbers of this scan, and the length of the scan (end byte minus start byte) should agree with the `play_rate?` setting times 4 bytes/sample times duration of scan in seconds. (Start with an erased disk pack to avoid the long subtraction.) If your number is 10% high, then, presumably, you have a 110-MHz oscillator instead of the standard 100-MHz. Maybe try this first with a known 100-MHz oscillator.

Note that this I/O-board oscillator is rarely used at field stations, so you might not need to be concerned about a wrong frequency.

Q: To erase a disk pack, I used the prescribed sequence: `protect=off; reset=erase`, but the `protect=off` failed. What's wrong, and what should I do?

A: When you start with an unused disk module or one that has had one or more disks replaced, then the `protect=off` sequence usually won't work presumably because the protect flag differs from one disk to another. But that's OK in this case because the following `reset=erase` will work and will also fix the protect flag. After that, `protect=on` or `protect=off` will work. Try it. `SSErase` also works on such disk packs.

Q: A `record?` query sometimes returns "halted", "throttled", "overflow", or "waiting". What do those mean, and what should I do about them?

A: A "halted" return means that the end of medium (end of disk space) was hit during recording with `bank_switch=off`. Have a look at `bank_switch` in

the documentation. The other three, "throttled", "overflow", and "waiting", are all error conditions.

The "throttled" error originates in the Conduant StreamStor (SS) card, which sets the suspend line on the FPDP bus whenever it is, at least temporarily, unable to accept data on the FPDP bus for recording. Conduant calls this status *throttled*, and it occurs whenever the disks are unable to keep up with the clock rate (e.g., from the formatter) and the selected recording mode. But throttled status can occur also in other situations such as flawed or missing disks. The IO board detects the suspend-line status and sets the suspend flag (SF) for `Mark5A` to read. When in recording mode, `Mark5A` periodically polls for SF, shows the throttled status in bit `0x400` of the `status?` query return, and causes the "throttled" error on a `record?` query. In this case, recording has stopped, at least temporarily, and some data were lost.

When the SS card is not configured for recording, the FPDP suspend line is set, but that's OK, and `Mark5A` ignores it. On `record=on:...`, the suspend line is reset by the SS card on the first clock pulse, provided that recording is possible. But if there is no clock, then the suspend line remains set, causing SF, and `Mark5A` debuggery will show "Mark5A WARNING: Recording throttled (can't keep up)". The lack of a clock (e.g., from the formatter) also causes `Mark5A` debuggery to show "Mark5A WARNING: Record pointer not incrementing," which seems incongruous, but, in this case, both WARNING messages are caused by the same situation, namely lack of a clock.

If you see these two WARNINGs while trying to record VLBI data, then the most likely problem is the formatter misconfigured, such that it is not putting out a clock, or a cabling problem, such that the clock is not reaching the Mark-5 IO board. There are also other, less likely situations, such as a failure in the IO board, that might cause these symptoms. Try making a TVG recording, which uses the IO board's internal clock and tests operation of the IO board.

An "overflow" error from a `record?` query corresponds to an almost identical situation, that is, the clock frequency, mode, and number of disks are such that recording can't keep up, but now reported by overflow in device status from a Conduant function. This also sets bits `0x82` and the message "Overflow" in the return from `status?`. In our experience, "overflow" from `record?` is almost always preempted by "throttled" above.

Finally, a "waiting" error from `record?` indicates that recording seems to have stopped for unknown reasons. This error should never happen.

Q: `Mark5A` has the wrong year! Doing `scan_check?`, `data_check?`, and `track_check?` on recently recorded data gives year `1995` instead of `2005`. What's wrong?

A: Check the Linux clock (date and time) in the Mark-5 machine. (Try `date -u` to see UTC.) An incorrect Linux clock setting is the most likely cause of this error. This incorrect year can also occur with a recording made with the

formatter clock offset into the future, accidentally or deliberately—the formatter might have been set ahead of real time in order to test run a future schedule.

The Mark-4 frame header contains only the last digit of the year, so `Mark5A` determines the year by calculating the most recent date and time consistent with the data having been recorded in the past (with respect to the Linux clock). This gives the right answer provided that the data were recorded not more than ten years ago, provided that the formatter had the correct date and time, and provided that the Linux clock is not significantly slow. The VLBA frame header contains the last three digits of the Julian Day Number instead of a year. In this case also, `Mark5A` calculates the most recent year consistent with the data having been recorded in the past. This works provided that the data were recorded not more than 1000 days ago and with the other provisos above.

Q: The "Ready" light on a disk module is flashing. What does that mean? What should I do?

A: According to Conduant, the "Ready" light should blink off and on whenever no recording is possible. This means, for example, that the disk pack is not yet initialized (as in `reset=erase` or `SSErase`) and so does not have a valid directory. Or recording was in progress and reached the end of medium, that is, trying to record off the end of the disk pack, or it was already recorded all the way to the end. End of medium during recording causes lots of warning messages on the debug display if you started `Mark5A` with `-m 0`.

The "Ready" light will also blink whenever there is a faulty or missing disk: Reading or playing might be possible in this case, but recording probably is not. A flashing "Ready" light during SSErase is not a problem.

A steady-on "Ready" light means that the module is ready to record.

Q: An interruption (e.g., power failure, program crash, operator turned off the active-bank keyswitch, operator tripped over a cable) occurred during `recording` or `net2disk`. Can (part of) this last scan be recovered?

A: Maybe. If `Mark5A` is still running, turn the keyswitch back on, and issue `record=off` or `net2disk=close` as appropriate. With `Mark5A` running (again), try `recover=0` as described in the documentation. Then note or log the record pointer (from `position?`) because this bytage might not be otherwise logged. The directory (e.g., from `DirList`) might show the broken scan with an appended + as if it were a continuation scan; but this directory entry might show a length of `0`, and its ending bytage is incorrect.

After `recover=0`, if you do another recording, it will work well, the directory entry for this new scan will be correct, but the directory will show a gap between the (incorrect) bytage for the end of the broken scan and the (correct) bytage for the start of this next scan.

Q: `WRSpeedTest` or `sstest` was run accidentally, and valuable data were overwritten. Help!

A: Some of your data are gone, but data after 999424 bytes (default) with `WRSpeedTest` or after 33554432 bytes with `sstest`, can (usually) be recovered using `recover=1` as described in the documentation.

Q: What should I look for in the `get_stats?` return?

A: Here is an example of the `get_stats?` output from a good set of eight disks after about 147370 blocks read or played at the Haystack correlator:

```
!get_stats? 0 : 0 : 143543 : 3825 : 1 : 1 : 4 : 0 : 0 : 0 : 0 ;
!get_stats? 0 : 1 : 143740 : 3622 : 2 : 3 : 3 : 0 : 0 : 0 : 0 ;
!get_stats? 0 : 2 : 143930 : 3442 : 1 : 1 : 3 : 0 : 0 : 0 : 0 ;
!get_stats? 0 : 3 : 143918 : 3449 : 4 : 1 : 5 : 0 : 0 : 0 : 0 ;
!get_stats? 0 : 4 : 143842 : 3527 : 2 : 1 : 1 : 1 : 0 : 0 : 0 ;
!get_stats? 0 : 5 : 143869 : 3503 : 1 : 0 : 4 : 0 : 0 : 0 : 0 ;
!get_stats? 0 : 6 : 143924 : 3448 : 4 : 1 : 4 : 0 : 0 : 0 : 0 ;
!get_stats? 0 : 7 : 143924 : 3454 : 1 : 1 : 3 : 0 : 0 : 0 : 0 ;
```

Here is a similar example from a less-good set of disks after about the same number of reads and plays:

```
!get_stats? 0 : 0 : 143450 : 4861 : 57 : 141 : 98 : 2 : 0 : 0 : 0 ;
!get_stats? 0 : 1 : 138046 : 7809 : 341 : 810 : 1264 : 298 : 40 : 7 : 0 ;
!get_stats? 0 : 2 : 143987 : 4476 : 20 : 62 : 49 : 1 : 0 : 0 : 0 ;
!get_stats? 0 : 3 : 141848 : 5537 : 143 : 515 : 487 : 60 : 5 : 0 : 0 ;
!get_stats? 0 : 4 : 143724 : 4824 : 34 : 105 : 79 : 13 : 0 : 0 : 0 ;
!get_stats? 0 : 5 : 143752 : 4906 : 18 : 62 : 33 : 3 : 0 : 0 : 0 ;
!get_stats? 0 : 6 : 130822 : 4049 : 18 : 36 : 13 : 2 : 0 : 0 : 0 ;
!get_stats? 0 : 7 : 130244 : 4273 : 67 : 176 : 164 : 17 : 0 : 0 : 0 ;
```

All the disks in this second example seem to be sporadically slower that than the ones in the previous module, and number one is particularly suspicious. This disk module is, nevertheless, in active use and is not generating recording or playing errors even at the maximum recording speeds (1024 Mbaud) and even near the end of the medium. Following is an example of a module with a significant problem affecting the last four of the eight disks:

```
!get_stats? 0 : 0 : 26077 : 25928 : 424 : 0 : 2 : 0 : 0 : 0 ;
!get_stats? 0 : 1 : 26087 : 26332 : 9 : 3 : 1 : 0 : 0 : 0 ;
!get_stats? 0 : 2 : 26069 : 26308 : 37 : 1 : 3 : 0 : 0 : 0 ;
!get_stats? 0 : 3 : 26052 : 26333 : 9 : 4 : 16 : 2 : 0 : 0 ;
!get_stats? 0 : 4 : 25608 : 25902 : 45 : 31 : 428 : 145 : 103 : 124 ;
!get_stats? 0 : 5 : 25831 : 26181 : 34 : 36 : 177 : 60 : 35 : 34 ;
!get_stats? 0 : 6 : 25531 : 24032 : 135 : 444 : 571 : 529 : 420 : 650 ;
!get_stats? 0 : 7 : 25425 : 25838 : 54 : 74 : 481 : 230 : 117 : 93 ;
```

The system in this third example required repairs to the backplane.

(These three examples are from earlier versions of `Mark5A` that had no replaced-block count nor SMART state at the end of the `get_stats` output.)

Revised: 2005 September 29, JAB