Atacama Large Millimeter / submillimeter Array

APP Mark6 Recorder Test Procedures

ALMA-05.11.50.02-0001-A-PRO

2014-11-18

Prepared by:	Organization Role	Date and Signature
G. Crew	MIT	
Approved by:	Organization Role	Date and Signature
Authorized by:	Organization Role	Date and Signature

Change Record

Version	Date	Affected Section(s)	Author	Reason/Comments
A	2014-11-18	All	G. Crew	First Issue

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

This document describes procedures for testing the Mark6 recorders. Such tests are in general aimed at either testing the recorders themselves, or for testing other parts of the system (*e.g.* the OFLS units or PICs).

1.2 Scope

This document is limited to describing various testing procedures and some theory of operations background information. It does not cover "VLBI" expert procedures. There is additional information in the set of manuals provided with the recorder system. Current copies are maintained at http://www.haystack.mit.edu/tech/vlbi/mark6/documentation.html beyond those supplied to EDM at the time of Acceptance.

Almost all of the tests mentioned here are intended to be performed by the APP team. Other documentation will provide non-test procedures.

1.3 Reference Documents

The following documents contain additional information, are referenced in this document, and should be consulted for further, more detailed information.

Table 1.1: *Reference Documents*

Reference	Document Title	Document ID
[RD1]	APP Update to Corr/Control Design	ALMA-05.11.61.01-0001-A-DSN
[RD2]	Mark6 Command Set	rev 1.2
[RD3]	Getting Started with your Mark6	rev 1.0.1
[RD4]	Mark6 User's Guide	rev 1.0
[RD5]	Mark6 Usage Examples	rev 1.0

1.4 Acronyms

ALMA Atacama Large Millimeter/submillimeter Array

APP ALMA Phasing Project

CCL Control Command Language

DBE Digital Back End

DiFX Distributed FX (Software Correlator)

EDAC Error Detection And Recovery

EP Engineering Port

MIT Massachusetts Institute of Technology

NTP Network Time Protocol

OFLS Optical Fiber Link System

PIC Phasing Interface Card

PNG Portable Network Graphics

TFB Tunable Filter Bank

VEX VLBI EXperiment file

VDIF VLBI Data Interchange Format

VLBI Very Long Baseline Interferometry

VOM VLBI Observing Mode

VSI-S VLBI Standard Interface for Software

Chapter 2

Operational Background

This section provides some context or theory of operation for the recorder.

2.1 VLBI Background

The Mark6 recorder is an improvement on the previous generation of Mark5 recorders developed as a replacement for tape devices which were used for many years with VLBI hardware correlators. (The Mark4 hardware correlator was the last of the line of VLBI hardware correlators developed at Haystack and used tape recorders.) In VLBI systems, the recorders record the "raw" baseband data at prodigious rates so that data from distant VLBI stations may be correlated at the VLBI correlator.

The Mark6 recorder is a commercial (COTS) hardware system with (open) software developed at Haystack with an eye towards some compatibility with the previous generation of Mark5 recorder, and general adoption in the community. The detailed command set is explained in [RD2]; but in normal operations, the recorders are operated transparently through the ALMA control system as explained in [RD1].

The embedded system is a Debian flavored UNIX operating system, which is the least problematic from an intellectual property or export control viewpoint. It is supplied with current software and security and (since it is an embedded system with extremely limited user access) it does not *require* updates to stay "current".

The recorder is provided with one user account (`oper`) which is necessary for the testing as described in this document. Certain operations must be performed as the super-user (`root`). However, once installed and tested, the recorder is completely controlled through a service interface socket connection.

A point of nomenclature: the recorders are known on the Control system side as Recorder1 through Recorder4 (with Recorder0 as a spare). However, the vendor, Conduant, labels each host with a number in series from 4000. The recorders at ALMA are 4004 through 4008. The expansion chassis also are numbered; this time starting with 5000. The recorders don't actually need or care what expansion chassis they work with, but we have arranged at ALMA that Recorder1 is 4005&5005, Recorder2 is 4006&5006, *&c.* .

For operations, the recorders need to be time-synchronized using NTP so that recordings are made at the proper time. Note, however, that in most cases, the underlying (high-precision) timing arrangements are made by the digital back end (DBE) which provides the data to the recorder in the form of time-tagged (VDIF) network packets.

A final comment is that historically, tapes were notorious for generating relatively high bit errors. (It really *is* a technical challenge to record to tapes at high rate; error rates at the 10^-4 level were typical.) As a consequence, most of the VLBI processing of the baseband data is somewhat tolerant of errors. Modern disks on the other hand are designed to work with systems that are far less forgiving. So the disk controllers in the disk hardware usually incorporate a certain amount of error detection and recovery (EDAC) and therefore *real* errors are rare, unless the disk is failing. Thus, as a general rule, it is not necessary to check every bit of a recording—failures, when they

occur are generally rather gross.

2.2 Implementation

The embedded application running on the recorder was (for a variety of reasons) partitioned into two applications:

`cplane` the higher-level server application (implemented in Python) providing a command interface, and

`dplane` the lower-level utility application (implemented in C) to manage the data operations

Generally speaking, one operates the recorder by making a socket connection (the default configured port is `localhost:14242`) to `cplane` and making one or more so-called VSI-S commands. A simple client (`da-client`) is provided for this purpose. One may also connect to `dplane` directly via a utility program (`dboss`) for certain operations.

2.3 Filesystem

After logging into the recorder as `oper`, all work is done in that user's home directory. It usually contains several subdirectories:

```
oper@Mark6-4005:~$ ls -l
drwxr-xr-x  2 oper oper  4096 Oct 18 15:09 bin
drwxr-xr-x 10 oper oper  4096 Oct 25 00:58 bmr
drwxr-xr-x  4 oper oper  4096 Oct 26 01:24 difx
drwxr-xr-x  2 oper oper  4096 Oct 26 00:49 logs
drwxr-xr-x 17 oper oper  4096 Oct 26 00:49 test
```

These are employed as follows:

bin This directory is in the user `PATH` and contains executables or test scripts such as might be required for some of this testing. These objects are outside of the normal Mark6 product (which is installed in normal system areas, *e.g.* `/usr/bin`).

bmr This directory contains special test software that is built and installed into ∼ `oper/bin`).

difx This directory is a work area for using the DiFX correlation software which was installed for testing the recorded data. This has subdirectories `data` and `test` for managing the correlation data and running the (test) correlations.

logs This directory is used to collect logfiles from various test sessions (for further analysis as required).

test This directory is where testing activities are concentrated. Some output is directed to a directory `out` to avoid clutter.

In particular, after login, `cd /test` before doing anything else.

2.4 General Usage

In normal usage, the recorder is operated according to a schedule supplied by the ALMA control system. However, for test purposes, it is relatively easy to make "manual" recordings. The Mark6 recorder is designed to be rather flexible, so to do so (connect via `da-client`):

- the recorder must be suppied with viable media (enough modules for the desired recording rate) (this uses the `mstat?` query)
- the recorder must be informed which modules to write to (this uses the `group` command)

- the recorder must be informed which network interfaces are supplying data and what the packetization structure is (this uses the `input_stream` command)
- the recorder can then record immediately for a specified recording duration (`record` command), or
- the recorder can make a recording of specified duration at a specified time (`record` command)
- the recorder can (after the recording) verify the integrity of the recording (using the `scan_check` query)

These commands may be issued manually to `da-client` when doing low-level tests. When the control system is active, there is a CCL script `VLBIRecorderSetup.py` that performs this setup in the way expected for normal operations.

Chapter 3

Installation

Installation of the Mark6 recorders is generally covered in [RD2] and [RD3]. Rails to support the recorders in the racks were provided, and the screw alignments (recorder chassis to rails) is straightforward.

One detail to pay attention to is that the cables which connect the recorders to the disk modules are routed through a 1U divider section which is usually placed between the recorder host section (with module slots 1 and 2) and the expansion chassis (hosting modules 3 and 4). Proper attention must be paid so that the pairs of cables to each module are attached (as explained in [RD3]) through the named slot. After the recorders are booted, this should be verified by noting that the disk serial numbers of each module appear in the appropriate slot.

It was found helpful to exceed the instructions of [RD3] by wrapping colored (red and yellow) electrical tape around the cables. Also labelling the connectors on the back is helpful. When cables are connected to modules, yellow goes on top and red on the bottom.

Figure 3.1: *Recorders 1 and 4 installed in the designated OSF computer room rack. The OFLS demux unit is at the top. Between the recorders is the network switch which connects the recorders to the AOS net and also provides a private network between the recorders.*

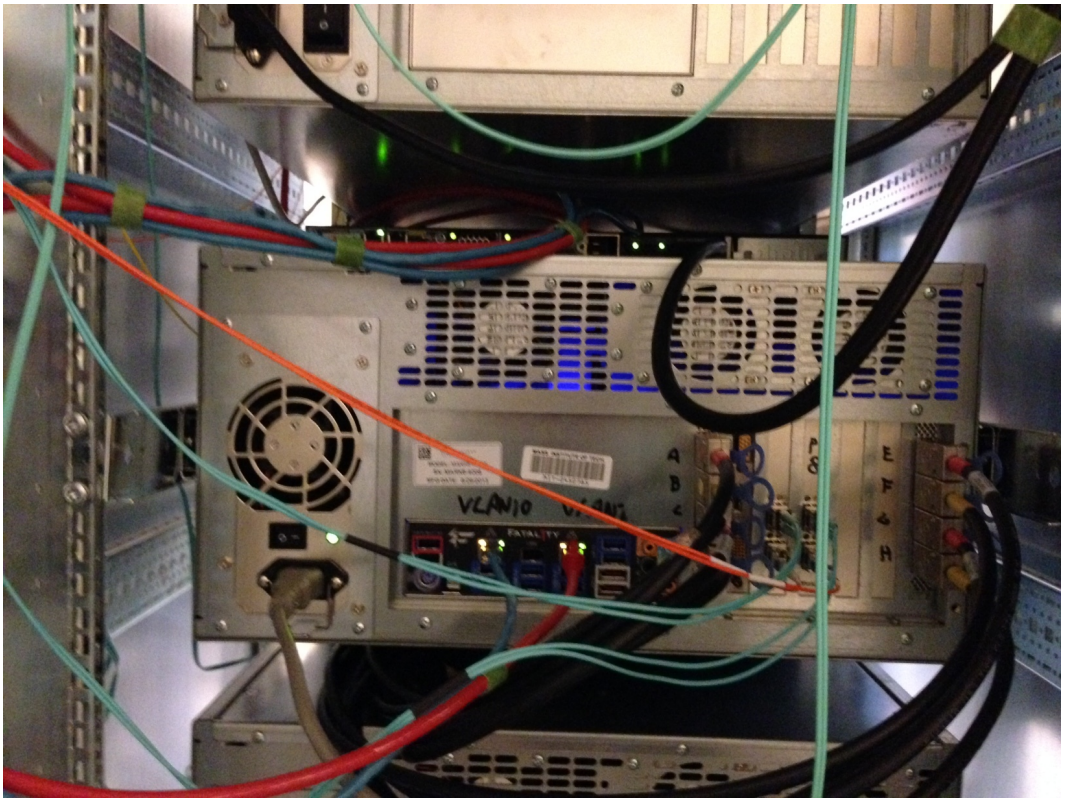

Figure 3.2: *Back side of the Mark6 recorder.*

Chapter 4

Startup, Shutdown and Module Handling

4.1 Startup

There are several power switches on the Mark6 recorder. The ones on the front are rocker-type for on/off. The ones on the back (0|1) are for the power supply. In the event of a power outage, the module keys should be turned to the off (vertical) position. Then:

1. Verify that all module present are unpowered (key in vertical position)
2. Verify that the back switches are in the powered position (1)
3. Press and hold (a second or two) the expansion chassis rocker switch
4. Press and hold (a second or two) the recorder host chassis rocker switch
5. Wait for the recorder to boot (two minutes)

At this point it is safe to power the modules. Turn them on one at a time (the largest current draw is during power-up to get the disks spinning) in slot order (1, 2, 3, and 4):

1. Turn the key to the on (horizontal) position
2. Notice that green LEDs (one per disk in the module for a total of 8) come on a few times
3. Proceed to the next module

The keys should remain in the locks, but they may be withdrawn in either the on or off positions. (*I.e.* the keys are required to allow transitions.) If there are issues with any of the modules it is more productive to trouble-shoot from a terminal.

Once all disks are powered, one may proceed to mount them using `cplane`. Generally, the modules will have been bound into a group "1234", so

```
oper@Mark6-4005:~$ da-client
Host: 127.0.0.1 port: 14242
>> group=open:1234
<<  !group=0:0:1234;
>> mstat?
<<  !mstat?0:0:1234:1:MHO%0011/24000/4/8:8:8:10068:24000:open:ready:sg:...
>>
```

which normally takes a few minutes.

4.2 Shutdown

Prior to shutting down the recorders, the disks should be unmounted from `cplane`:

```
oper@Mark6-4005:~$ da-client
Host: 127.0.0.1 port: 14242
>> group=unmount:1234
<<  !group=0:0:1234;
>>
```

which normally takes a few minutes. If there are any issues, you should become `root` and unmount any mounted disk with `/mnt` in the mount point.

1. At this point you should turn the module keys to off (vertical) one at a time (no need to wait, no particular order).
2. Press and hold the host chassis rocker switch (a second or two)
3. Press and hold the expansion chassis rocker switch (a second or two)

The 0|1 power switches in the back can also be powered off (to 0) for safety.

4.3 Module Removal

Modules should only be removed when not powered (key vertical).

When powered-off, the two data cables should be disconnected, using the blue pull-tabs.

The module lock (at the lower-left corner in the front) is designed to gently extract the module. Pull it forward until it "clicks" and the module should then be free to slide out. Hold it by the handle—it is heavy.

4.4 Module Insertion

Modules should only be inserted when power to the slot is off (key vertical).

Slide the module in on the two rails at the bottom of the slot.

Use the module lock (at the lower-left corner in the front) to gently make the mating connection to host power. (You should have it "clicked" down to before insertion, and "click" it up to make the connection.)

Connect the two data cables; the yellow one goes in the top slot. There should be a "click" when it is fully inserted.

At this point the key may be turned horizontal (power on). You should see the 8 green LEDs flicker as the host computer notices the disks.

4.5 Module Initialization

The procedure to set up modules is described in the Mark6 manuals. Generally modules will be shipped to ALMA already initialized. However, to create a fresh module you only need to know the slot number and the "label" for the module (8 characters, including %). Then in `da-client`, for example:

```
mod_init=1:8:MH0%0004:sg:new;
```

is the command to initialize the module in slot 1 to the module "label" `MH0%0004`. The "sg" refers to the so-called scatter-gather filesystem used by the Mark6. Finally "new" causes a complete erasure of the module. Leaving this out:

```
mod_init=1:8:MH0%0004:sg:;
```

is used to take a module back to the no-group initial module state. This is needed if the module has been previously "grouped"; this command (without the "new") destroys memory of those associations.

Normally, modules used at ALMA are grouped 4 to a recorder (group named by slots: "1234") to allow 16 Gbps recording. So one would repeat the appropriate commands above on the modules in slots 2, 3 and 4 (*i.e.* change the slot number and module label name).

4.6 Module Trouble-shooting

4.6.1 Missing Disks

A common cause of problems with the modules is that one or both of the data cables are not fully inserted. The linux utility `lsscsi` may be used to identify which disks are present:

```
oper@Mark6-4005:~$ lsscsi
[0:0:0:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdb
[0:0:1:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdc
[0:0:2:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdd
[0:0:3:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sde
[0:0:4:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdf
[0:0:5:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdg
[0:0:6:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdh
[0:0:7:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdi
[0:0:8:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdj
[0:0:9:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdk
[0:0:10:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdl
[0:0:11:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdm
[0:0:12:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdn
[0:0:13:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdo
[0:0:14:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdp
[0:0:15:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdq
[3:0:0:0]    cd/dvd  ATAPI    iHAS124   W      HL04  /dev/sr0
[4:0:0:0]    disk    ATA      WDC WD5003AZEX-0 80.0  /dev/sda
[14:0:0:0]   process Marvell  Console          1.01  -
[17:0:0:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdr
[17:0:1:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sds
[17:0:2:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdt
[17:0:3:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdu
[17:0:4:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdv
[17:0:5:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdw
[17:0:6:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdx
[17:0:7:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdy
[17:0:8:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdz
[17:0:9:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdaa
[17:0:10:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdab
[17:0:11:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdac
[17:0:12:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdad
[17:0:13:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdae
[17:0:14:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdaf
[17:0:15:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdag
```

There are two disks controllers; each gets two modules (disks [0:0:0:0] through [0:0:7:0] and [0:0:8:0] through [0:0:15:0], or disks [17:0:0:0] through [17:0:7:0] and [17:0:8:0] through [17:0:15:0]), for 32 mounted disks in all. The linux kernel assigns "scsi disk" names to all of them (`/dev/sdb` through `/dev/sdag`).

If a cable is not properly inserted (front or back), a contiguous group of 4 disks will be missing from the list above.

4.6.2 Module Initialization Problems

On relatively rare occasions, the kernel doesn't quite keep up with the disks when these are being inserted for the first time, or built into modules. A reboot is often a way to resolve this.

4.6.3 Other issues

Consult the Mark6 forum http://www.haystack.mit.edu/tech/vlbi/mark6/documentation.html.

4.7 Housekeeping Parameters

During normal operation (software release R2014.2 and later) the recorder housekeeping is captured routinely and made available at the monitoring site http://monitordata.osf.alma.cl, organized by date and time. The recorder data are under `CONTROL_VLBI_Recorder1/` through `CONTROL_VLBI_Recorder4/`. This data is gathered by the ALMA control system software whenever current releases are active (and the recorders are on the network).

The data are collected internally by a script (`m6sensors.sh`), which takes a number of arguments. In normal usage, the "alma" argument provides a terse one-liner to the control software. With the "full" argument, it outputs a fairly detailed list of the raw sensor data gathered from a variety of sources:

```
oper@Mark6-4005:~$ m6sensors.sh full
# date
time/ 20141118T194941.434306457

# sensors
in0:           +0.95 V  (min =  +0.00 V, max =  +1.74 V)
in1:           +1.82 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
in2:           +3.28 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
in3:           +3.28 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
in4:           +0.98 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
in5:           +1.65 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
in7:           +3.41 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
in8:           +3.25 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
fan2:         3221 RPM  (min =    0 RPM)  ALARM
SYSTIN:        +42.0°C  (high =  +0.0°C, hyst =  +0.0°C)  ALARM  sensor = thermistor
CPUTIN:        +28.0°C  (high = +80.0°C, hyst = +75.0°C)  sensor = thermistor
AUXTIN:        +40.0°C  (high = +80.0°C, hyst = +75.0°C)  sensor = thermistor
PECI Agent 0:  +33.5°C
cpu0_vid:     +1.708 V

# proc
MemTotal:       66077064 kB

# pscmd
23890 oper     87.3  7.1 dplane
23893 oper      0.0  0.0 cplane

# dfcmd
/dev/sda1 440G 25G 393G 6% /

# lscsi
[0:0:0:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdb
[0:0:1:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdc
[0:0:2:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdd
```

```
[0:0:3:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sde
[0:0:4:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdf
[0:0:5:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdg
[0:0:6:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdh
[0:0:7:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdi
[0:0:8:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdj
[0:0:9:0]    disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdk
[0:0:10:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdl
[0:0:11:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdm
[0:0:12:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdn
[0:0:13:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdo
[0:0:14:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdp
[0:0:15:0]   disk    ATA      ST3000DM001-1CH1 CC29  /dev/sdq
[4:0:0:0]    disk    ATA      WDC WD5003AZEX-0 80.0  /dev/sda
[17:0:0:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdr
[17:0:1:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sds
[17:0:2:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdt
[17:0:3:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdu
[17:0:4:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdv
[17:0:5:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdw
[17:0:6:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdx
[17:0:7:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdy
[17:0:8:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdz
[17:0:9:0]   disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdaa
[17:0:10:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdab
[17:0:11:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdac
[17:0:12:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdad
[17:0:13:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdae
[17:0:14:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdaf
[17:0:15:0]  disk    ATA      WDC WD4001FAEX-0 1L01  /dev/sdag
```

```
# ntp
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*10.197.50.101   LOCAL(0)         2 u  148 1024  377    0.502   -3.571   0.967
+200.89.75.197   200.54.149.24    2 u  673 1024  377   30.734    2.195   0.944
+200.1.19.4      200.27.106.115   2 u 1012 1024  377   35.891    1.241   2.874
+200.1.22.6      69.36.224.15     2 u  512 1024  377   41.778    6.211   2.539
+200.1.19.16     200.27.106.116   2 u  879 1024  377   37.398    2.042   3.993
```

In the sections above, "sensors" is provided by `lm_sensors` and provides low-level voltages, temperatures, and the fan speed. These are mostly provided to allow long-term monitoring of the system health, since if any of them are seriously out of range, the computer will shut itself down.

There are a few diagnostics on total memory ("proc") running processes ("pscmd"; specifically that both `cplane` and `dplane` are running) and disk usage ("dfcmd", since if the root disk is full, nothing will run properly). The "lscsi" output we've seen above, and "ntp" provides the offset to other recorders and/or hosts.

4.8 CCL Interface to Housekeeping

This housekeeping information discussed in the previous section is available in summary form via the `CCL` recorder `STATUS()` method.

Other methods are available but are not documented here.

Chapter 5

Monitoring and Maintenance

During the development period, Haystack is responsible for any maintenance. However, as these are new machines, little is required.

It is important to note that some of the test logs and correlation activities produce significant amounts of data. If the ~ `oper`/directory fills up with too much data, that will also fill up the root filesystem which will prevent the computer from operating normally.

So, delete temporary products, and offload test results when possible.

Chapter 6

Mark6 Scripts and Programs

One should recognize that while the Mark6 is presented to ALMA as a completed hardware component, to the rest of the VLBI community the Mark6 is still being developed to support a number of different capabilities in the future. Thus the testing for ALMA has been carried out primarily with the 1.1 software release which contains all the required functionality. Patches to that release, developed during testing will be provided as the 1.2 software release "delivery". However, it is expected that ALMA may wish to upgrade to later releases in the future for more robust operation.

As a consequence of this, the Mark6 software is installed as a collection of parts, and some of them (especially the testing ones) are evolving. In this section we review the testing tools available for the testing carried out for ALMA.

All of the programs mentioned here have some help available at the command line, either via `-h` or `--help`.

6.1 Mark6 Components

The Mark6 components are installed from unpacked distributions placed in `/usr/local/src` and installed (primarily) into `/usr/local/bin` or `/usr/bin`. These include:

6.1.1 cplane

This is the high-level Mark6 server. It manages all Mark6 operations and controls the lower-level "data plane" utility `dplane`.

6.1.2 dplane

This is the low-level "data plane" utility. It is managed by `cplane`. Because it reads the ethernet devices in "promiscuous" mode, it must run as root.

6.1.3 dboss

This is a utility for directly commanding `dplane`. Only the "t" and "s" commands are still occasionally useful. (The command is a relic from initial development.)

6.1.4 dpstat

This utility captures status message that `dplane` emits to monitor the low-level state. It is invoked with the hostname of to be monitored. (At ALMA the messages don't leave the machine, so

```
$ dpstat ‘hostname‘
```

is the standard invocation.

6.1.5 gather

This low-level primitive will "gather" the scans from one recording into a single output file.

6.1.6 gator

This utility script invokes `gather` to collect the scattered files into a single file for examination.

6.1.7 dqa

This utility program reads an entire scan and analyzes it in detail. It is thorough, but slow.

6.1.8 da-client

This utility opens a socket to `cplane` and manages the command-response converstation. It also makes sure that commands end with a semicolon. Consult [RD2] for the exact command syntax and meaning.

6.1.9 M6_CC

This is the utility script invoked by the `cplane` to run a recording schedule. It is provided separately to allow per-site modifications to the general paradigm.

6.1.10 m6sensors.sh

This utility script queries a variety of sources to gather the housekeeping reported by the recorders in the high-level (CONTROL) software.

6.2 Development Scripts

The developmental scripts are distributed via an `svn` repository checked-out into ∼ `oper/bmr` and refreshed as needed. It has been stable for some time as far as the Mark6 is concerned, but it also supports some of the DiFX development particular to VLBI at ALMA. Eventually these tools will migrate into the Mark6 distribution. (They are NOT required for normal operations of the recorders.) These tools install in ∼ `oper`/[bin]. They include:

6.2.1 m6tester.sh

This is a general "wrapper" script that allows execution of one of several standard scripts that generate a test VLBI session, record (bogus) data for it and analyze the result.

As a side effect, it exercises the recorder in an operational manner, verifying its performance for the sample schedule.

loopback.sh

This version allows a Mark6 to test itself (one 8 Gbps stream).

dberecord.sh

This version is usable when a digital back end (DBE) is available.

peerpush.sh

This is a version where a pair of Mark6s are configured to test each other. This version runs `push_test` on one to simulate the DBE and records on the other.

peertest.sh

A variant form of `peerpush.sh` with equivalent functionality. This version exercises the recorder's ability to run an XML schedule file.

dualtest.sh

This form uses `push_test` to simulate 16 Gbps recording (two 8 Gbps streams).

6.2.2 miscellaneous supporting utilities

create-schedule.py

A Python script which generates a testing schedule consisting of some number of VLBI scans. The duration and separation of the scans can be random and specified within certain ranges to simulate a wide variety of VLBI recording possibilities.

vdif_time

A small program that converts between several different (non-standard) time representations found in VLBI work. VDIF times are specified as seconds within one of a set of 64 epochs (two per calendar year).

push_test

A small program that generates a legal VDIF packet stream for the recorders to record. Its distinguishing characteristic is that the data in the packets is statistically valid (but not particularly random) and marked to allow a companion program `mark_check` to validate it.

vv

A small widget built on `libpcap` to allow raw packet capture and some "on-the-fly" diagnostics on the data stream. It started as a clean way to hex-dump packets caught with `wireshark` or `tcpdump`, but grew (organically) a few more uses. The most useful capability is one to compare the time in the data packet stream with the record clock. This is most useful to establish that the packet timing is correct to a resolution of several milliseconds.

Since this program has full access to the raw packet stream, and that is considered "privileged" information, this program needs to be run as root.

byte-rate.py

This is a widget to monitor the incoming packet rate to the network devices and log the value with the system clock time. This allows for an application-independent measurement of packets arriving to the recorders.

timechk.sh

This is a simple widget to use `vv` to sample incoming packet times and to compare them with the system clock, and also to notice the current number of processor interrupts. Analyzing the logged data allows one to work out systematic offsets between PICs. This is not needed for use in normal operations, when the packet times are set correctly, automatically. Rather it is for Engineering Port situations where the PIC times are set by hand, and therefore only approximately correct.

g4sc.sh

This is a simple driver script that can be editted as needed to perform more thorough scan checks on scans recorded from the control system.

ssh-cron.pl

A perl script that allows one to prepare an SSH "agent" capable of handling operator authentication for a remote, long duration session. This is necessary to allow access to the OSF recorders only during engineering time to set up a long duration, unattended module test.

6.2.3 module qualification

When it is desirable to test the disks in the modules more directly (and keep `cplane`/`dplane` out of it), one has many options for disk testing. Mark5 modules require "conditioning" prior to use. Fundamentally, this is due to the way disk controllers cope with bad surfaces (and have done so for several decades). The internal disk real estate is subdivided into sectors (*e.g.* 512 bytes) and if the controller is unable to read data from a given sector, it is declared "bad" and "spared". That is, the bad sector is transparently replaced by one from a special set of good sectors. This process takes time, however (and bad sectors truly are bad), so it is desirable to write and read an entire module prior to use.

hammer.sh

Is a fairly simple utility that writes files of a fixed size until a module is full, and then reads them back and checksums them. This is equivalent to the Mark5 "conditioning" process.

hammerplot.sh

This is a script to graphically plot the log data saved after `hammer.sh` is done.

6.2.4 evalpush.sh

This is a VLBI scan evaluator that can invoke any of several checking programs `mark_check`, `scan_check` or `dqa` to verify a schedule of recordings.

mark_check

This is a program written specifically to verify data that presumably was generated by `push_test` and report on issues. It has modes to randomly check some fraction of data, or to sequentially check every packet as desired.

scan_check

This is a standard VLBI capability to verify that, following a VLBI recording, that the data is valid. In the Mark6 implementation, it is available directly within `cplane` to produce a one-liner response to the `scan_check` query, or in this case as a stand-alone command to provide progressively more verbose reports on the scan (fragment) contents. When invoked by `cplane`, the `scan_check` runs in a relatively fast mode where it merely checks the basic integrity of the scan and verifies a small sample of the data. When invoked offline at the UNIX command prompt, one can use options to increase the level of checking.

6.2.5 vdifuse

This program provides a FUSE (Filesystem in User SpacE) interface to the scatter-gather files recorded by the Mark6. Basically it presents an interface to the kernel so that scans (which in fact are scattered on up to 32 separate filesystems) appear to ordinary UNIX programs as a single file. It is still under development, but it is essential to allow current Mark6 recordings to be accessible to DiFX.

6.3 "Doit" and Friends

Given the above plethora of scripts and testing options and hardware configurations across the various Mark6 recorders setup in various ways at various times, it was found convenient to write a simple "just do it" script to launch the test of the hour. This script typically lives in ~ `oper/test` and is named with the name of the recorder, *e.g.* `./doit-4005.sh`. As cables are reconfigured, it is simple to adjust the script with, *e.g.* new ethernet device names, or the address of peer recorders.

Additional "one-off" scripts are also written to automate the mechanics of testing data files in various ways. (These usually start as a single UNIX command line, and then migrate into a small file with a slightly more general version.)

6.4 DiFX

As indicated, DiFX is installed on the recorders to allow data recorded to be analyzed from the perspective of the ultimate consumer (*i.e.* the VLBI correlator). Normally, after a VLBI session, the recorded modules are shipped to the VLBI correlator (*e.g.* Haystack or Bonn). Having DiFX installed means that certain tests can be carried out *in situ* by a qualified DiFX operator. These tests are truly only needed to support the path to Commissioning the VLBI capability. It is expected that a qualified DiFX expert is available to perform the correlation and to analyze the results.

Details on DiFX including some documentation, may be found at the wiki for the software project, http://cira.ivec.org/dokuwiki/doku.php/difx/start.

For the purposes of ALMA testing, a top-level script, `corel_alma.sh` is written to allow correlation of specific scans which must be configured through a supplementary script/file `corel_params.sh` (which is itself usually a symbolic link to one of several files in the ~ `oper/bmr` area).

6.5 Science Scripts

For completeness, we mention several scripts here that are used in the development and exercise of the complete VLBI system.

When the control system is active, the VLBI Observing Mode (VOM) is run out of (python) scripts stored in `/groups/science/scripts/APP`. Since interfaces or capabilities have changed through the development process, there are frozen copies tied to various releases. However, as of `R2014.4`, there are three scripts that are relevant to hardware testing, described in the following subsections. The Software Operations Regression test suite for VLBI uses these for its weekly regression test of the APP functionality, https://ictwiki.alma.cl/twiki/bin/view/SoftOps/WeeklyRegTestVLBI.

6.5.1 VLBIRecorderSetup.py

This script assumes all four recorders are present with working modules and that the OFLS fibers are connected nominally. It then issues the commands to `cplane` to prepare the recorders for normal operations and verifies that `cplane` has implemented these commands.

6.5.2 VLBITestObs.py

This script launches a VLBI test observation. It has command line flags to set a (prodigious) number of options. The essential ones select the receiver band and antennas to designate for special purposes. Use the `--help` argument for more details.

6.5.3 VLBITestAdj.py

This script is intended to make (test) adjustments to the VOM while it is operating. The principal use of this is to remove or add antennas to the set used for phasing.

6.5.4 ripVLBIlog.py

ALMA software components log their data to a variety of sources while running, and also to a common Java logging facility which creates a (rather large) XML file after some minutes of operation. These logs are periodically gathered and made available to a web site (ordered by time) `http://computing-logs.aiv.alma.cl`.

This script is designed to extract the specific logs from `VLBITestObs.py` or similar sources and provide them as an ASCII file, rather than requiring the user to load the XML into some other tool.

Chapter 7

Specific Test Descriptions

7.1 General

Prior to any test session, you should:

- review the general health of the recorder
- restart `cplane/dplane`
- verify that sufficient disk space exists in ~ `oper/`and on the modules

Following every test session, you should:

- leave the recorder in a usable state
- clean up temporary files
- offload data products
- write the report

7.2 Module Test

7.2.1 using doit

As this is primarily a test of the modules, rather than the recorder, it is sufficient (and easier) to test using single input streams (8 Gbps) to a pair of modules. Two configurations are easy to use:

peerpush.sh In this configuration a pair of recorders are cabled together. Each recorder can generate packets using `peer_push` on one interface, and receive packets to record on the other in a pair of modules. The route tables need to be adjusted so that the packets are actually sent to the approprate interface (using `ip neigh` as explained by the scripts). This is configured in the `doit-*.sh` script.

loopback.sh In this configuration, one recorder sends packets to itself via a cable from one network interface to another. The route tables need to be adjusted to prevent the kernel from routing packets via the loopback interface as described in the script. This is configured in the `doit-*.sh` script.

Depending on the level of checking required, an environment variable `chks` can be set to a list of test to run. In most cases it is sufficient (and faster) to check with `chk` which signals that only `scan_check` should be used; but `mark` will turning on checking with `mark_check` and `dqa` with (you guessed it) (dqa).

The test sequence is as follows:

```
# environment preparation
export chks=chk,mark
# run the test on a pair of 48TB modules for 12 hours, Axx is a label
./doit-4005.sh 48TB Axx 12
# wait 12 hours & verify that the script ran to completion
tail m6t-m.*.log
# verify the log files are present
ls -ltr out/* | tail
# look for problems:
it=Axx
grep FAIL out/$it.scans
grep check out/$it.chk | grep -v 0.fail
# remove the scan data
rm /mnt/disks/data/?/?/*.vdif
# make sure nothing is hanging; if so, kill it.
ps x
```

If using `peer_push.sh`, you can test from two recorders simultaneously.

7.2.2 using hammer

For situations where (re)cabling is not desirable, the `hammer.sh` script is a simple way to test a module. With modules inserted, *e.g.* :

```
hammer.sh size=40g slots=1234 purge=true
```

will launch the test. After enough time has gone by, you can plot the results with

```
hammerplot.sh logfiles ...
```

which uses `gnuplot` to generate a summary PNG plot.

7.3 Link Test and Packet Rate

Whenever a source of packets is present at the recorder, there are several methods that may be used to verify that the ethernet link is present and that packets are being received.

7.3.1 ifconfig

The recorder network devices are brought up or taken down by the `ifup` and `ifdown` commands which take as an argument the name of the ethernet device. (Typically these are `eth2` through `eth5` for the four ports on the Miricom network card(s).) At boot time, the device statistics are cleared and the interface status message is as follows:

```
$ ifconfig eth3
eth3      Link encap:Ethernet  HWaddr 00:60:dd:44:b3:eb
          inet addr:172.16.3.4  Bcast:172.16.3.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:9000  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:83
```

thereafter the various counters increment according to packets received. For normal ALMA PIC operation, the packet rate is 125000 packets per second received. Occasionally the system may send some network query packets (ICMP which affect TX; or, a human may cause it, *e.g.* if the interface is “pinged”).

For convenience, one can use `grep` to see only the lines with "RX", "TX" on them, as those are the most important. If more precise timing is desired, the script `byte-rate.py`, mentioned above in Section 6.2.2 can be used to put system-level timestamps on the packet counters.

7.3.2 interrupts

As shown in the preceding section, **CPU!** interrupts are assigned to each of the interfaces. Statistics associated with these are available from the UNIX kernel via `/proc/interrupts`. A readable version relevant to the 10 GbE interfaces may be obtained as follows:

```
$ cat /proc/interrupts | grep eth | tr -s ' '
 84: 2 0 0 2901879475 0 0 0 0 0 0 0 0 PCI-MSI-edge eth3
 86: 2 0 0 0 0 2931489631 0 0 0 0 0 0 PCI-MSI-edge eth5
$ cat /proc/interrupts | grep eth | tr -s ' '
 84: 2 0 0 2903340143 0 0 0 0 0 0 0 0 PCI-MSI-edge eth3
 86: 2 0 0 0 0 2933012445 0 0 0 0 0 0 PCI-MSI-edge eth5
```

Normally the iterrupts are assigned at boot to be placed on different **CPU!**s (one **CPU!** per column above). The numbers increment with the number of interrupts which is proportional to the number of packets received.

7.3.3 filled packets

When run at verbosity level 3 (`dplane 3`), the application provides additional information in its output. The `m6tester.sh` application starts `dplane` in such a way that the log file name is `m6t-d-timetag.log`.

In particular, one of the things logged is the insertion of filled packets into the data files written. A simple `grep` for `filled` is sufficient to recover a per-recorder log of filled packets by data stream.

7.4 Multi-Quadrant ASUMMER Test

For some of the tests we conduct with the ASUMMER logic we would like to record the sums with the PICs. In these tests, the source of the signal might be some programming of upstream devices (*e.g.* TFBs ro **DRX!**s). The PICs themselves would commanded via the Engineering Port (EP).

7.4.1 Multi-Stream Time Offsets

In normal operations, the Control system takes care of providing all Correlator cards with exact times, and the VOM takes care to provide the PICs with VDIF headers that are synchronized to ALMA time. When lower-level tests are conducted, however, it is impossible to manually command all the PICs with the proper time when they are called upon to transmit packets via the EP.

When such tests are constructed, the `timechk.sh` script (Section 6.2.2) may be used to capture the packet times and time-stamp them with the recoreder clock. Note that the recorders are NTP-sycn'd to an error of at most a ms, and that the packet queues are not that deep (perhaps a few ms). Thus a packet so stamped with have some gross offset (due to the crudity of the commanding—perhaps as much as a minute) and some residual error that is only a few milliseconds.

By fitting the output of the script to a line for each PIC, it is possible to generate an average timing solution. This average tends to converge to packets in the middle of the receive queue (which is the same for all data streams). Thus the *relative* difference between these average timing solutions is adequate to align the data for the VLBI correlator (*i.e.* , make a good guess for the "clock early" parameter of the VEX input file presented to DiFX).

7.4.2 Analysis

A detailed explanation of the procedure for the analysis of such data sets is well beyond the scope of this document. However, the ideas are relatively simple, and we can spend a few words here to explain. In the general VLBI setup, the DiFX correlator is provided with data files containing the recordings together with a wealth of meta-data describing the experiment (usually contained within the VEX input file).

For the tests we have at ALMA most of this information is not necessary, as what we mostly want to do is to correlate the data files in various combinations. A relatively easy way to do this is to construct a temporary VEX, setup the correlation, and then examine the results using standard VLBI analysis tools. The principal one here is called `fourfit` which, among other things which are not relevant here, looks for the correlation peak and reports on the strength of the correlation.

7.5 Multi-Quadrant Science Correlations

When the Control system is available (R2014.4 or later), one may run a simulated VLBI observation using `VLBITestObs.py` (Section 6.5.2) according to the general instructions provided for the regression test (Section 6.5). Depending on how that observation was set up, it may be possible to make arrangements such that the signals presented to the several PICs should correlate in various ways. When this is done, we may directly correlate the results as described in Section 7.4.2 without needing to worry about the issues covered in Section 7.4.1.

That is to say, it is far easier to just run the simulated VLBI observation than it is to make all the same arrangements using the only the low-level EP commands.

Aside from choosing a source (a bright quasar available at the target time of day), there are four principal ways to run this test:

A all quadrants with the same frequency setup, but with a the same, single antenna. This directly vets the VLBI backend, as the recorded signals should be highly correlated.

B all quadrants with the same frequency setup and same set of antennas to be phased. This is useful to directly compare the processing in each quadrant. The results should be very similar, but not identical.

C all quadrants with the same frequency setup and but with a different set of antennas in the phased sum for each quadrant. This allows some checks of the phased-sum performance.

D as a normal observation.