Atacama Large Millimeter / submillimeter Array

# ALMA Phasing Project Update to Corr/Control Design

ALMA-05.11.61.01-001-A-DSN

2013-05-16

| Prepared by: | Organization Role | Date and Signature |
|---|---|---|
| G. Crew<br>M. Mora-Klein | MIT<br>NRAO | |
| 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 | 2013-04-24<br>2013-05-16 | All<br>6.3 | G. Crew | CDR Draft<br>editorial |
| | | | | |

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Purpose

This document presents the design of the VLBI Observing Mode (VOM) introduced for the ALMA Phasing Project. The purpose of the APP is to enable ALMA to participate in VLBI observations. To do this, the signals from the antennas in the array must be coherently added and recorded for eventual VLBI correlation. A full overview of the project is presented in the project plan [RD1].

The software described in this document covers the design of the additions to the Observatory software necessary to run the observation.

## 1.2 Scope

There is a companion design document [RD2] addressing the modifications to TelCal to support the phasing project, so those aspects of the design will not be covered here. In addition, some specific details influencing the design are better addressed in design memos, so their contents will not be repeated here.

In addition to the two main design documents, the software management plan [RD3] includes information about the schedule of development.

## 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 Implementation Plan | ALMA-05.11.10.01-0001-A-PLA |
| [RD2] | APP Update to TelCal Design | ALMA-05.11.62.01-001-A-DSN |
| [RD3] | APP Computing Management Plan | ALMA-05.11.60.01-001-A-PLA.pdf |
| [RD4] | APP Requirements | not assigned |
| [RD5] | DiFX enhancements | not assigned |
| [RD6] | ICD between APP and ALMA Computing | ALMA-05.11.00.49-70.00.00.00-A-ICD |
| [RD7] | Maser user guide | not assigned |
| [RD8] | Optical Fiber Link System user guide | not assigned |
| [RD9] | Mark6 user guide | not assigned |
| [RD10] | SDMTables_postSDM2FBT.pdf | not assigned |

## 1.4 Acronyms

**ACS** ALMA Common Software

**AIPS** Astronomical Image Processing System

**ALMA** Atacama Large Millimeter/submillimeter Array

**APP** ALMA Phasing Project

**AOS** Array Operations Site

**ASCII** American Standard Code for Information Interchange

**ASDM** ALMA Science Data Model

**BDF** Binary Data Format

**CAI** Correlator Antenna Input

**CAN** Controller Area Network

**CASA** Common Astronomy Software Applications

**CC** Correlator Cards

**CCC** Correlator Control Computer

**CCL** Control Command Language

**CDP** Correlator Data Processor

**CVR** Central Variable Reference

**DAS** Data Acquisition System

**DiFX** Distributed FX (Software Correlator)

**EDV** Extended (Header) Data Version

**FITS** Flexible Image Transport System

**FITS-IDI** FITS Interferometry Data Interchange Format

**FTE** Full Time Equivalent

**GPS** Global Positioning System

**GUI** Graphical User Interface

**HTML** HyperText Markup Language

**IDL** Interface Definition Language

**IF** Intermediate Frequency

**LO** Local Oscillator

**LTA** Long Term Accumulator

**MIT** Massachusetts Institute of Technology

**MJD** Modified Julian Day

**NRAO** National Radio Astronomy Observatory

**OSF** Operations Support Facility

**OFLS** Optical Fiber Link System

**OPT** Observing Preparation Tool

**PI** Principal Investigator

**PIC** Phasing Interface Card

**PPS** Pulse Per Second

**PSN** Packet Serial Number

**RDC** Residual Delay Correction

**SB** Scheduling Block

**SCPI** Standard Commands for Programmable Instruments

**SFIOM** Single Field Interferometry Observing Mode

**SVN** Subversion

**TE** Timing Event

**TFB** Tunable Filter Bank

**UTC** Coordinated Universal Time

**VEX** VLBI EXperiment file

**VDIF** VLBI Data Interchange Format

**VLA** Very Large Array

**VLBI** Very Long Baseline Interferometry

**VOM** VLBI Observing Mode

**VSI-S** VLBI Standard Interface for Software

**VTP** VDIF Transport Protocol

**WVR** Water Vapor Radiometer

**XML** eXtended Markup Language

# Chapter 2

# Prerequisites

Before proceeding to the adopted design, we briefly review the driving requirements on this software. We also touch on some of the assumptions about the installed hardware and operating conditions which further guide what the software is, and more importantly isn't required to do.

For the reader's convenience, we reproduce the block diagram for the entire project from the project plan [RD3] which shows the enhancements from a hardware-centric viewpoint in Figure 2.1. The corresponding software view will appears in Section 3.1.

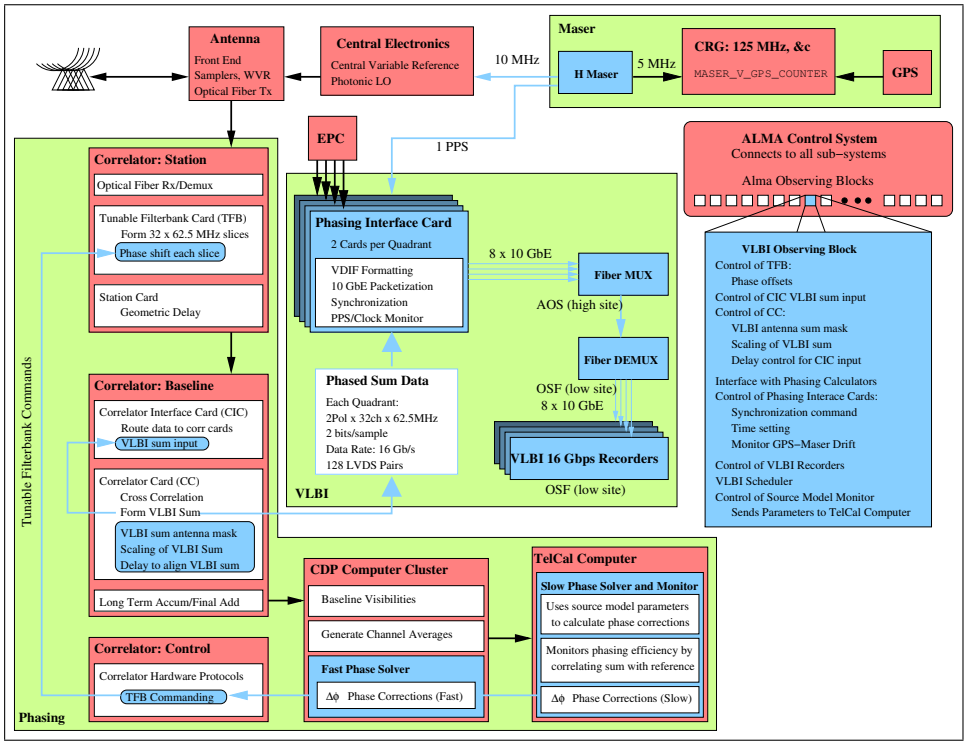


***Figure 2.1:*** *ALMA Phasing Project System Diagram. This shows how the existing ALMA system is being modified to support VLBI. Objects in pink currently exist, objects in blue are new, and the green backgrounds group related components of the system.*

## 2.1 Requirements

The requirements shown in Tables 2.1 and 2.2 are drawn from the requirements document [RD4], and are shown again here as they drive the design in certain ways. Aside from these, it

should be pointed out that VLBI observations are planned in advance and the scans are scheduled in UTC: all stations need to observe at the same time or VLBI correlations are compromised. Thus it is a fundamental consideration that VOM "soldier on" in the event of most anomalous situations, rather than aborting the observation and rescheduling after the issue is addressed. Note that the requirements refer to VLBI scans (see disambiguation in Section 3.3).

## 2.2 Assumptions

We begin by pointing out that a VLBI observation can be achieved with minimal modifications to ALMA's Single Field Interferometry Observing Mode (SFIOM hereafter). The SFIOM already provides mechanisms to control an array of antennas and run a sequence of scans on specified targets with all the necessary calibrations—in short nearly everything we need.

In addition to the existing capabilities of the SFIOM, the VOM needs the following:

- A physical infrastructure, see Section 2.2.1
- Inputs for the VLBI observation, see Section 2.2.2
- Post observation arrangements, see Section 2.2.3

and we shall discuss these in this order in the following sections.

### 2.2.1 Physical Infrastructure

We briefly review the physical enhancements to the ALMA system that will be in place when the VOM is used, and a few comments about how these are used. We also consider mention of the things we do not expect the VOM to do.

Since VLBI operations are expected to be relatively rare, it is anticipated that components of the system may be mothballed between VLBI sessions to increase longevity and/or to avoid wasting AOS power. For example,

- there will be a procedure prior to VLBI sessions to verify that the observatory is ready
- switch-over between hydrogen maser and rubidium clock: It is not likely this would be done often; the switch-over would be accomplished via a procedure and a change in the device configuration database (*i.e.* to use one and not the other).
- powering on/off the PIC cards: There would be a control command in the correlator software system for this, but it probably would not be done during the session itself.
- powering on/off the optical fiber link system: This would be done by procedure (*i.e.* flip the power switches).
- powering on/off the recorders: This would be done by procedure (*i.e.* power/reboot the computers).

The software devices for the optical fiber link system and the recorders should be able to operate and return, *e.g.* a power-off status if the physical devices are not available.

The (hydrogen) maser replaces the rubidium (maser) clock to stabilize the LO system and the sampling to the stability required for VLBI. We return to some detail about the maser and its software device in Section 7.1.

The eight phasing interface cards (PICs), which produce the summed data used for VLBI, are installed in the correlator but are not needed in normal operations but are operational throughout VLBI observing. We discuss the PIC and its hardware device in Sections 5.3.1 and 6.2.

Aside from the phased-sum data, the PICs may receive 1 PPS signals from the Maser and a test GPS unit. If these are available, the offset of the ALMA TE to GPS and Maser "Tick" may be directly measured.

The firmware to properly support the phased sum and the revised LTA protocols is installed in the appropriate correlator cards. One of the protocols throws a switch that replaces CAI-63 with the phase-sum data.

***Table 2.1:*** *Driving VOM Requirements*

| Req. # | Baseline Requirement | Additional Goals | Comments |
|---|---|---|---|
| **APP0110** | **Maser status:**The Maser status / health information shall be accessable via a network interface and recorded at least once every 10 minutes. | | |
| **APP0140** | **Phasing efficiency (stability):** The APP phasing efficiency shall be as stable as the atmospheric coherence timescale of the median antenna of the array. | | The APP should not make things worse than the atmospherics. IF a set of antennas are well behaved (and some are not) the phasing system should phase up the stable ones and discard the noisy ones. Phasing efficiency is real/theoretical. |
| **APP0310** | **Recorder control:** Commands to the recorders shall adhere to the VLBI Standard Software Interface Specification (VSI-S). | | |
| **APP0350** | **Media capacity:** The disk modules will hold a minimum of 9 hours of data. | | Nominally, four 16 TB modules/recorder provides 9 hours of continuous recording or 18 hours at 50% [observational] duty cycle. |
| **APP0360** | **Experiment session:** Each VLBI session shall be described in a manner compatible with existing VEX file systems in use at 3 mm and 1.3 mm observatories that are expected to participate in VLBI observations with ALMA. | Use version 2.0 when the draft is released | The VEX file principally describes the schedule, the targets and the frequency tuning. Since support for arbitrary VLBI experiments is a goal of the VEX specification, this project will consider VEX to be limited |
| **APP0370** | **Session duration:** The APP system shall support sessions lasting up to 18 hours. | | Long sessions may require lower duty cyles to satisfy APP0350. |
| **APP0390** | **Interscan gap:** The APP system shall support scans separated by a minimum of 10 seconds. | fast switching is eventually desirable | |
| **APP0400** | **Scan duration:** The APP system shall support scan durations between 10 and 900 seconds. | | |
| **APP0410** | **Experiment scans:** The APP system shall complete at least 90% of scheduled scans. | | |
| **APP0420** | **Scan scheduling:** The APP system shall support scans scheduled in UTC time and shall start/stop within 2 seconds of the scheduled time. | | |
| **APP0430** | **Band support:** Band 3, 6 receivers shall be fully supported. | All bands supported, specifically including band 7. | Band 3 support is required to support early commissioning tests even for measurements that only require Band 6. |
| **APP0440** | **Minimum bands:** APP shall support simultaneous use of one to four ALMA frequency basebands | | |

*Table 2.2: Driving VOM Requirements, Continued*

| Req. # | Baseline Requirement | Additional Goals | Comments |
|---|---|---|---|
| **APP0450** | **IF frequency:** The APP system shall support faithful programming of the IF band specified by the VEX file. | | ALMA and EHT stations must be band-compatible, and normally the LO is tuned to the PI intent in possibly a more flexible manner. |
| **APP0460** | **TFB tunings:** The TFB channel placement shall be capable of being made compatible with the $2^n$ MHz sampling schemes of traditional VLBI. | | Spacing/alignment is at MHz intervals, where $n$ is 5 or greater. The per-antenna LO offsetting shall result in frequency alignment of the sums. |
| **APP0470** | **Observing correlator:** The ALMA correlator shall operate in a single Nyquist sampled, single region frequency division mode covering the full 2 GHz bandwidth on each quadrant | | Mode #13 of ALMA Memo 556: which provides 32 subchannel filters (TFBs) for 2 GHz total bandwidth, 2048 spectral points, 976 kHz spectral resolution, 1.28 km/s velocity resolution at 230 GHz; the minimum dump time is 0.512 seconds. |
| **APP0480** | **LO Tuning:** All Antennas shall have the same LO tuning | | Spacing/alignment is at MHz intervals, where $n$ is 5 or greater. The per-antenna LO offsetting shall result in frequency alignment of the sums. |
| **APP0490** | **Antenna participation:** The phasing system shall be capable of phasing up an array consisting of an arbitrary odd number of antennas $< 64$. | Compatibility with subgrouped operation. | Subarrays are not currently supported by ALMA, but the implementation should not preclude phasing a subarray when that capability is made available generally. |
| **APP0500** | **Antenna 63:** The antenna assigned to CAI-63 shall be part of the observing array but omitted from the phased sum | | Antennas count from 0-63 |
| **APP0510** | **Log archival:** Information necessary for the post-observation (VLBI) correlation and analysis shall be archived. | | Necessary for calibration |
| **APP0520** | **Independent systems:** The phasing and recording systems shall be operated separately. | | *I.e.* the phasing system must be operable even if the sum data is not recorded; the recorders may record data even if the phasing system is not active. |
| **APP0530** | **Independent quadrants:** The APP system shall support independent operation of the four correlator quadrants. | | Each quadrant and VLBI backend is essentially an independent data stream and there is no reason to introduce coupling between them. |

The optical fiber link system (OFLS), which connects the PIC to the Mark6 recorders at the OSF, is also not needed in non-VLBI ALMA operations, but should be active for VLBI sessions. We discuss the OFLS and its software device in Sections 7.2.

The recorders will require manual intervention whenever modules need to be exchanged. (Modules which have valid data on them will be passed on to shipping for delivery to the VLBI correlator after the VLBI session is concluded.) Otherwise, the VOM controls the recorders to arrange for recordings of each scan and verification that the data is recorded properly. The recorder monitoring and its software device are discussed in Sections 5.3.2 and 7.3.

### 2.2.2 Observation Preparation

Every VLBI observation has an associated VLBI EXperiment (VEX) file which specifies the tuning of every station and the scan/target sequence. These are typically generated using a program called `SCHED`; and the current version (1.999) allows specification of some information that will be useful for ALMA VLBI observations. On the other hand, this format is intended to be sufficiently flexible to allow rather arbitrary VLBI observations to be specified—indeed, some of what may be specified will be beyond ALMA's capability to follow.

Conversely, ALMA observations in Single Field Interferometry mode are generally planned using the Observing Tool which generates a scheduling block. Many of the observations that could be planned with the Observing Tool are incompatible with VLBI.

It is assumed that the initial implementation of the VOM at ALMA will employ only a reasonable subset of the eventual full capabilities. These are discussed in more detail in Section 4.

### 2.2.3 Post-Processing

Historically, VLBI stations provide the recording media and "station logs" to a common correlation site. There the data is replayed by a special VLBI correlator to produce the visibilities which are ultimately analyzed for the science. In ALMA's case, the "station logs" are implicit in the observatory data (or metadata) that is to be sent to the VLBI PI, so generation or extraction of the "station log" *per se* can be simply accomplished through one or more Python scripts executed from CASA by the PI.

In addition to the "station logs", the maser drift is also needed for the correlation. This may be obtained as discussed in Section 7.1.

These and other details of the post-processing are best covered in a separate design/planning document that describes the VLBI post-processing [RD5].

Moving beyond the needs of VLBI, by construction the VLBI Observing Mode will operate in a manner similar to SFIOM, and therefore the ALMA data taken should be amenable to normal analysis in CASA. Depending on the options used in phasing, however, there may be additional analysis steps due to deactivation of some normal processing in the CDP nodes.

# Chapter 3

# VLBI Observing Mode

## 3.1 Design Overview

As promised in Section 2, a diagram of the observatory software for the APP may be presented that matches Figure 2.1; *i.e.,* Figure 3.1, reproduced from the project plan [RD3].

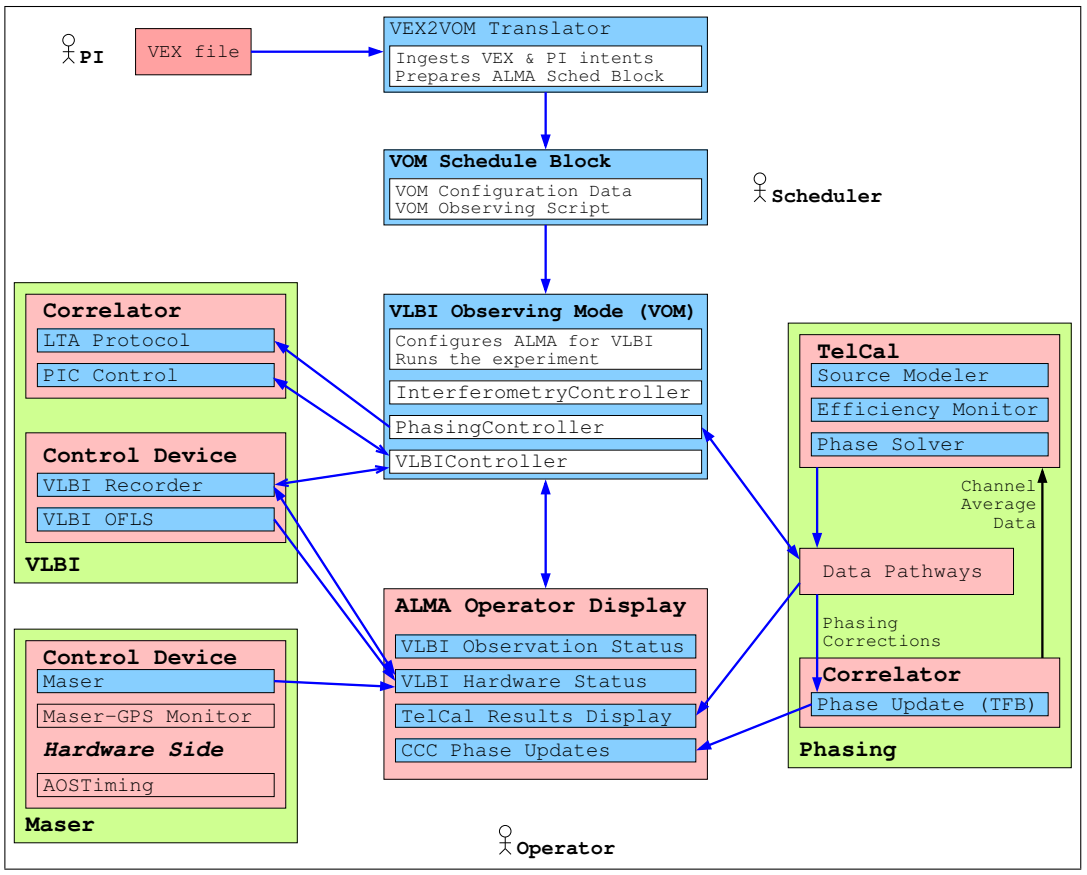


***Figure 3.1:*** *ALMA Phasing Project Software Diagram. This shows the new software components added to support VLBI. Objects in pink currently exist, objects in blue are new, and the green backgrounds group related components of the system.*

In this diagram, the pinkish color is used for existing items, blue/white for new items, and greenish backgrounds serve to gather related functionality. Note that much of the existing observatory software is suppressed. However, a few pinkish areas are included for context.

The dark blue arrows in the software architecture diagram identify the software interfaces which must be defined between the various components. These are enumerated in Appendix C.

The observation starts with the PI's planning at the top of the figure. A VLBI observing run is typically broken into several sessions of approximately 12 hours duration; a fresh instance of the VLBI Observing Mode (VOM) will be invoked for each. Each such session is described by a VLBI Experiment (VEX) file. As with other observing modes, the observing script and other information is provided in a scheduling block which in this case is prepared by the `VEX2VOM` tool.

The VOM is an augmented version of the Single Field Interferometry observing mode (SFIOM). Once the VOM is created, it manages the observation through the existing `InterferometryController` as well as two new controllers, the `PhasingController` and the `VLBIController` which manage two independent domains of activity. Note that the phasing system can operate even if the PICs are turned off; and likewise the PICs can capture data even if the sum is not phased (this would probably only be useful if a single antenna was in the array).

On the VLBI side, the work is mostly management of the dedicated hardware. On the phasing side, there is a loop between TelCal (Telescope Calibrations subsystem) which analyzes the phases of the antennas and the correlator which adjusts the relative antenna phases. The connection between the Correlator and TelCal flows through several existing interfaces which (for now) are hidden here behind an opaque "Data Pathways" box. TelCal does its work from "channel average" data which is a normal Correlator data product. Finally, the operator is apprised of the system behavior with new graphical user interface (GUI) panels.

A discussion of the details of the design is perhaps best begun with a relatively high-level walk through the VOM's activities in one session as we do in the following Section.

## 3.2 One VOM Lifetime

A simplified view of of the VOM lifecycle is shown in Figure 3.2. For simplicity, most of the normal activities of the SFIOM which the VOM implicitly performs are not shown; a few landmarks are indicated with a light blue. The other colors are tied to the different controllers which execute the functionality.

After the VOM is created the first order of business it to initialize the new hardware. First, the LTA needs to be programmed to pass the phased-sum along to the correlator on one of the antenna inputs (CAI-63) rather than the antenna that is physically connected there. Next, the LTA needs to scale this phased-sum product according to the number of antennas in the array.

The phased-sum gets to the recorders through the PIC cards; these need to be programmed with the current time so that the recorded data is properly time-tagged. Since it is critical that this be done correctly for VLBI data, the VOM will verify that the PIC cards have properly interpreted the command and check that the offsets of the observatory time from the test GPS and Maser ticks are also sensible.

After that, the recorders need to be prepared to receive the data. Presumably the operators will have installed empty disk media into the recorders; so this is mostly a check that the recorders see the required number of free bytes to record the full session, and that their clocks are accurate. The recorders are given a copy of the VLBI scan schedule so that they can autonomously make the recordings at the appointed time.

From this point onwards, the VOM is driven by the scan schedule as specified in the VEX file. Thus it needs to wait until shortly before a scan is to start. At this time it points the antennas of the array, programs the Central LO and Front Ends, and programs the correlator for a suitable scan sequence (*e.g.* Mode 13). The details of the scan sequence are discussed in the next section (Section 3.3. However at this point it is important to note that the logic operating in TelCal operates on a "slow" timescale (many seconds) and the logic operating in the CDPs is on a "fast" timescale (on the order of a second).

Depending on the phasing strategy (current weather and scan intents), the application of water vapor radiometer (WVR) delays (as calculated by the CDPs) will need to be enabled or disabled. The residual delay calculation may also need to be enabled/disabled, and the delay center may need to be moved. Thus both the CDPs and TelCal need to be instructed on the phasing plan. More details on all of this are to be found in the discussion of the `PhasingController` in Section 5.4.

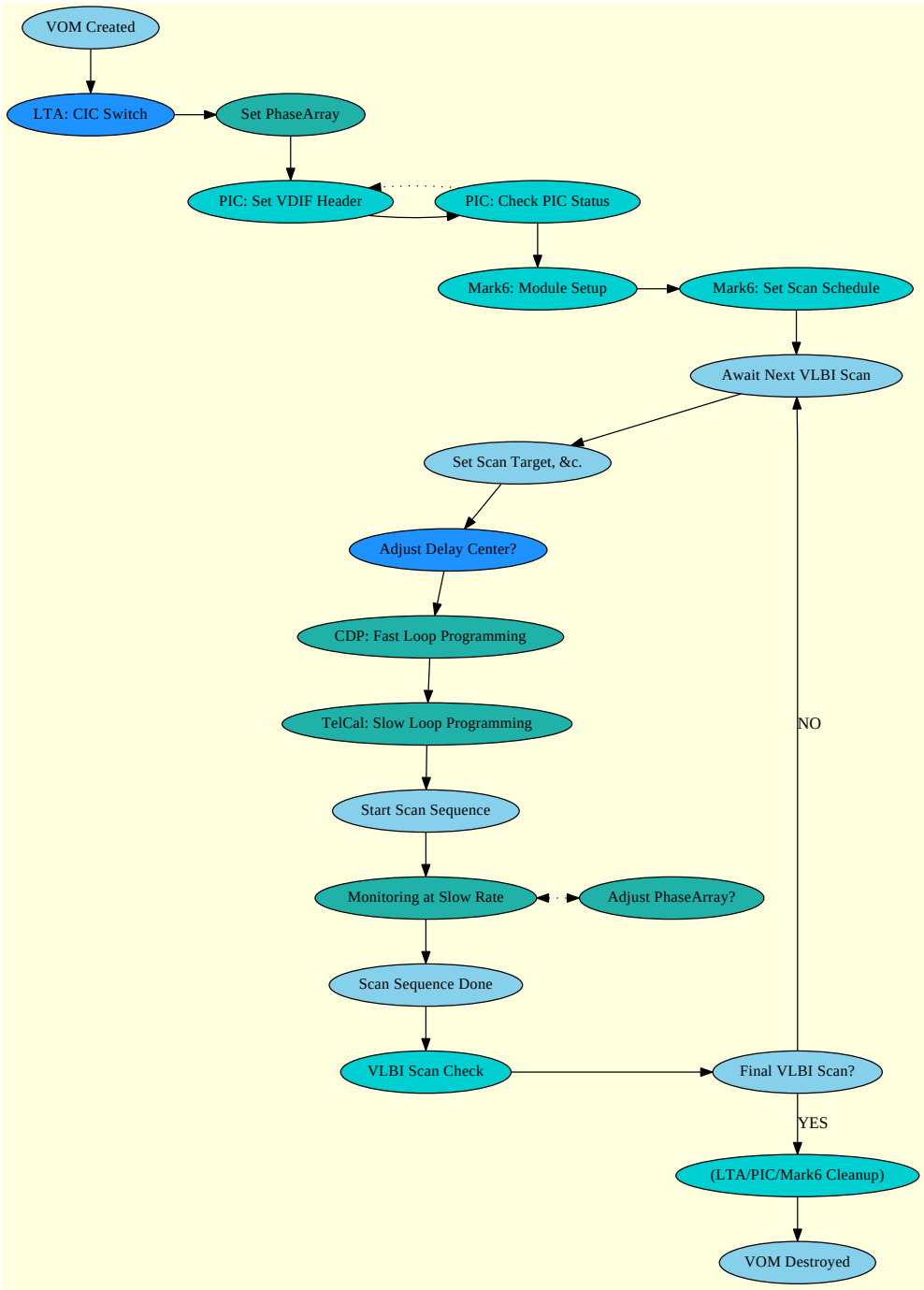


***Figure 3.2:*** *Simplified VOM state diagram. The colors reflect the controllers responsible for the activities, and the ovals label some important steps. Activities in common with the SFIOM are either in light blue or suppressed.*

Then the phasing loop is started and runs for the duration of the VLBI scan. The VOM participates in the processing on the "slow" timescale, monitoring status, and perhaps adjusting membership in the array according to the efficiency of the phasing result and whatever else TelCal

determines.

Finally, at the end of the VLBI scan, the recorders perform a scan check which verifies that an appropriate amount of valid data is present. Once this is complete, the VOM waits for the next VLBI scan, or else it is done.

There is likely no cleanup that needs to be performed before the VOM is destroyed, but potentially, the LTA should be reset to the normal ALMA operational mode as a precaution.

## 3.3 Scan Timescales

The phasing system operates on two timescales; a (more complex) "slow" scale operating in TelCal and a (simpler) "fast" one managed in the correlator. The slow timescale is used for management of the phasing system by TelCal and the faster one is available for simple calculations in the CDP. A suitable subdivision of VLBI scans into ALMA scans will be made to optimize the phasing loop performance with respect to all considerations. Figure 3.3 is presented to understand this overview.

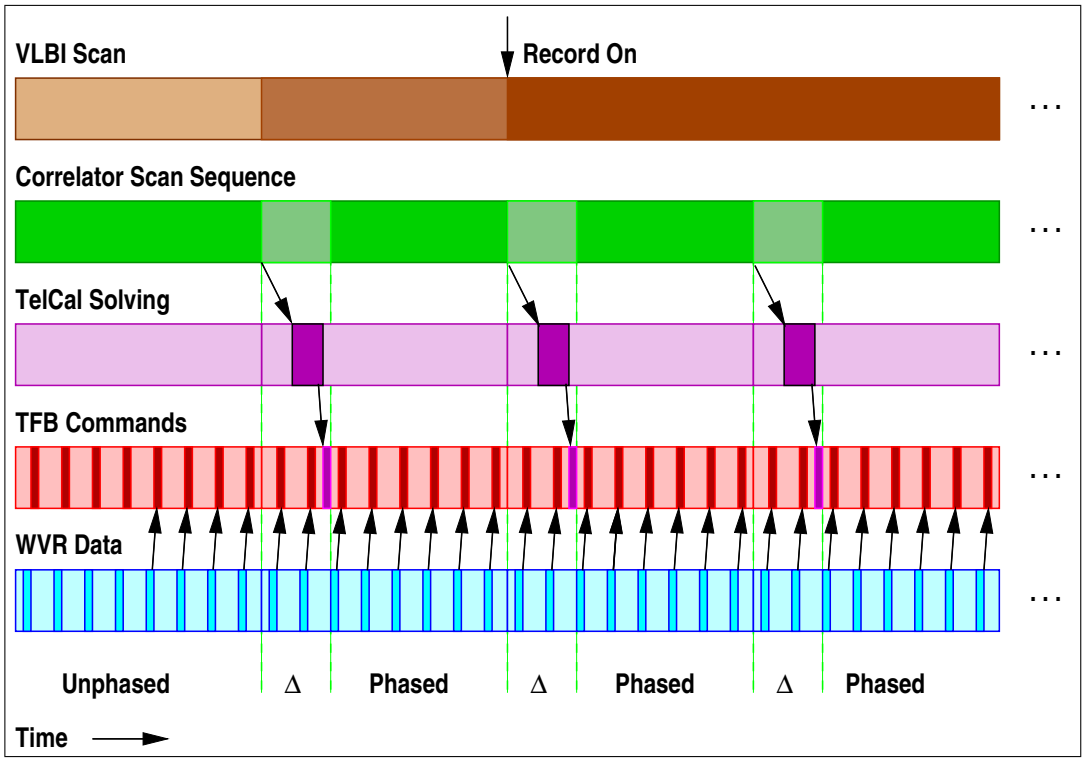


***Figure 3.3:*** *Scans in the Phasing System. Each VLBI scan is partitioned into "subscans" for correlation and "slow" timescale processing in TelCal. WVR adjustments are made in the CDP on a "fast" timescale. See text for discussion.*

First it is important to distinguish the following kinds of "scans" and "subscans" within the system to avoid confusions:

**Subscan** : The correlator works with subscans and sequences of subscans only, and may be reconfigured between subscans.

**Subscan sequence** : A sequence of subscans, with optimized inter-subscan time. The observing mode commands the correlator to run subscan sequences.

**Scan** : One ALMA scan, which is composed by a subscan sequence (one or more subscans). TelCal does calibration calculations at the end of every scan.

**Scan sequence** : A sequence of ALMA scans, with optimized inter-scan times.

**VLBI scan** : One VLBI target observation with specific start and end times, part of a longer VLBI session. In ALMA terms this corresponds to one or multiple scan sequences, with scans composed of only one subscan each. Since VLBI data recording will be continuous during an observation, the main reason for this definition is the TelCal processing (*i.e.* the "slow" phasing loop) on scan boundaries.

The flow of data and phasing commands in the loops is then as shown in Figure 3.3. In this diagram, the first few scans of a (much longer) scan sequence for one VLBI scan (brown) are shown. In the initial scan, the phasing system is unphased, in the second, it is partially phased, and with the third scan ALMA is phased to a level where recording of the data for VLBI use is sensible.

From the correlator perspective, it processes each scan and delivers the baseline correlation data to TelCal when complete. However, the early part of each scan was phased according to TelCal's previous instructions and will be flagged so that TelCal solves only the latter portion with the current phasing solution. (*I.e.* it should derive solutions after the last "slow" phase ajustment.) Once TelCal receives a scan, it calculates (purple) a new slow phasing solution and relays it to the CCC for adjusting the TFBs.

Simultaneously, the antennas are providing WVR data which may be used to make phase adjustments on that timescale for delays caused by fluctuations in water vapor. In this diagram, the commanding of TFB phases by the CCC from the two sources are shown as uncorrelated, but in practice these also need to be correlated with adjustments to phase from the ALMA delay server.

Aside from the phase adjustments, the logic in TelCal should be able to calculate the efficiency of the phased sum, and be able to identify problem antennas—*i.e.* antennas which are adding more noise than coherent sum data. Thus on the scan-sequence timescale, the VOM will be in a position to make adjustments to membership in the array of antennas contributing to the phased sum.

Finally, we note that despite these interruptions, the correlator processes the antenna data without interruption of the data flowing to the PIC cards for capture as VLBI scans. The TFB adjustments will produce small glitches in phase for some small number of samples as they are made, but this is a negligible contribution to the ultimate VLBI correlation. Likewise other adjustments (*e.g.* to the set of antennas being phased) will normally not produce significant glitching. The normal ALMA flagging and blanking of data will still, however be in effect.

## 3.4 Detailed Design

In the following sections we present considerably more detail on the design. Section 4 discusses the preparation of the scheduling block and the executable Python script. Then the observing mode controllers are discussed in Section 5. The controllers do their work through the interfaces to the correlator software, covered in Section 6 and the devices, covered in Section 7. And the enhancements to the operator GUIs are discussed in Section 8. After this, there is a brief discussion of contingencies and fault tolerance in Section 9.

The reader is directed to [RD2] for details of the TelCal design and the some of the phasing algorithm and considerations behind the current design.

## 3.5 Design Issues

As this document was being prepared, we have learned that acceptance of our proposed science data model tables into the "official" ASDM tables might not be possible. This is under discussion, and is in some sense a detail (albeit important) for the current design: the data to be exchanged between the affected software agents is understood, and if it proves necessary to repackage and document the exchanges via other pathways (*i.e.* non-standard ones) that is certainly possible.

The current design assumes the capability of accessing TelCal results from the observing mode during scan sequences to be already available. However, this is a feature currently scheduled for early 2014, but can easily be bypassed in the early stages of the project.

# Chapter 4

# Inputs

In this section we discuss the inputs to the VLBI Observing Mode. The VOM has the usual inputs and operating context as the SFIOM, *i.e.* project and scheduling block(s) and observing script. In addition, however the activities in this mode are coordinated with those at other VLBI stations, as described by a VLBI EXperiment (VEX) file. We elaborate on this in the following subsections of this section.

Note that we do not in the scope of the current APP project envision changes to the observing tool used to prepare proposals. That tool is available to generate appropriate SFIOM scheduling blocks as needed, and to support VLBI proposals as might be arranged in future observing cycles.

Note also that VEX is by design very flexible, so many things are (in principal) possible. In the following section, we spell out how we propose to use VEX and point out which capabilities we propose to support. Quite possibly our implementation will work fine for some of the "unsupported cases". (The limitations are largely driving by our requirements, as shown in Section 2.1.)

## 4.1 VLBI Experiment Description

### 4.1.1 VEX Overview

The scheduling of the VLBI array, including ALMA as one or more elements ("stations") of the VLBI array, will at the end of this APP project be driven by a VLBI Experiment (VEX) file. The VEX file describing a VLBI experiment is designed to provide all of the information necessary for each station to set up its receiver frontend and backend electronics, to point the telescope at targets selected by the observer, and to control recording of the voltage signals received by the telescope. In addition, the VEX file contains all of the necessary information for the VLBI correlator to read in the voltage data recorded by all of the participating stations, correct for differences in the local master clocks driving the frequency timing at each station, and perform the correlation of the VLBI experiment.

In general, two distinct VEX files are used in this process. The first VEX file describes what is *supposed* to happen during the observation. It is designed by the astronomer (observer) responsible for the experiment in a way that the astronomer believes will produce the best scientific results from the experiment, within scheduling constraints that may be imposed by the array. Using scheduling tools such as the NRAO `SCHED` program (see `http://www.aoc.nrao.edu/~cwalker/sched/`), the astronomer produces a VEX file describing which astronomical sources should be observed, the times and durations of the observations of each source, and the frequency setups (center frequencies and bandwidths for individual frequency chunks) to be used for the observations of each source. The astronomer submits the VEX file to the organization operating the VLBI array, where a support scientist checks over the file to make sure that the experiment described by the astronomer conforms to the observation allocation allocated to the astronomer, and that there are no major problems with the experiment. For example, the VEX file will be checked to ensure that the experiment lies entirely within the date and time range allocated to the experiment, that the amount of recorded data does not exceed the limit imposed by the array for the experiment, and that the frequency and recording setups specified in the VEX file are valid for each station in the experiment. Once

the VEX file has been approved by the organization operating the array, the VEX file is provided to each station in the array, where a local support scientists (VLBI friends) may perform additional checks to ensure the validity of the VEX file for the control of their respective stations. The VEX file is then provided to local software programs at the individual stations, where the observation as described by the VEX file is converted into instructions specific to each individual station describing how the specific hardware and software interfaces controlling the VLBI stations are to be controlled to perform the experiment.

Following the observation by the VLBI array, a second VEX file is produced to describe what *actually* happened during the experiment. This second VEX file is produced by the VLBI correlator group that will correlate the signals recorded at the individual stations. Using log files produced by the VLBI stations, feedback reports written by the VLBI stations, and general knowledge from the organization operating the array, the first VEX file is edited to describe the electronics and recording setups that were actually used by the participating VLBI stations. Information about the offsets and rates of the local clocks used for frequency and time distribution at each station are extracted from the log files. Identification information for the specific data-recording media used at each of the stations during the experiment is inserted into the VEX file so that the correlator knows how to locate the data for each station during the correlation. Misconfigurations of individual stations (such as accidental swapping of cables for different polarization) are also corrected in the second VEX file at this point. The VEX file is then supplied to the VLBI correlator to set up the operation of the correlation of the VLBI experiment. Initial test correlations on bright "fringe finder" sources are performed by the correlator for each frequency setup to ensure that the data recorded by all of the stations and the setup information contained in the VEX file are valid, and to determine more accurately the clock offsets and rates for each of the stations. The results of these initial correlation tests are fed back into the correlator VEX file, and the full experiment is then processed by the correlator.

The VEX file format is text-based, allowing human editing of the VEX instructions as necessary. Although the goal of the APP is to have the production of a VEX file and interpretation of the VEX file by the ALMA system be completely software driven with no requirement of human modifications to the VEX files in order to get ALMA to observe in VLBI mode properly, it is expected that in the early development and commissioning stages of this APP, significant human editing of VEX files will be necessary for ALMA operation.

### 4.1.2 VEX Version 2.0

The APP will develop software that supports VEX files with a format version of 2.0. (The support of future versions of the VEX standard may require additional development efforts for ALMA, but that is outside of the scope of this APP.) The majority of the VLBI community currently uses the VEX 1.5b1 standard (see http://www.vlbi.org/vex/docs/vex%20definition%2015b1.pdf), plus enhancements to the VEX format that have come about following developments in recording technology and correlator capabilities (VEX 1.5c, see https://safe.nrao.edu/wiki/bin/view/VLBA/Vex15doc). The VLBI community is currently making substantial modifications to the VEX format in an effort to provide experiment definition files that can deal with the capabilities of new high-bandwidth recording systems using multiple recording devices and media devices simultaneously (of which the ALMA observatory is a fine example), that can support the new capabilities of current and planned VLBI correlators, including correlating multiple field centers simultaneously, and that can deal with controlling new telescope systems such as the upgraded VLA and ALMA in a manner consistent with how they are used for their regular, non-VLBI observations. Some information about the development process for the VEX 2.0 standard can be found at https://safe.nrao.edu/wiki/bin/view/VLBA/Vex2.

VEX 2.0 will have several critical advantages over the VEX 1.5 standard for ALMA phasing. First, VEX 2.0 will enable the description of systems for receiver backends and recording systems that use multiple devices that may need to be individually controlled. This is necessary in order to properly describe the multiple Mark6 recording systems that will be used for ALMA VLBI recording, and the multiple disk packs (recording media) that will be used for recording ALMA data for each Mark6 unit when ALMA observes with large bandwidths. A second critical feature of VEX 2.0 is the ability to enter information describing how the astronomer wants the measurement

of individual target-object VLBI scans to be performed. For example, VEX 2.0 introduces a new parameter for describing scans called `intent`, which is directly derived from the `intent` parameter for scans used by ALMA and the VLA to tell the on-line system whether the source is a calibrator or target, whether phase calibration should be applied or not, and so on. There is no suitable equivalent method to describe this information at the scan level in VEX 1.5. (Although VEX 1.5 does have the concept of a `literal block` where arbitrary text information may be supplied that could be parsed by individual VLBI stations as needed, VEX `literal blocks` are not allowed to be present inside of `scan blocks`.) VEX 2.0 will also expand the concept of a VLBI "station" to describe phased arrays that are being used as elements of a VLBI array. Although phased arrays can be included in a basic, limited way in earlier versions of VEX, prior to version 2.0 there was no mechanism to describe the individual elements of the phased array in VEX. Information about which elements of the phased array should actually be used for phasing, or how many elements were actually used for phasing at any given instant in time, had to be provided outside of VEX or a non-standard `literal block` values within VEX. As the gain of the phased array depends on how many elements were used in phasing, this is important for getting good amplitude calibration of the phased array. And as the location of the phased array for determining the $(u, v, w)$-coordinates of baselines to the phased array is usually offset from the delay center of the phased array to be used for determining the station delay during VLBI correlation, information about the locations of the elements making up the phased array, and the list of those elements actually used at any given instant in time are important for observations measuring emission far away from the phase center direction of the observation.

For these and other reasons, APP development will target VEX 2.0 as the VEX standard to use to describe VLBI observations to be performed by ALMA and for correlation of experiments involving ALMA. Unfortunately, the VLBI community has not yet agreed upon a final VEX 2.0 standard definition. This means that the VEX description being implemented by the APP is potentially a moving target. However, it is expected that a final VEX 2.0 standard will be approved by the VLBI community by the time that this APP is to be completed. The VEX 2.0 standard has been in development since 2009, and the draft version seems to be approaching a final version. The APP team made several suggestions for enhancements to VEX in 2012 to better support phased array observations in general and ALMA in particular, and these are expected to become part of the final 2.0 version.

In the meantime, other VLBI stations are migrating to the expected VEX 2.0 standard. For example, the use of the upgraded VLA as a phased-array element of a VLBI station is controlled through a VEX file using features of VEX 2.0 for control of the VLA phasing system, such as the `intent` parameter for turning phasing on and off. The VLA uses a program called `vex2opt` to convert VEX files into instructions that can be read by the VLA Observing Preparation Tool (OPT). VLBI at ALMA will use a similar program, called `VEX2VOM` to convert VEX files into instructions for controlling the ALMA array for VLBI experiments in the VLBI observing mode (VOM). The NRAO `SCHED` program (see http://www.aoc.nrao.edu/software/sched/index.html) supports scheduling the phased VLA for VLBI, producing VEX files with version 2.0 features to control phasing at the VLA. See the guide to VLBI at the VLA on the NRAO website at https://science.nrao.edu/facilities/vla/docs/manuals/obsguide/modes/vlbi for more information.

It is expected that VLBI stations that will co-observe with ALMA will be able to support VEX 2.0 by the end of this APP. During the commissioning and testing period, when some other stations may not yet support VEX 2.0, the APP will strip the version 2.0-specific commands from the main VEX file, and provide those stations with a VEX 1.5 file to control those stations.

### 4.1.3 VEX Details for ALMA Phasing

This area describes the features in VEX 2.0 that will be used to control phasing of ALMA. Only features that describe specific aspects of how phasing will be controlled through VEX are described here.

VEX files are divided into a number of `blocks`. Within each major `block` section, subblocks describing the specific functionality of an individual unit are defined within `def blocks` for most items, or within `scan blocks` for VLBI observation scans.

### Data Acquisition System (DAS)

The DAS block provides information about the electronics systems for backends and recorders that are used for the VLBI experiment. New features in VEX 2.0 will be used to describe the separation of recorded data among the Mark6 recording units and diskpacks used to record the ALMA VLBI signal. DAS `def blocks` will also be used to describe the setup of the ALMA correlator and realtime phasing system, which are effectively the backends for the ALMA VLBI system. Notes on individual parameters for DAS `def blocks` are given below.

`equipment` This parameter is used to define specific pieces of equipment in a backend/recording system. For ALMA, this will be used to specify quadrants of the ALMA correlator that are associated with specific frequency setups and possibly with separate masks of antennas used for phasing the array (hence allowing a crude form of subarraying of ALMA).

`equip_set` This parameter allows specific setup parameters to be passed to the electronics. For ALMA, this will be used to pass parameters for setting up phasing in VOM and TelCal. These parameters may be common for all quadrants of ALMA, or parameters may be specified separately for each quadrant. Note that proper interpretation of these parameters is not likely to be fully implemented in this project. See Section 4.1.5 for more details regarding possible parameters to pass to ALMA to control phasing.

### Frequency Setup (FREQ)

The FREQ block provides information on the frequencies of the basebands that are to be set up. For this project only standard frequency setups tested and approved by ALMA and the APP will be allowed to be scheduled by the APP software. The ALMA scheduling software will internally handle the necessary instructions to the ALMA system to get the exact frequency setup specified. (For VLBI experiments, the ALMA system will not be allowed to find its own frequency settings.)

`chan_def` This parameter specifies the sky frequency of the named baseband for the observation, along with the baseband bandwidth and net sideband (upper or lower sideband for the ALMA digital sampling).

### Mode (MODE)

The VEX MODE block is a collection point to describe the aggregate collection of setup parameters for all stations participating in a common scan of an object. Different MODE `def blocks` can set up a VLBI observing mode for different frequencies, or they can also set up an observing mode using the same frequency with different hardware parameters. This can be used, *e.g.* to enable ALMA to perform one VLBI phasing scan using a standard least-squares solver for the phasing algorithm, and the next scan to be performed using a least-absolute-deviation solver, by using different DAS references in different MODE `def blocks`.

### Schedule (SCHED)

The scheduling block is the place where the VEX file lists the schedule of observations to be made, specifying the source object to observe, and the start and stop times for which each element in the VLBI array will observe that source. VEX 2.0 adds several new parameters to SCHED blocks that are of interest to ALMA phasing.

`intent` The intent parameter is based on the scan intent parameters used at ALMA and the VLA to provide instructions to the online system for processing data from observations scans. For VLBI phasing, the VLA uses the intent values `AUTOPHASE_DETERMINE`, `AUTOPHASE_APPLY`, and `AUTOPHASE_OFF` to control whether or not phasing is applied, with the meanings "actively phase during a scan", "hold phase from a previous active scan", or "do not apply phase offsets", respectively (see http://www.aoc.nrao.edu/~cwalker/sched/Scheduling_VLA.html).

The ALMA VLBI scheduler will use the `intent` identifiers to control phasing with possible values of `True` or `False`:

- `CALIBRATE_AUTOPHASE`: If `True`, the ALMA system will calculate new phase corrections for each ALMA scan observed for the target source. Note that the current project will only support the value of `True` for this `intent`.
- `APPLY_AUTOPHASE`: If `True`, the ALMA system will apply the last previously determined phase correction values. If `CALIBRATE_AUTOPHASE` is `True`, then the ALMA system will continuously calculate phase adjustments and apply them to the target source during the VLBI scan. If `CALIBRATE_AUTOPHASE` is `False`, then the last phase corrections from the previous VLBI scan with the same frequency setup will be applied. If `APPLY_AUTOPHASE` is `False`, then no phase corrections will be applied. Note that `CALIBRATE_AUTOPHASE` may be `True` while `APPLY_AUTOPHASE` is `False`—in this case the ALMA system will monitor phase corrections that it calculates should be applied, but will not actually apply them. Also note that this project will only support the value of `True` for this `intent`.
- `CALCULATE_SOURCE_GEOMETRIC_PHASE`: If `True`, the ALMA system (currently TelCal) will calculate phase corrections to correct the delay model used by the ALMA correlator. This may have uses for at least two circumstances. First, the ALMA phasing system does not fully correct the total geometric delay for each ALMA antenna in some modes. The ALMA correlator delay model may be quantized, leading to 20° phase jumps as a function of time. When this `intent` is `True`, the ALMA phasing system (not the standard ALMA correlator delay compensation system) will predict phase offsets to correct for this quantization problem. (This is expected to only be a problem for the largest AMA configurations with large distances from the ALMA delay reference center, where multiple quantization jumps within an ALMA scan may happen.) The second circumstance when this `intent` may be desired is when the ALMA antenna pointing direction is required by the ALMA system to be the same as the ALMA correlator phase center direction, but the observer wishes to have the VLBI beam located in some other direction. This could be used, *e.g.* to use one quadrant to phase up ALMA on a bright in-antenna-beam calibrator in one correlator quadrant, and to phase up the VLBI beam on a weak target somewhere else in the ALMA antenna beam in a different correlator quadrant. Note that this project will only support the value of `False` for this `intent`.
- `APPLY_SOURCE_GEOMETRIC_PHASE`: If `True`, the ALMA system will apply the last previously determined source geometric phase corrections. This `intent` probably only makes sense when `CALCULATE_SOURCE_GEOMETRIC_PHASE` is also `True`. Note that this project will only support the value of `False` for this `intent`.

`station_array_weights` This parameter allows the weighting of each element of a phased array for the contribution to the phased array sum to be specified. This enables the astronomer to select a "mask" describing which ALMA antennas should be used for phasing for each scan, should the user wish to change the phased array elements per scan. It also allows the VEX file to record which ALMA antennas were actually used to generate the VLBI signal during observations, and this information can be used by the VLBI correlator for proper amplitude calibration of the VLBI data stream. The antenna mask can be specified separately for different ALMA quadrants. Note that use of this information for ALMA scheduling is not planned for the current project.

### Site (SITE)

The VEX SITE block has proposed changes for VEX 2.0 to describe the constituent elements of a phased array.

`site_type` This parameter describes the type of station that is being described. In addition to the previously allowed values `fixed` and `earth_orbit`, the APP team has proposed adding the type `fixed_array` to describe phased arrays.

`site_array_stations` This proposed parameter would provide for a phased array site the list of all antenna elements making up the phased array. Further selection of which antenna elements are to be used for individual scans is described in Section 4.1.3.

#### Source (SOURCE)

The SOURCE block provides a mechanism to describe not only the direction of a source (right ascension and declination), but also the source structure (CLEAN components) and flux density as a function of frequency of the sources. VEX 2.0 will add the capability to provide polarization information, as well as to specify moving sources such as planets (which cold be used for calibrating antenna performance parameters for ALMA) and sources fixed on the Earth's surface (which would be used for scheduling observations of the polarization BEACON for ALMA). The following parameters are of particular interest for ALMA VLBI scheduling.

`source_model` This parameter allows 2-D Gaussian CLEAN components with their corresponding Stokes flux-density values, to be specified. Multiple uses of this parameter specification may be used to specify multiple CLEAN components. The list of CLEAN components will be used by the ALMA scheduler to provide the TelCal phasing software with a description of the source model for sources that are resolved by ALMA interferometry.

`source_type` This parameter specifies what kind of source is being described. For VEX 2.0, allowed values are `star`, `earth_satellite`, `ephemeris`, and `tle`, representing sources with fixed right ascension and declination (allowing for proper motion), Earth-orbiting sources that can be described by an orbit model, sources that are described by a table of geocentric Cartesian coordinates (and velocities) in the J2000 frame (this can easily be used to describe planet sources using the NRAO `SCHED` program's interface to SPICE planetary models), and sources described by two line element (TLE) orbit models for Earth-orbiting satellites. A proposal for VEX 2.0 is to add a category `ITRF` to specify target sources such as the ALMA polarization BEACON that are fixed on the Earth's surface.

### 4.1.4 ALMA and VLBI Calibration

In addition to measurements of astronomical target sources, numerous sorts of calibration measurements need to be performed as part of an experiment, to calibrate the ALMA system (*e.g.* to perform pointing scans to improve the pointing accuracy of individual ALMA antennas), to calibrate the ALMA interferometry data (*e.g.* to observe ALMA amplitude calibrators), and to calibrate the VLBI observations (*e.g.* to observe bright fringe-finder sources). One of the goals of the VEX2VOM development will be to build in automatic ALMA calibration scheduling, to that necessary measurements for ALMA calibration will be performed during "gaps" in the VLBI schedule provided by the VEX file. However, the observer writing the VEX file should have the full ability to control ALMA calibrations by scheduling observations of appropriate sources in the VEX file and providing the appropriate ALMA `intent` parameters for these VLBI scans. The ALMA system is internally driven by the `intent` parameters, and the necessary calibration functions will automatically be performed when these scan `intents` are passed (correct operation requires observing an appropriate source, of course).

#### ALMA Scan Intents

The list of ALMA scan `intents` and their meanings are given below (taken from http://aramis.obspm.fr/~alma/Enumerations/ASDMImpl-2009-01-B/idl/namespaceScanIntentMod.html and http://casa.nrao.edu/Release3.4.0/docs/doxygen/html/CScanIntent_8h_source.html):

| | |
|---|---|
| `CALIBRATE_AMPLI` | Amplitude calibration scan |
| `CALIBRATE_ATMOSPHERE` | Atmosphere calibration scan |
| `CALIBRATE_BANDPASS` | Bandpass calibration scan |
| `CALIBRATE_DELAY` | Delay calibration scan |
| `CALIBRATE_FLUX` | Flux measurement scan. |

| | |
|---|---|
| CALIBRATE_FOCUS | Focus calibration scan. Z coordinate to be derived |
| CALIBRATE_FOCUS_X | Focus calibration scan; X focus coordinate to be derived |
| CALIBRATE_FOCUS_Y | Focus calibration scan; Y focus coordinate to be derived |
| CALIBRATE_PHASE | Phase calibration scan |
| CALIBRATE_POINTING | Pointing calibration scan |
| CALIBRATE_POLARIZATION | Polarization calibration scan |
| CALIBRATE_SIDEBAND_RATIO | Measure relative gains of sidebands. |
| CALIBRATE_WVR | Data from the water vapor radiometers (and correlation data) are used to derive their calibration parameters. |
| DO_SKYDIP | Skydip calibration scan |
| MAP_ANTENNA_SURFACE | Holography calibration scan |
| MAP_PRIMARY_BEAM | Data on a celestial calibration source are used to derive a map of the primary beam. |
| OBSERVE_TARGET | Target source scan |
| CALIBRATE_POL_LEAKAGE | Calibrate the polarization leakage? |
| CALIBRATE_POL_ANGLE | Calibrate the phase/delay difference between polarizations using a source with known polarization properties? |
| TEST | Test? |
| UNSPECIFIED | Unspecified scan intent |

(Subscan `intents`, which may be used internally by the VEX2VOM system, can be found at http://aramis.obspm.fr/~alma/Enumerations/ASDMImpl-2009-01-B/idl/namespaceSubscanIntentMod.html and http://casa.nrao.edu/Release3.4.0/docs/doxygen/html/CSubscanIntent_8h_source.html.)

### VLBI Scan Intents

At this time, there are no scan `intents` agreed upon by the VLBI community. However, this may change in the future, and observers should be prepared to insert VLBI scan processing `intents` such as the following `intents` taken from https://safe.nrao.edu/wiki/bin/view/VLBA/Vex2doc:

| | |
|---|---|
| VLBI_CALIBRATE_AMPLI | Is this source usable as an amplitude calibrator? |
| VLBI_CALIBRATE_BANDPASS | Is this source usable as a bandpass calibrator? |
| VLBI_CALIBRATE_DELAY | Is this source expected to be useful (strong and point-like enough) for delay (fringe) determination? |
| VLBI_CALIBRATE_PHASE | Is this source usable as a phase calibrator? |
| VLBI_CALIBRATE_POLARIZATION | Is the source suitable for polarization calibration? |
| VLBI_CALIBRATE_POLARIZATION_ANGLE | Is this source suitable as a polarization angle calibrator? |
| VLBI_CALIBRATE_POLARIZATION_LEAKAGE | Is this source suitable as a D-term calibrator? |
| VLBI_CLOCK_CHECK | An intent of this scan is for use as a real-time (or near-real-time) check of delays |
| VLBI_OBSERVE_TARGET | This identifies a source as being an astronomical target of the observation |
| VLBI_QUALIFIER_CODE | An integer to indicate the position in a sequence of scans that may have otherwise similar identifiers. This value will make it into a FITS-IDI file and will be visible within AIPS. |
| VLBI_SPECTRAL_LINE_SOURCE | Is this source expected to be more strongly detected as a spectral line source? |
| VLBI_CALIBRATOR_CODE | Any single uppercase letter can be specified. The blank character is the default for each scan if none is provided. This value will make it into a FITS-IDI file and will be visible within AIPS. |

### 4.1.5 Phasing Parameters

Phasing parameters will be specified in the VEX file through the use of `equip_set` parameters in the DAS.

The mechanisms for passing all the parameters that might be needed are still under study. The CCL language calls are certainly available to pass them in from the observing scripts. That machinery also allows the use of so-called "expert parameters", *e.g.*

```
appParamXXX = int(sb.getExpertParameter('AppParamXXX', default=0))
```

within the Python script.

The setup of the VDIF headers for the PICs also can be tweaked by this mechanism. (The required parameters are determined by the VEX file, *e.g.* number of channels; but they might be modified through this pathway.)

#### ALMA Phasing Parameters

Here a list of probable phasing parameters for use with ALMA is provided. The parameter names provided are the names that would appear in the `equip_set` parameter name fields in the VEX file. These names will be converted to ALMA phasing parameter names by prefixing the name with `AppParam`. VEX2VOM will perform initial parsing of the values provided in the VEX file, to verify that all parameters have valid values during the check-in of the VEX file by ALMA.

**ALMA_interferometer_vis_integration_time**
: Set the integration time of individual ALMA interferometer visibilities. Type: floating point. Units: seconds. Valid range: $0 \leq$ value. Default $= 0$. Note that a value of 0 will instruct the ALMA phasing system to choose a good value.

**ALMA_interferometer_channel_bandwidth**
: Set the bandwidth of individual ALMA frequency channels per ALMA subband for the ALMA interferometer. Type: floating point. Units: hertz. Valid range: $0 \leq$ value $\leq$ subband bandwidth. Default $= 0$. Note that a value of 0 will instruct the ALMA phasing system to choose a good value.

**comparison_antenna**
: Set the comparison antenna for VLBI phasing. Type: string. Valid values: One of the names of the ALMA antennas or one of the special values `near_delay_center`, `near_array_center`, `not_phased`, and `default` Default = `default`. Note that the special values force the ALMA system to decide which antenna to use as the comparison antenna.

**reference_antenna**
: Set the reference antenna for VLBI phasing. Type: string. Valid values: one of the names of the ALMA antennas or one of the special values `near_delay_center`, `near_array_center`, and `default` Default = `default`. Note that the special values force the ALMA system to decide which antenna to use as the reference antenna.

**phasing_algorithm**
: Set the algorithm used to determine the phase calculations for VLBI phasing. Type: string. Valid values: one of `least_squares`, `SVD`, `default`. More valid values may be added in the future should additional algorithms be made available.

**phasing_uv_max**
: Set the maximum $(u, v)$ baseline distance to use when selecting visibilities to use for phasing calculations. Type: floating point. Units: wavelengths. Valid range: $0 \leq$ value $< \infty$. Default $= 0$. Note that a value of 0 will instruct the ALMA phasing system to use all baseline lengths greater than the minimum length.

**phasing_uv_min**
: Set the minimum $(u, v)$ baseline distance to use when selecting visibilities to use for phasing calculations. Type: floating point. Units: wavelengths. Valid range: $0 \leq$ value $< \infty$. Default $= 0$.

**solution_interval**

Set the solution interval (subscan duration) of the phase solution. This sets the length of the ALMA scans during the VLBI scan. Type: floating point. Units: seconds. Valid range: $0 < \text{value} < \infty$, or possibly the special values 0, $-1$, $-2$, and $-3$. Default = 0. Note that a value of 0 will instruct the phasing system to use fixed, default values for the solution interval that may depend on the ALMA band being used and the length of the maximum baseline to be included in the solution.

## 4.2 VEX2VOM Design

`VEX2VOM` will be a package of software that converts a VEX file describing a VLBI experiment involving ALMA into a (Python) script that can be integrated into a scheduling block and executed by the operator to perform VLBI observations at ALMA.

### 4.2.1 Design Overview

The design for `VEX2VOM` is that the observer will deliver to the organization responsible for operating the VLBI array that includes ALMA a VEX file that includes commands to observe with ALMA. This VLBI array organization will then provide to ALMA the VEX file from the observer. This could be handled through a web (HTML) interface, or in early stages of the APP it may be handled through an e-mail attachment to the ALMA VLBI friend. Next, the `VEX2XML` conversion tool is run, reading in the VEX file from the archive and converting it to an XML formatted file. Then, the `VEX2VOM` program is run, taking in the VEX XML file and current ALMA status information and other (human) inputs, and outputting a (Python) observing script and expert parameters included in an ALMA scheduling block. This scheduling block could be stored in the ALMA project archive.

Finally, when it is time to perform the VLBI observation, the ALMA operator will select the appropriate scheduling block from the archive, and instruct the ALMA system to run it. Note that this script is open to human edits as needed throughout this process. The basic sequence of operations including the `VEX2VOM` package and its output scripts is as follows:[1]

1. The observer prepares a VEX file containing commands to control ALMA for VLBI observations.

2. The observer submits the VEX file to the organization operating the VLBI array of which ALMA is an element.

3. The VLBI array organization passes the VEX file on to ALMA.

   project archive.

4. The `VEX2XML` program is run, producing an XML file of the VEX commands as output.

5. The `VEX2VOM` program is run, taking the VEX XML file as its main input. `VEX2VOM` will output an ALMA scheduling block including the (Python) script that interfaces with the VOM.

6. The ALMA VLBI friend will examine the VEX file submitted by the observer and the result of the `VEX2VOM` conversion process (including examining log file outputs of the `VEX2XML` and `VEX2VOM` programs). If the VLBI friend determines that there are problems with the VEX schedule, the ALMA VLBI friend will inform the VLBI array organization about the problems related to ALMA observations with the VEX file. The VLBI array organization will then instruct the observer to prepare a corrected VEX file, and the process returns back to stage 1. Otherwise, the process continues below.

7. At some time before the VLBI observations are scheduled to begin (typically a few days to a few weeks prior to the scheduled start time of the observations), the ALMA VLBI friend will

[1] Some of these details depend on the precise arrangments the Observatory chooses to make for VLBI observations. They are provided to be suggestive rather than proscriptive.

re-run the VEX2VOM program, taking in the VEX XML file from the ALMA project archive as its main input. The VEX2VOM program will also gather current status information about ALMA (current positions of the ALMA antennas, the status of the receivers in each of the antennas, and so on), and use this information to generate optimum parameter settings for VLBI phasing that may not have been specified by the observer (such as the selection of reference antennas). The updated ALMA scheduling block is put into the ALMA project archive.

8. Close to the time when the experiment is scheduled to begin, the ALMA operator will select the scheduling block from the ALMA project archive, and instruct the ALMA system to run it. This needs to be early enough to allow time for any system internal preparations to start the actual observation at the absolute start time.

9. The scheduling block script will perform the initialization of the ALMA system, including sending initial configuration information to VOM, as necessary for the VLBI observations to be performed by the experiment.

10. The scheduling block script will bypass any VLBI scans or calibration activities scheduled prior to the current time, and configure ALMA for the next scan that is scheduled for a time in the future. (This allows the script to be re-run by the ALMA operator should the VLBI observations be stopped for any reason—see stage 11 and Section 9.

11. The ALMA operator and possibly the ALMA VLBI friend will oversee the VLBI experiment and monitor the APP GUI. If the VLBI observations are stopped for any reason, the ALMA operator may go back to stage 8. If a significant problem is determined to be present in the ALMA scheduling, the ALMA VLBI friend may need to edit the VEX file submitted by the observer, and return to stage 4.

12. At the end of the scheduling block, the observation finishes and the script ends.

13. The ALMA VLBI friend will examine the output metadata including log file output from the VEX2VOM scheduling block script (and possibly analyze the ALMA interferometry data), and make a report of the outcome of the VLBI experiment at ALMA. This report will be submitted to the organization responsible for the VLBI array operation.

### 4.2.2 VEX2XML

The VEX2XML converter will read in a VEX 2.0 file and convert it into an extensible markup language (XML) file. An XML schema for VEX 2.0 will be developed as part of this project, and interfaces to standard ALMA programming languages (*i.e.* Python, C++, Java) will be made. For items that currently have existing ALMA schema, we will of course be re-using those. Similarly, the Mark6 recorders currently have a schedule schema for recording a VLBI schedule, so we shall be using that. A conversion program will be written that takes in the VEX 2.0 file prepared by the observer and outputs XML data to a file that will be imported into the ALMA project archive.

#### VEX2XML Inputs

VEX2XML will read a VEX 2.0 file. Note that in principal, a VEX file may reference other (ASCII text) files containing VEX instructions. This may be used *e.g.* to provide a means to specify the current positions of the individual ALMA antennas.

#### VEX2XML Output

The output of this tool is an XML file which will (eventually) be integrated with the existing system (scheduling blocks). However full integration with the ALMA Scheduling system is beyond our scope.

Note also that current practice is to limit ALMA scheduling blocks to less than 2 hours duration so as to avoid memory limits or catastrophic loss of metadata in the event of a component crash. Since VLBI sessions may last 18 hours (requirement APP0370) partitioning the full session into several scheduling blocks in some sensible way must also be arranged.

## 4.3 VOM Scripts

The ALMA scheduling blocks prepared by `VEX2VOM` will themselves include Python scripts that will be able to run within the ALMA system. Existing ALMA scheduling block scripts for the single field interferometry mode (SFIOM) will be used as the starting point for the development of the VOM scripts to be written by `VEX2VOM`. The scripts will perform the basic sequence of events:

1. Set up the basic running environment, with the ALMA experiment name coded into the script. The usual ALMA logging and alarm capabilities will operate normally. The VOM software will direct logs relevant to be retrieved for "station logs" either to an `APP` audience, or at least mark them with an `APP` token. In particular the shift log tool should be used to capture operator comments for use in the VLBI correlation.

2. The current (UTC) time will be checked to ensure that the script is not running too early or too late.

3. The script will read in data from the ALMA system to learn current ALMA system parameters. Information such as the status of the ALMA antennas (which antennas are allocated to the experiment by the ALMA operator, the locations of the ALMA antennas, the system temperatures of the various receivers, and so on), the delay center of the ALMA array, and so on will be read in in order to make decisions regarding the subsets of ALMA antennas to use for VLBI phasing, the reference and comparison antennas to use, and so on. Information describing the current system timing (the delay between telling VOM to start a VLBI scan and the appearance of the first samples coming out of the phasing system, the gap time between ALMA scans performed by VOM within one VLBI scan, and so on) will be imported from the ALMA system to calculate the exact ALMA schedule to use based on the requested VLBI scan times from the VEX input information.

4. VOM will be initialized with information gathered from the VEX input and the status information gathered from the ALMA system.

5. ALMA initialization operations will be performed as necessary.

6. With reference to normal ALMA timekeeping the script will arrange for the next scan to be run. If no scans remain, the script will proceed to step 11. The script will be constructed such that if the observation needs to be aborted (in order to correct some hardware issue), it can be restarted once the problems are resolved—the script will ignore scans scheduled in the past.

7. The script will pass setup information for the next VLBI scan to VOM. This information will include the frequency setup information, the set of ALMA antennas to be used for observations, the set of ALMA antennas to be used for VLBI phasing, the phasing parameters to be used for this scan, the scan intents, and so on.

8. The VOM will then be told to perform the observation of the VLBI scan.

9. When VOM is finished with the VLBI scan, the script will check if GUI directives have been made (*e.g.* to skip the next scan or to proceed to shutdown).

10. The VOM script goes back to step 6.

11. ALMA shutdown actions for the experiment will be performed as necessary.

12. The VOM script ends.

The PICs may be managed by this script as part of the startup or shutdown operations. Since VLBI is not a normal activity at ALMA, and since the PICs consume precious power, they will most likely be turned off between VLBI sessions. A short script can be used to activate them as part of the pre-session activities and deactivate during the post-session activities. The VOM observing script itself can do both of these activities. The PIC software implementation is discussed in Section 6.2.

## 4.4 Metadata Scripts

The VLBI correlator that will process the phased-array data produced by ALMA and other stations participating in the VLBI experiment needs to know exactly how the observations at each station were set up, where the stations are located, when the station observed which source, which basebands corresponding to which frequencies were recorded onto which modules in which locations in the output data format, the offset between the timestamps recorded in the output data and the true reception time of the wavefront (often measured using a comparison of the station maser clock to a GPS-derived reference time), and so on. This information is provided to the VLBI correlator in the correlator VEX file derived from the observation VEX file produced by the observer.

Additional metadata are also required for the conversion of ALMA's linear polarization data into circular polarization data for compatibility with other VLBI stations that traditionally record circular polarization data. The values of the $X-Y$ phase difference, the polarization leakage terms (a.k.a. the D terms), and the relative gain of the $X$ and $Y$ receiver systems for the ALMA VLBI data need to be provided as a function of baseband frequency and time.

Additional metadata including information about the effective system temperatures for the baseband frequencies, the conversion values from correlation units to janskies for the various baseband frequencies, weather data (temperature, pressure, humidity, ...), and so on need to be provided to be placed into the metadata section of the output VLBI data produced by the VLBI correlator.

Finally, the maser drift and offset between GPS and UTC are extremely important.

### 4.4.1 Metadata in ALMA Output

The metadata required for VLBI correlation and further processing of the VLBI data will be stored in the output metadata generated during the VLBI observation at ALMA, or it will be produced by the ALMA VLBI friend using the ALMA data and metadata generated during the VLBI experiment at ALMA, possibly including extra information derived from calibration observations performed by ALMA near in time to the VLBI observation.

Much of the necessary metadata will be recorded in the ALMA metadata stored in ALMA Science Data Model (ASDM) tables generated within the experiment. These tables will include information about the locations of individual ALMA antennas used during experiments, output information from the TelCal phase solver that describes the number of antennas used in the phase solutions, the number of antennas used to form the VLBI signal, the phasing efficiency, and so on.

Additional information will be stored in the log file outputs of the various system components responsible for VLBI observations at ALMA (`VEX2VOM`, the VOM script, VOM, the TelCal phase solver, and so on).

The polarization calibration information needed to convert ALMA data from linear to circular polarization may need to be produced by the ALMA VLBI friend after the VLBI experiment, although some polarization metadata may also be present in the ALMA output metadata. For example, for ALMA receiver bands that have stable leakage terms, the leakage terms may be taken from the output ASDM tables or from the online ALMA system parameter information. On the other hand, for ALMA receiver bands that have leakage terms that are variable on timescales less than the duration of the experiment, or less than the time difference to the nearest regular ALMA calibration measurement, the leakage terms will need to be derived by the ALMA VLBI friend from the VLBI observations themselves. In this case, the ALMA VLBI friend would need to process the ALMA interferometry data produced during the VLBI observation (using standard ALMA processing software and pipelines, such as CASA tools). The derived metadata are needed only prior to the correlation of the VLBI experiment, which may usually take place several weeks following the VLBI observations. However, for special observations requiring rapid correlation (rapid response observations), the processing may need to be performed as quickly as possible.

### 4.4.2 Parsing Scripts

Software will eventually be developed to automatically parse the ALMA metadata output for each VLBI experiment. This software will probably take the form of Python scripts that can automatically be run on the ALMA output data. A design goal is for these scripts to be able to

run at the ALMA operations site, at ALMA regional centers, and at the VLBI correlator sites that will process ALMA VLBI data.

Note that such software development is not scheduled in the current project. We present the description here however, as it is the natural progression from what we are developing, and some of the capability *may* arise out of our commissioning or science activities.

During the phasing development period for ALMA, the metadata gathering will be performed by hand. As practical within resource (developer time) budgets, small parsing scripts to extract specific metadata components may be developed during this project.

### VOM2VEX

A script would be developed that will write out a VEX 2.0 file containing the description of observations that actually happened at ALMA, to be provided to the VLBI correlator.

Ideally, the output VEX file will be based on the input VEX 2.0 file, replacing the VEX statements dealing with ALMA by VEX statements that match the actual manner in which the VLBI experiment was performed at ALMA. Additional VEX statements will be added to the output VEX file to describe things such as the positions of the individual ALMA antennas if this was not specified in the observing VEX file, the list of ALMA antennas used for each VLBI scan and their contributing weights, and so on.

Then, the equivalent of the Unix `diff` program will be run on the original observing VEX file and the output correlator VEX file with updated ALMA information to make a context-based Unix `patch` file. Both the correlator VEX file and the `patch` file will be recorded in the ALMA project archive, and both be provided to the VLBI organization operating the VLBI array including ALMA. The VLBI correlator group may then apply the `patch` file to the original observing VEX file, along with other patch files including update information from other VLBI stations, to generate a correlator VEX file with updated information for all VLBI stations in the experiment.

### VOM2VLBILOG

A script will be developed to generate a single log file combining the log file outputs of all of the ALMA phasing components (VOM script, VOM, TelCal phase solver, GUI, ...) operating during the VLBI experiment. Ideally, for backwards compatibility this log file should use the format of the Field System (see http://lupus.gsfc.nasa.gov/fsdoc/field_system_log_file_format.pdf or http://lupus.gsfc.nasa.gov/software_fs_main.htm).

### VOM2VLBIMETA

A script will be developed to derive ancillary metadata for VLBI calibration from the ALMA metadata. For an example of typical metadata needed for processing of VLBI data, see http://www.aips.nrao.edu/CookHTML/CookBookse65.html. Output metadata files will include the following files:

**ANTAB** ANTAB table entries to describe the phased ALMA amplitude calibration information will be prepared. See http://www.aips.nrao.edu/cgi-bin/ZXHLP2.PL?ANTAB for a description of this file format. Information about the system and antenna temperatures measured as a function of time, and antenna gain curves (with elevation) are provided in this file. Note that the ANTAB format provides system and antenna temperatures indexed by the FITS intermediate frequency (IF) number and polarization sense. As it is not obvious how the VLBI correlator will convert the ALMA frequency channels into FITS IFs, the ANTAB output produced by this ALMA script will include in the comments section prior to the antenna and system temperatures a conversion chart describing the ALMA frequency channels (center frequency, bandwidth, polarization sense) and their index position in the ANTAB file. It is the responsibility of the VLBI correlator group to convert the ALMA ANTAB file into an AIPS ANTAB file, including the conversion to the FITS IF numbers and the conversion from ALMA linear polarizations to circular polarizations.

**FLAG** A table with flag entries to describe the actual times when VLBI data from ALMA are valid will be generated. VLBI data recording may start before antennas have fully settled on-source, leading to problems with amplitude calibration. In addition, the first part of phasing scans usually need to be flagged until the phasing algorithm has determined appropriate phase offsets for the antennas and basebands, and the phasing equipment has applied these corrections.

**Polarization Phase** No standard AIPS table appears to exist which can describe the $X-Y$ phase offset for polarization corrections in a time-dependent fashion. Therefore, an ANTAB-like file will be generated with an `RLDIF` group, similar to the ANTAB `TSYS` and `TANT` groups. Only the `INDEX` and `INDEX2` keywords will be allowed. The format of the `RLDIF` values which follow are:

```
Day_no hh:mm.mm col1 col2 ...etc
```

where the column values give the $X-Y$ phase difference in units of degrees. See http://www.aips.nrao.edu/cgi-bin/ZXHLP2.PL?ANTAB and http://www.aips.nrao.edu/cgi-bin/ZXHLP2.PL?RLDIF for more information. Also, see the note about ALMA versus FITS frequencies in the ANTAB description above.

**WEATHER** A weather table containing information about the weather conditions as a function of time (temperature, pressure, humidity, and so on) will be supplied to improve calibration of the VLBI data. These data may be used, *e.g.* to provide corrections for atmospheric transmission and for corrections to the VLBI delay component for ALMA. (Although the WVR corrections will hopefully help to minimize phase variations between ALMA antennas, the overall tropospheric delay component, which depends on the pressure, temperature, elevation angle, and so on) may need to be removed to provide the best VLBI interferometry data for some projects.

# Chapter 5

# Controllers

## 5.1 Single Field Interferometry Mode

The VOM is an enhancement of the normal SFIOM in that it adds new capabilities (phasing of the array and recording of the phased sum for VLBI use). There are only minor differences in its use of the array. So there will be minor modifications to the existing `InterferometryController` as discussed in Section 5.2 and two new controllers added. The `VLBIController` (discussed in Section 5.3) is responsible for the recording back-end (PICs and recorders), and the `PhasingController` (discussed in Section 5.4) is responsible for managing the phasing system.

For overall reference, a component diagram for the VOM is shown in Figure 5.1. The pinkish boxes represent unmodified components, blue is used for new components, and purple for modified ones. The new components and modifications will be discussed in some detail in the following sections.

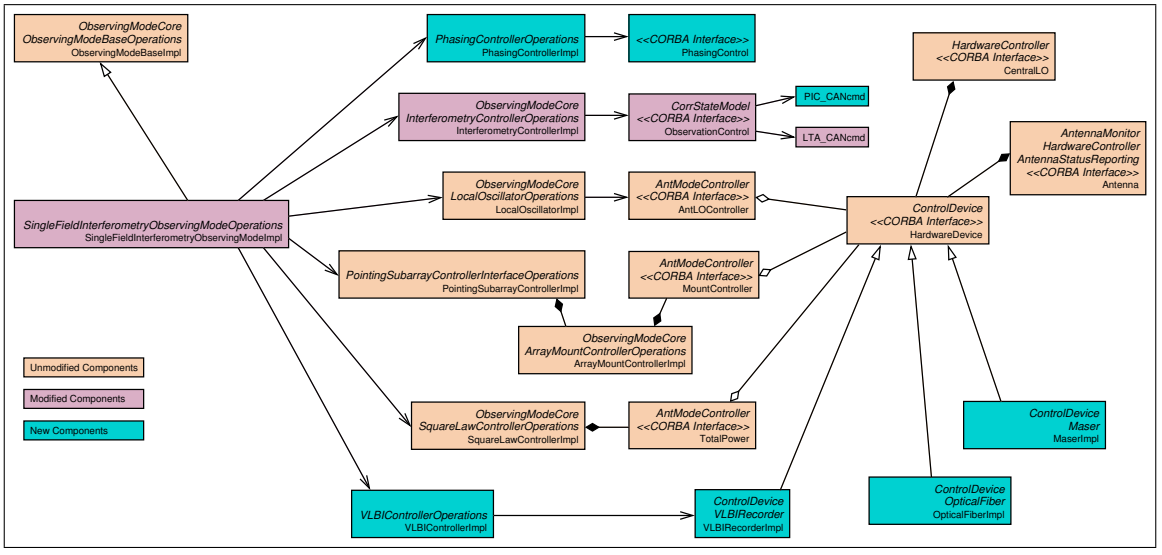


***Figure 5.1:*** *Component diagram for the VOM. This is similar to that of the Single Field Interferometry Observing Mode except for the modification of the* `InterferometryController` *and the addition of two new controllers and VLBI devices. Fragments highlight scan and subscan boundaries.*

There are several workable ways to implement the VOM:

1. Add the VOM as a new observing mode
2. Create it as a subclass of SFIOM

3. Create it as an enhancement of SFIOM

The principal disadvantage of option 1 is that this will require much code duplication, as most features are actually shared. Option 2 is a clean solution, but this places the new code as an adjunct to the existing mode, so this could produce problems during development or future maintenance. Option 3 guarantees that changes to SFIOM (which is still evolving) will automatically be available to the VOM; and also that those changes which might break the VOM will be more obvious during this stage.

Subsequently we will assume that changes are made within the existing SFIOM module, thus we will refer to the modified SFIOM as VOM. However, as a practical matter, the amount of new code to be developed is safely contained within the same new controllers so refactoring of the implementation should not be hard in later stages if so desired.

Note that the underlying interferometry observing mode infrastructure will remain in place, actually producing an ASDM and binary data set in the archive, just as usual, which will be still compatible with normal reduction tools.

On a syntactic note, we shall generally use "`App`" or "`APP`" as a token in the code to make it easy to track the APP enhancements to the ALMA code. The token "`VLBI`" could also be used; however the system we are building allows phasing of the array for purposes other than VLBI.

Before moving to the individual controllers we present some operations and nomenclature considerations.

### 5.1.1 Operations Concept

The `VEX2VOM` tool will have parsed the VEX file to determine the targets, tunings, VLBI scans and so forth, and prepare an observational script (see Section 4) such that these capabilities are accessed normally.

When the VOM is instantiated, one of the first tasks is the creation of its controllers. The two new controllers would not be created until specialization of the mode to phasing and/or VLBI activities is indicated. Note that the main purpose of having two new controllers (rather than one) is to cleanly separate the software responsibilities between these two activities so that for testing (if nothing else) they may be run independently.

Figure 5.2 shows the individual actions performed by each controller during initialization and one VLBI scan, which will be followed through in the next sections. Some of the pseudocode here makes use of functions or interfaces that will be presented in still more detail in Appendix C.

As the observation script commands the controllers through the `SingleFieldInterferometryObservingMode` interface, a few changes are required at this level (see SFIOM commands described in Figures 5.2 and 5.3):

- Methods to activate and configure the Phasing and VLBI controllers are to be added.
- A new `doAppScanSequence` command will start a scan sequence corresponding to one VLBI scan. It will eventually make use of the existing `doScanSequence`, but adding a few extra setups and verification steps. Correspondingly, the script might want to command additional calibrations.
- Methods to support operator queries and commands from the VLBI Observation Status GUI (described in Section 8.1).

### 5.1.2 Array Nomenclature

To avoid confusion, we present here a few terms which will turn up in various places regarding the controllers:

**array** the full set of antennas assigned at the creation of the VOM, it is provided as a list of antenna names in some unspecified order; the correlator converts this list internally into CAI (correlator antenna input) assignments; it has `N_array` members, and includes:

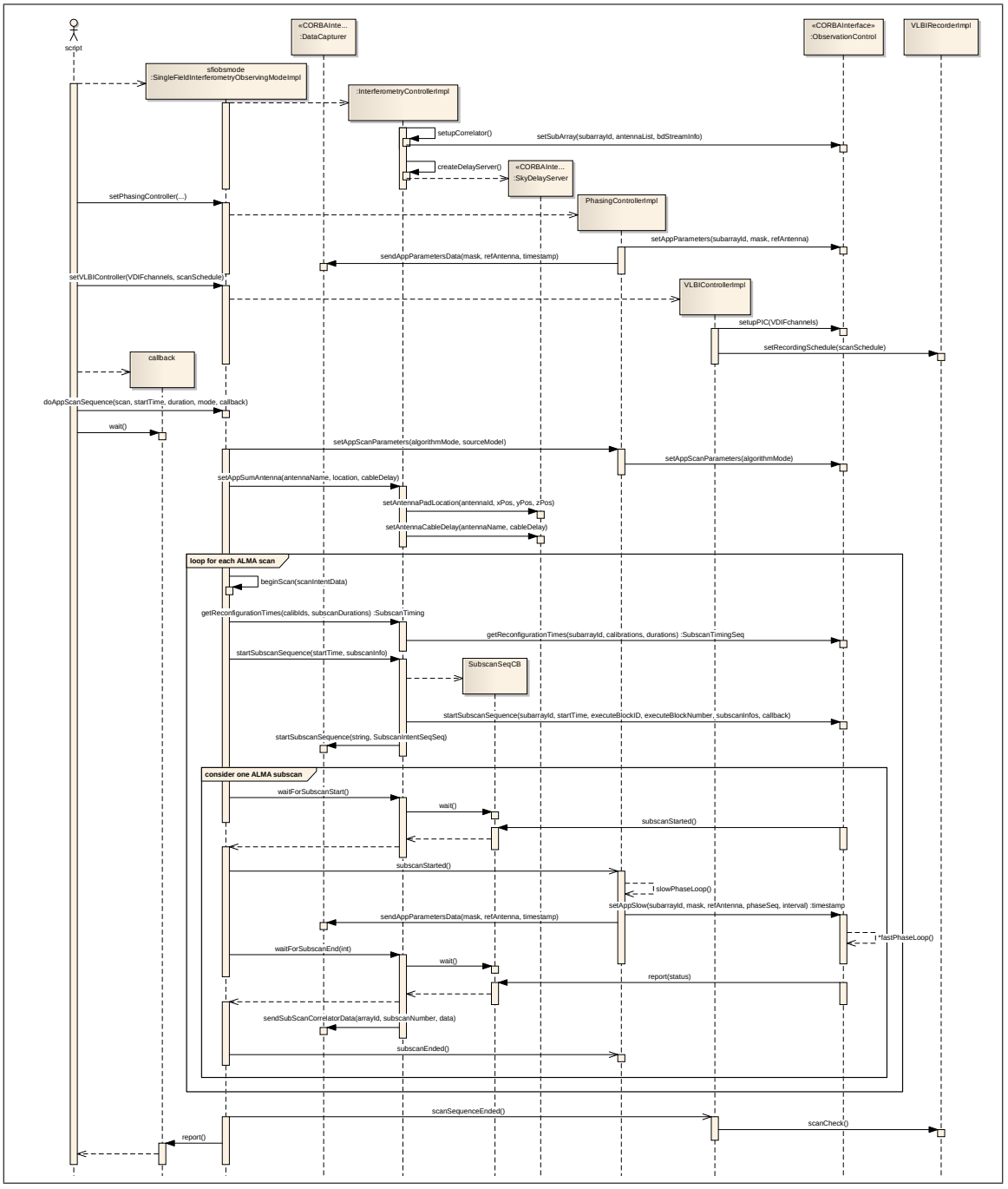


***Figure 5.2:*** *VLBI scan sequence diagram from the VOM perspective, including new* `VLBIController` *and* `PhasingController`*. The observation is initiated by an observing script. Fragments highlight ALMA scan and subscan boundaries inside a single VLBI scan. The* `slowPhaseLoop` *and* `fastPhaseLoop` *tags are further expanded in Figures 5.4 and 5.5.*

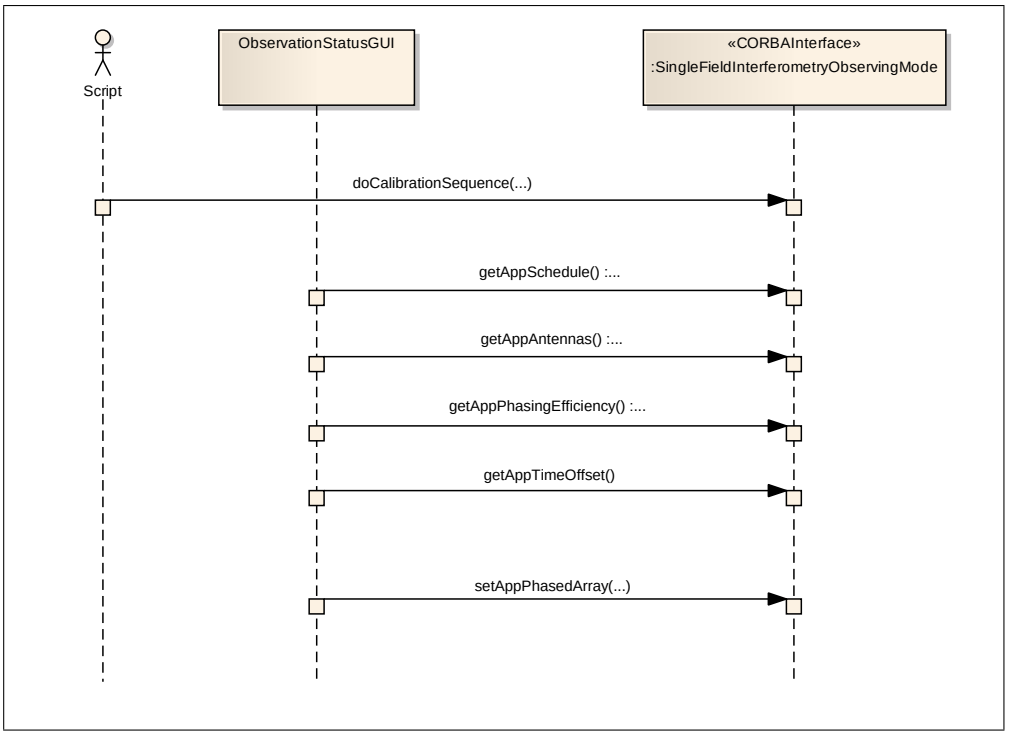


***Figure 5.3:*** *Optional VOM commands that can be issued at any time during an observation, thus not necessarily part of a normal lifecycle. Each message group represents one possible (independent) request process. Clients include the observation script and the VLBI Observation Status GUI. Input and output parameters are yet to be defined, according to specific client implementations.*

**phasedArray** the subset of the array which will receive phase corrections ($N_p$ members; $N_p$ must be odd for the recorded data to be usable).

**refAntenna** a member of `phasedArray` which is used to fix the ALMA array phase (*e.g.* by being assigned 0 phase in the solution); it may change dynamically.

**cai63Antenna or phasedSumAntenna** the antenna input appearing to the correlator at CAI-63. There may or may not be a physical antenna present at this input, but in VLBI operations it is always replaced with the phased sum data that is also sent to the PICs.

**compAntenna** refers to a distinguished member of the `array`, not included in the `phasedArray`. It is used to measure ("compare") the efficiency of the phased sum through its correlation with the sum. (The correlation scales as $sqrt(N_{\text{phased}})$.)

**compareArray** Since there may be several antennas assigned to the observation, but not actively being phased, we in effect have an array of potential comparison antennas.

From the above it is obvious that a minimum of three available antennas (plus CAI-63, with or without a physical antenna) is needed in order to operate the phasing system.

Note that "subarray" refers to a set of antennas which is used in an independent observation. While the `phasedArray` is, strictly speaking a subarray of the array, this is confusing terminology which we shall now avoid.

Note also that the use of the antennas by the phasing system can in principal be different for each of the four quadrants. (*I.e.,* each quadrant processes a different sub-band of the chosen receiver, and the optimal phasing might, in principal be different.) Thus decisions about the `phasedArray` are made on a per-quadrant basis.

Finally, different domains of ALMA software use different local methods to refer to antennas: names, CAIs, ASDM tags, *&c.* New software will use the nomenclature of the appropriate domain; but at every boundary crossing (*e.g.* VOM to Telcal) integration testing must ensure that the intended mappings are not compromised.

## 5.2 Interferometry Controller

This existing controller needs to be modified to accommodate a few minor changes for the APP. It is created immediately when the SFIOM is.

```
InterferometryControllerImpl intController;
```

There are two areas where different behavior is needed as discussed in the next subsections.

### 5.2.1 Management of the Array

The SFIOM is presented with an array of antennas to use. The VOM will use the same antennas, but with one significant change: the phased sum will replace the data that would normally be present at CAI-63. Thus, Scheduling and Control must be able to include that antenna in the array, and the CDP nodes must be able to calculate baselines to it. So, some special considerations apply to this antenna:

- The antenna at CAI-63 must not be commanded.
- If no physical antenna is actually present at CAI-63, the software configuration must be modified so that one is.
- The "pad position" (close to the center of the phased array) and "cable delay" (the effective clock cycles needed to calculate the sum) for the antenna at CAI-63 need to be changed in the `DelayServer` at the beginning of an observation. Methods to do so are already implemented in the `DelayServer`.

### 5.2.2 Reliable scheduling of the scans

VLBI requires that scans at all sites be executed at the same time. This means that antennas must be on source before the nominal start time, and if a phasing system is in use, that it is initialized before recording starts. The antennas may go off source immediately following the stop time. The time between VLBI scans is available for local calibration activities to the extent that it does not interfere with the scans. (These would be inserted into the observing script by `VEX2VOM`.)

The VEX schedule specifies when all the VLBI stations are recording the same scans. We will need to ensure that adequate time between these exists so that ALMA has enough time for required operations (*i.e.* enough time to tune the LOs and point to the target source). This requires defining a reasonably upper bound for the scan setup time, `margin`; then the scan starts are moved forward by `margin`, and the scan durations (as presented to the `ScanExecutor` helper class) are extended by `margin`. If retuning of the LOs is required between scans, a larger `margin` would be supplied by the script.

So we need a modified `ScanExecutor`, *e.g.* `AppScanExecutor`, or just a method (*e.g.* `doAppScanSequence`) in `ScanExecutor`, which can be aware of the modified schedule and wait until the desired absolute time (*i.e.* `VEX_scan_start_time` – `margin`) before proceeding to execute the sub-scan sequence. (The normal `ScanExecutor` performs the scan as soon as possible to maximize use of the observatory.)

Note that if the requested scan sequence start time cannot be met (within a certain margin of tolerance), or a commanded start time is already in the past (*e.g.* an aborted script that is attempted to be resumed), it should not be started and fail. The script should then attempt to command the next available scan sequence.

## 5.3 VLBI Controller

The new controller will be added to SFIOM:

```
VLBIControllerImpl vrc = null;
```

The assignment is initialized to `null` so that these capabilities are not available until requested through the observation script.

### 5.3.1 Management of the PICs

At the beginning of a VLBI session, the PIC pair for each quadrant to be used must be activated and programmed to generate accurate VDIF headers. This functionality is encapsulated in the correlator software and initiated through a single `setupPIC` command, with a structure containing channel data coming from the VEX file. The underlying details are discussed in Section 6.2.

Each PIC has some monitoring capabilities: it can capture statistics on the states for all of the TFB channels, and it can measure the offset of the observatory PPS from the test GPS PPS and the Maser PPS. Periodically during the VLBI session, the VOM should check on these to issue corrective actions or inform the operator about abnormal situations. The corresponding query interface is described in Section 6.5.

There is a validity bit in the outgoing data packets; this may be toggled by the observing mode in response to, *e.g.* phasing issues. Additionally, packets can be turned off if the anomalous conditions are likely to not be resolved quickly. Section 9 elaborates on system fault tolerance and corresponding actions.

### 5.3.2 Management of the recorders

Each quadrant has an associated Mark6 recorder connected to its PICs via the optical fiber link system. (*I.e.* each recorder gets data from one polarization pair on 2 private network links.) The four recorders are thus managed by quadrant.

The VLBI Recorder device (Section 7.3) will provide access to the VSI-S (VLBI Standard Interface for Software) layer to operate the recorder:

- Recorders can be given a schedule to record.
- Recorders can be given a scan to record.
- Recorders can abort recordings in process (schedule or scan).
- Recorders can provide status following each scan.

Operators will have tasks related to module management. The recorder status should provide usage information which can determine when the modules will need to be swapped. It should also provide data on the health of the recorder itself.

There are a setup task at the start of a VLBI session (set up the recorders for a session and give them the schedule) and one periodic monitor task (scan checks). These are further detailed in Section 7.3.

Note that it is important to abort the recording schedule in the recorders if the observation script is aborted due to abnormal situations, *i.e.* right before deactivating the `VLBIController`. For discussion on conditions that might lead to such a situation see Section 9.

## 5.4 Phasing Controller

The new controller will be added to SFIOM:

```
PhasingControllerImpl vpc = null;
```

Again, the initialization is to `null` so that these capabilities are not available until requested.

In practice the four frequency bands (correlated in the four quadrants) are independent, so the `PhasingController` will do its work on a per-quadrant, per VLBI-scan basis. TelCal operates on an ALMA scan basis. In the next sections we summarize some of the activity in the phasing loop and the participation of TelCal and Correlator.

### 5.4.1 The Phasing Loop

The phasing system uses the channel-average data from every baseline in the `phasedArray`, provided to the correlator as an antenna mask. This data is flushed to TelCal (where the phasing system intelligence resides) on subscan boundaries, however the metadata for the scans is only available at scan boundaries. Thus we must have one subscan per scan. At present, subscan durations might be *e.g.* 2–30 s, but there is setup work (presently, about 1.5 s) between subscans which reduces the data seen by TelCal. This setup work can be eliminated for VLBI which has no need to reprogram the correlator between subscans. Note that this overhead impacts the ALMA data, but since the correlator runs continuously, it isn't paid in the VLBI data.

From the channel-average data, the phasing system can estimate the phase corrections to apply (via tunable filter bank commands) so as to bring the sum of signals from all antennas in the mask into constructive interference. Typically there will be one or two channel-averages per frequency band—one is sufficient for a phase correction across all bands, two are sufficient for a delay-like phase slope across the bands. The VOM receives this information as being published in a new ASDM table (`CalAppPhase`) by TelCal on the "slow" timescale. Included in `CalAppPhase` are also measures of the quality of the phasing solution. The VOM then is in a position to discard antennas from the `phasedArray` if it would improve the resulting sum.

Note that TelCal is largely stateless—it processes scans which are marked in the VEX input as to be phased, and ignores other scans. Thus any memory of processing from one scan to the next resides in the `PhasingController` (or its client `ObservationControl`). In particular, pre-observation activities should establish the delay between the two polarizations, so the `PhasingController` will need to be instructed of this and pass this delay along to `ObservationControl`.

This is the "Active" mode TelCal operation (specified as a scan intent). There is also a "Passive" mode for TelCal, where it does not perform any fits, but merely updates the last fit solution for changes in the source model. These are specified by scan intents (in the VEX file; transferred to the correlator subscans) which wake up the TelCal phasing engine to process each correlator subscan. For completeness, there is also an "Idle" mode where the TelCal phasing engine does not do any

processing. As TelCal publishes its data, it is the `PhasingController` which is responsible for maintaining state between scans.

On the correlator side of the phasing loop, there are several options on how to process and apply corrections. The most important is the WVR correction which is where the CDP nodes calculate delays on the "fast" timescale (about once per second). Depending on the weather, it may or may not be sensible to apply this correction. Note that the baseline-based WVR correction that the CDP nodes currently perform should in all cases be turned off. (The correction that it normally performs hides the effects of water vapor from TelCal, and moreover does not assist coherent summation of the antenna data. In this case, the WVR correction is made on a per-antenna basis, either to all antennas, or a correction relative to the reference antenna.)

A secondary one is the residual delay correction (RDC; that part of the geometric delay which cannot be controlled in the hardware). The hardware delay corrections are made as needed (subject to the restriction of once per TE, *i.e.* 48 ms, and at a specific ms within the TE interval). As these corrections are made, the phase of a TFB at the affected antenna will jump discontinuously by about 11−22 degrees, depending on where it is in the band. The RDC is made in the CDP nodes to remove the `average` of this effect over the integrations from the baseline visibilities. For an array with all short baselines and a relatively fast "slow" timescale, this correction should (probably) be turned off so that TelCal can see the average effects and correct them. In an array with long baselines and a relatively slow "slow" timescale this should (probably) be turned on so as TelCal will be offering corrections which will be too late to be useful. (That is, unless a suitable predictive capability can be developed. After all, the geometric delay is known, so a more highly developed TelCal implementation than we propose could take this into account.)

Finally, there are (in general) algorithmic choices that can be made for how (and where) to solve for the phasing corrections. For example, TelCal can pass the source model phases to the CCC and the CCC can have the CDP nodes do the fitting.

We have laid out an architecture here which may be adapted to many situations, but in the current scope of the APP we are not likely to implement or commission all of the possibilities. The result is effectively an enumeration of phasing modes:

**Active/Passive/Model** informed by scan intent:
TelCal generates slow phases from current data (Active), from previous data (Passive), or just provides source model phases (Model)

**WVR/no-WVR** informed by script/SB input:
CDP does/does-not generates fast WVR phases, and the correction is either absolute, or relative to the reference antenna.

**RDC/no-RDC** informed by script/SB input:
CDP does/does-not apply residual delay correction

**Simple/Complex** informed by script/SB input informed by script/SB input:
CDP does only the WVR or RDC corrections, or does its own fit.

This architecture for the phasing loop should be general enough to accommodate all phasing scenarios. The APP project scope is limited to phasing-up on bright sources, so we are not committed to implementing all cases until there is an established need. (*I.e.* the "Complex" case where the CDP nodes do the fitting work is not likely to be implemented.) Work on characterization of conditions at ALMA are ongoing and should inform our development.

### 5.4.2 VOM and TelCal

The "slow" part of the phasing loop is primarily a conversation between the VOM and TelCal, mediated through `ExecutionState`. The timing diagram in Figure 5.4 shows this portion of the phasing loop, together with some of the initialization activities. The internal TelCal design is further described in [RD2].

There are three TelCal interfaces that participate in this process:

**ReceivingDataManager** Defines the TelCal bulk-data receiver component that subscribes to the correlator data stream.

**GetTelCalResults** Defines an interface used by other subsystems to get the TelCal calibration results.

**ParameterTuning** Allows to set and get parameters used by the TelCal subsystem.

When the VOM starts a scan it will inform the `ReceivingDataManager`, which will parse the observing intents and receive the channel-average data from the correlator. Once a `ScanProcessedEvent` is received, results are calculated and published through `GetTelCalResults`. `DataCapturer` and `ExecutionState` consume the `AppPhaseReducedEvent` and request the full results from TelCal.

`DataCapturer` is the component in charge of gathering the observation metadata from all subsystems and build the corresponding ASDM tables, for usage during offline analysis. It can also return table "slices" to TelCal on demand. To cover VLBI needs two new tables are to be added to the data model (fully described in Appendix A):

**CalAppPhase** Contains phase corrections and measures of the quality of the phasing solution from TelCal.

**AppParameters** Contains antenna mask, referenceAntenna, *&c.* from the `PhasingController`.

In the meantime the VOM will have requested the latest phase results to `ExecutionState`, which are returned as soon as available. The VOM will then analyze the results and determine if antenna mask adjustments are needed. Finally it will forward the results to the correlator, and additional metadata to `DataCapturer`. Note that this will effectively occur after a scan has ended and the next one has already started. In order to consider only data with the latest correction for the next TelCal reduction the beginning of the scan will be flagged as invalid.

Note that TelCal has access to the ASDM data through `DataCapturer` during an observation. Therefore, metadata such as latest phased and comparison array and reference antenna, determined by the VOM, can be accessed through that interface.

### 5.4.3 VOM and Correlator

The "fast" loop is a conversation between the VOM and the correlator, through the `ObservationControl` interface (controlled by the CCC), as shown in Figure 5.5.

Right after the initialization of a new correlator subarray, the VOM will set the relevant parameters for a VLBI observation, including antenna sum mask, reference antenna and algorithm mode.

There is one LTA Protocol which controls whether the phased sum or a physical antenna appears on CAI-63. This switch will be turned on at the beginning of a VLBI observation (and off otherwise). The sum signal then needs to be programmed: the participating antennas need to be specified, and the scaling of the sum adjusted. Thereafter the PICs will receive a valid phase-sum signal. This programming is described in more detail in the section on the LTA protocols (Section 6.1).

In general, three different algorithm modes are defined from the correlator perspective:

**Simple slow** Only TelCal ("slow") corrections are applied as soon as received by the CCC.

**Simple fast** TelCal corrections are applied, and the CDP calculates WVR corrections during the scan, which are added to the "slow" results and applied in regular "fast" intervals.

**Complex fast** In addition to the WVR the CDP also calculates the phase fit, based on the source model.

Optionally, residual delay corrections can be turned on or off as well.

At the end of every subscan phase corrections will be calculated by TelCal and forwarded by the VOM to the CCC via the `setAppSlow` command. Based on this the CCC will immediately apply the corrections determined by TelCal to the TFBs (this essentially completes the "slow" part of the loop).

The CDP will start flagging data as invalid for TelCal at the beginning of every subscan (except the first one). This data is "invalid" insofar new phase corrections are applied once the subscan

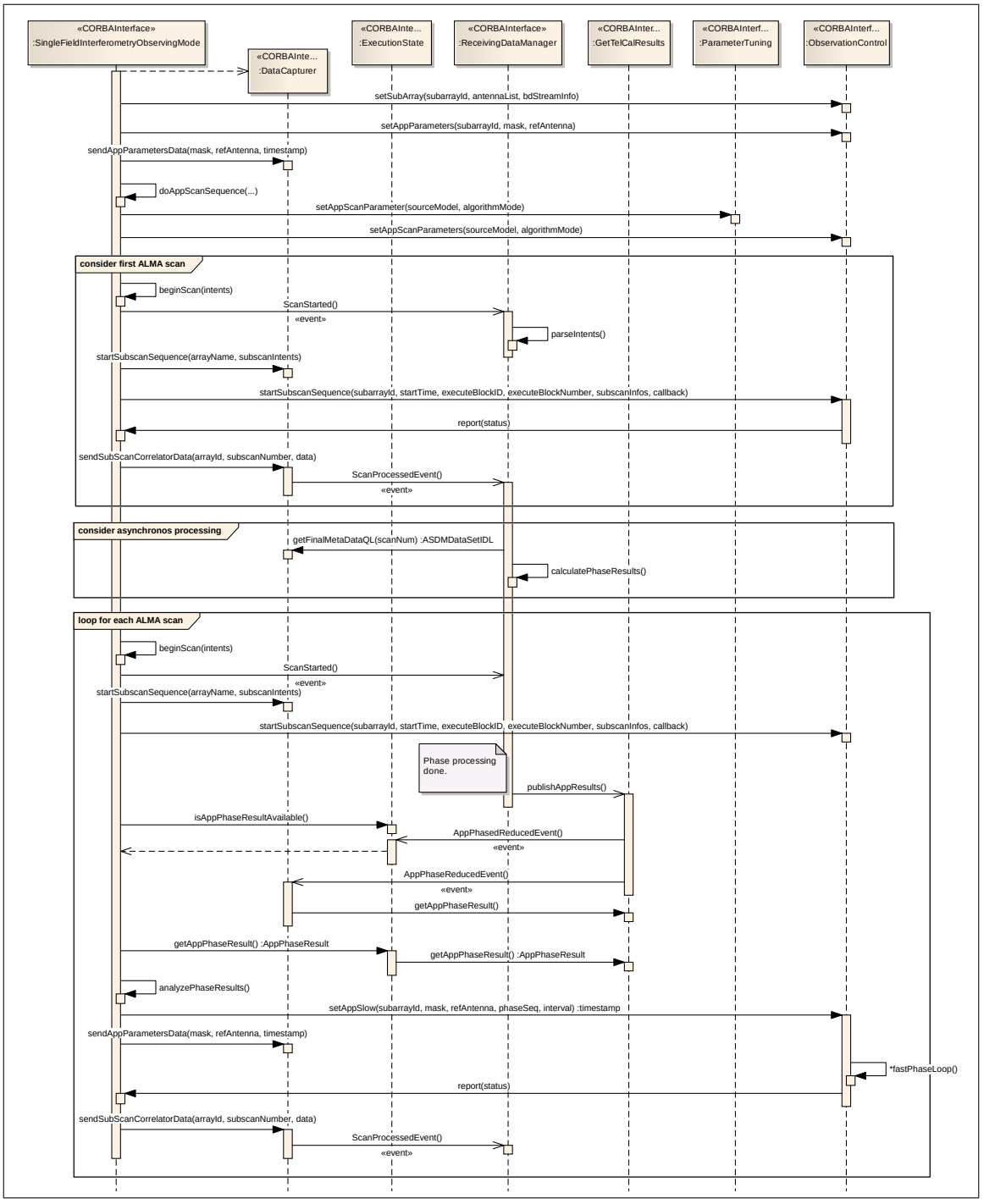


***Figure 5.4:*** *VLBI scan sequence diagram between the* ***VOM*** *and* ***TelCal****, representing the "slow" phasing loop. Fragments highlight the first (unphased) scan, the asynchronous phase processing (overlapping with scans) and the loop for every subsequent ALMA scan.*

has already started; thus the first part of that subscan will contain none or old (from the previous subscan) corrections that would introduce noise to new TelCal calculations (see phasing scans in Section 3.3). Once the slow corrections are applied by the CCC it will inform the CDP to stop flagging the data. Depending on the algorithm mode in use the CCC will also forward additional data needed for the fast loop.

If in "fast" mode the CDP will calculate corrections based on the initial TelCal corrections and/or the source model on regular intervals (typically once per second) and send them back to the CCC for application on the TFBs.

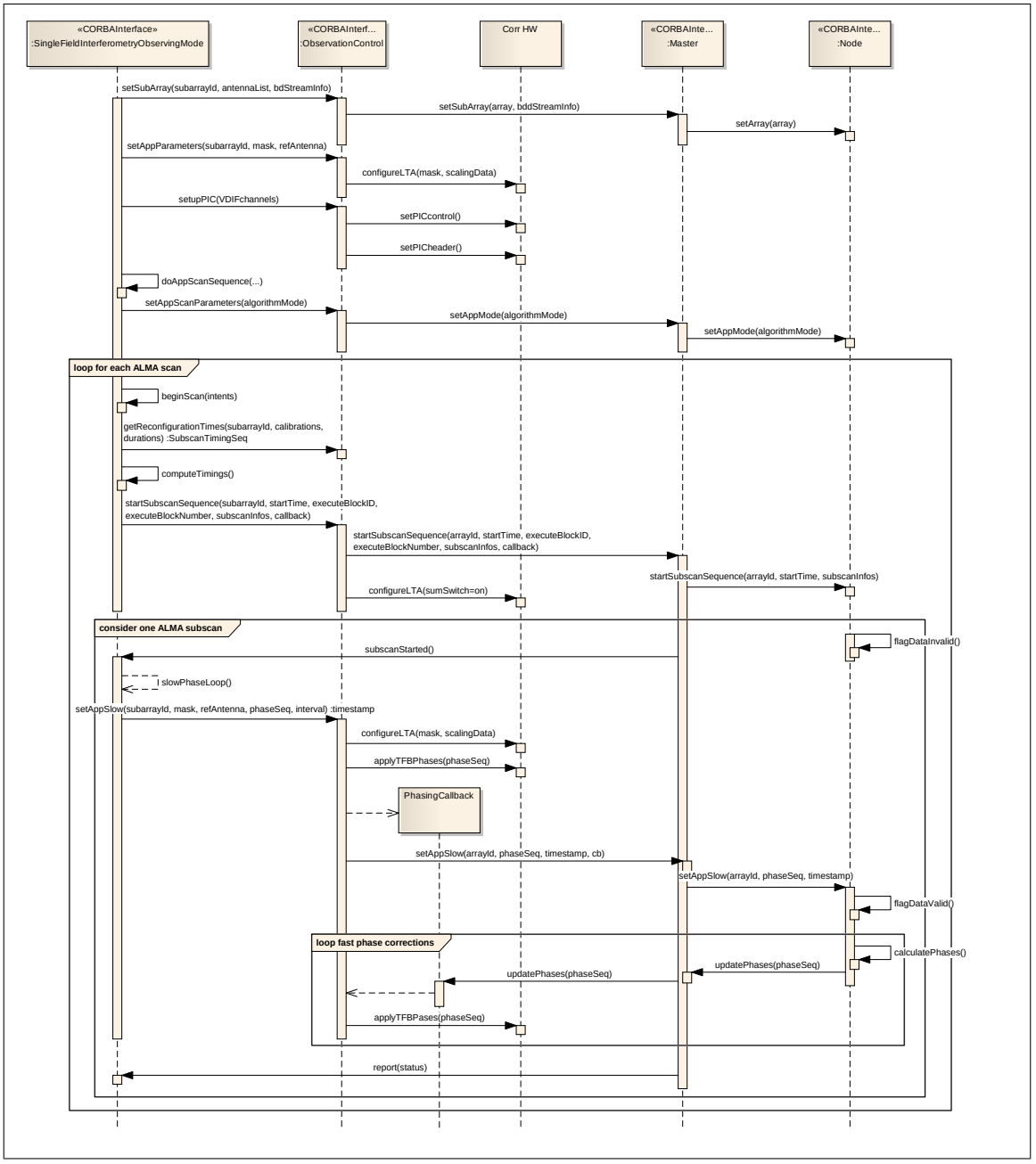


***Figure 5.5:*** *VLBI scan sequence diagram between the* ***VOM*** *and* ***Correlator****, representing the "fast" phasing loop. Fragments highlight ALMA scan and subscan boundaries, and the phase correction loop to be repeated within a subscan.*

### 5.4.4 Array Adjustment

Aside from the phase updates, the VOM is in a position to make automatic adjustments of the `phasedArray` composition, and it will also provide an interface so that the operator can do this on demand via the GUI. The results provided by TelCal include an overall figure-of-merit for the `phasedArray` based on the fact that the correlation of a comparison antenna with the phased-array sum should scale with $sqrt(N_{\text{phaseD}})$. Additionally, there will be information about individual antenna performance relative to the phased sum. The internal logic is thus:

```
PhaseResult result = getAppPhaseResult();
double efficiency = computeEfficiency(result);
Antenna refAnt;
AntennaSeq drops, adds;
ACS::Time last;             // time of last adjustment
if (efficiency < efficiencyThreshold)
    if (ok == getRefAntenna(result, &refAnt, &drops, &adds, last))
        adjustPhasedArray(phasedArray, refAnt, drops, adds, &last);
// getRefAntenna() checks last relative to adjustmentRate
// adjustPhasedArray() updates last
```

where the (normalized) `efficiencyThreshold` can be set to trigger the automatic adjustment at some level of inadequacy, the `drops` and `adds` are pairs of antennas that could be dropped or added (respectively) based on TelCal's assessment, and as the `refAnt` itself might be a poor performer, it may be updated as a side effect. The `efficiencyThreshold` may be lowered to zero to disable the automated capability. Additionally, the time of adjustment must be noted so that the rate of adjustments is not large. (Otherwise, the jitter in the array phase could become significant.) The `adjustPhasedArray` method shares the new array with both TelCal and the CCC. This transaction occurs during the "invalid-for-TelCal" front portion of the subscan, so there is no need for careful timing.

The `vpc.adjustPhasedArray` interface can be exposed to the operator GUI. This call provides the adjustment of the `phasedArray` for TelCal's next pass through the slow loop (and logs the transaction). However, the controller should arrange to use the call only during the "invalid-for-TelCal" portion of the subscan (*i.e.* hold the operator request until TelCal "slow" results arrive). Likewise, the GUI should probably arrange to either lower the `efficiencyThreshold` or the rate limiter `adjustmentRate` to prevent the operator efforts from conflicting with those of this VOM logic.

# Chapter 6

# Correlator Implementation Details

While the VOM is the higher level coordinator of a VLBI observation, the correlator software will expose some interfaces to enable access to the hardware level commands, in addition to the fast phasing loop implementation. Observing mode commanding of the correlator is normally done through high level commanding of the `ObservationControl` component (part of the CCC), which will then forward commands to the correlator hardware or the CDP Master. General status requests or queries are usually done through the `ObservationQuery` component (also part of the CCC).

This section describes the logic to be implemented in the correlator software. The detailed list of available CAN messages is defined in [RD6] and not repeated in detail here. As there are a number of new commands that will go to the CAN bus during an observation (most importantly TFB phase updates), a careful schedule of messages is to be considered to avoid concurrency issues and overloading the available bandwidth.

Section 6.1 describes updates to the LTA card protocol; Section 6.2 refers to the addition of the PIC cards; Section 6.3 describes updates to the TFB card controlling, as part of the phasing loop; and Section 6.5 defines the relevant status query interface. The CDP changes for the fast phasing calculations are discussed in Section 6.4. Finally Section 6.6 discusses improvements to the correlator software simulation infrastructure to support development and debugging efforts.

## 6.1 Correlator: LTA Protocols

The LTA protocols control the generation of the phased-sum and the provision of it to the correlator and the PICs.

There are 16 LTAs per quadrant; a given LTA controls the correlator cards (CCs) for two 62.5 MHz channels (two planes). Some of the CCs can see all 64 antennas; the baseline plan is to have a set of these provide the sum for the 32 channels (both polarizations) and send these to the pair of PICs installed in that same quadrant. (While there are physically enough CCs to compute a second sum, that would require a second set of PICs and cabling and is not in the current plan.)

The "sum" output is routed to CAI-63 via a software switch (`SET_CIC_SUM_INPUT_SWITCH`). Since the revised LTA firmware will have this internal switch (*i.e.* use CAI-63 or use the sum), the SFIOM should acquire a command to by default reset the LTA firmware to use the actual CAI-63.

The set of antennas to be phased-up and summed is defined for the LTA cards through an `AntennaMask` variable. It contains 64 bits of information: 1 bit for antennas used and 0 otherwise. (A list of used antenna names is equivalent.) This mask must be sent to the LTAs so that they can provide the appropriate sum as a replacement for CAI-63 (on all quadrants). Since the phasing system must not try to include its own product in the phased sum, antenna CAI-63 must never be included in the phasing sum.

The logic that calculates the sum of 2-bit values from $N$ antennas, will need to know where to place the thresholds in this 8-bit summed value so as to reduce it once more to 2 bits. These thresholds are only a function of $N$ (*i.e.* the number of 1's in the mask) and (in a minor way) the 2-bit statistics of the ALMA correlator. A lookup table for the required CAN protocol is provided in Appendix D.

Methods exposed to the `PhasingController` (see Section 5.4) to trigger LTA commanding include:

- `setAppParameters(subarrayId, mask, refAntenna, algorithmMode)`: Notifies general phasing parameters at the beginning of a VLBI scan. The antenna sum mask (to be translated from antenna names to CAN ids) and sum scaling data will be set in the LTAs at that time. The CCC will set an internal flag to turn the sum switch on during subscan configuration. Note that if this command is never called the CCC will always set the sum switch off by default. The algorithm mode enumeration will define whether "slow" or "fast" phasing mode is to be used.

- `setAppSlow(subarrayId, mask, refAntenna, phaseSeq, interval)`: Updates phasing parameters during an observation. Typically this command will be called once TelCal corrections are available in the `PhasingController`, some time after each subscan start (except the first one of every scan sequence). If changed, the new antenna mask and scaling data are updated in the LTAs. An application timestamp is returned to the observing mode for inclusion in the metadata.

## 6.2 Correlator: PIC Control

A new card type will be added to the baseline correlator, called Phasing Interface Card (PIC). There will be two PICs per quadrant, one for each polarization, and they operate independently. Their primary role is to format the sum data, received from the CCs, into VDIF format and transmit it through the optical fiber link to the recorders. These cards, as all other cards in the correlator, are controlled through CAN commands.

At the beginning of a VLBI session the VOM will command the CCC to set up the PIC cards. The involved actions are essentially:

```
PICStatus getPICstatus();       // get PIC status
PICError  setPICcontrol(...);   // adjust PIC state
PICError  setPICheader(...);    // define data format
```

which provides the complete state of the PIC. At the start of the observation, a PIC might be fully programmed, or powered-off (to save power). So:

```
// is it on and programmed?
getPICstatus();
//  yes: you're done
//  no: turn it on (several steps)
setPICcontrol();
```

where the power-on sequence sequentially manipulates control bits to take the PIC from "off" to "on" as described in [RD6].

Following power-on or reset, a PIC needs only a few details for the VDIF header in order to generate packets. The VDIF header contains some static information (*e.g.* quadrant/polarization), including timing information (used to align data in the eventual VLBI correlation). See [RD6] for more details on the VDIF packet. The following method:

```
// fully specify the VDIF header and start packet generation
setPICheader( ... timing, channels, ... );
```

supplies this information aligned on both a TE and PPS event. Because TEs are 48 ms apart, this happens once every 6 seconds, so programming all 8 PICs requires one minute (without parallel execution). As there is a dedicated CAN bus for every quadrant this setup can be done in parallel for every quadrant, but sequentially for each of the two cards per quadrant.

```
// does it have the right time and data format?
getPICstatus();
//  yes: you're done
```

```
//   no: give it the time and data format, repeat
setPICheader();
```

The `getPICstatus()` method receives the full VDIF header at the previous TE so the validity of the timing can be checked. Aside from the timing: normally 32 channels are to be output; however, a smaller power of 2 (*e.g.* 16, 8, 4, 2 or 1) may be useful in some situations.

The `GET_PIC_STATUS` CAN command also provides more extensive status and statistics. This is intended to be used periodically during observations. More on this subject is discussed in section 6.5.

Based on status information from the PICs and other parts of the system the VOM or the operator might decide to flag data as invalid in abnormal circumstances. The VDIF header contains a bit to flag VLBI data, which might be toggled during an observation. Overall system contingency conditions are discussed in section 9.

Methods exposed to the `VLBIController` (see Section 5.3) to trigger the above PIC commanding include:

- `setupPIC(VDIFchannels)`: Perform complete preparation for a VLBI scan on all PIC cards. This includes power up or reset, and apply of the new header. A structure (`VDIFchannels`) including information for each PIC's channels is passed as parameter, while the rest of the header data can be generated internally by the CCC (see Appendix E for details on the header structure). Note that as this command performs status verifications first, it may also be used to update the header during an observation.

- `setPICinvalid(bool)`: Sets the header flag to mark data as valid/invalid. Note that this command might be invoked by multiple independent clients, thus a verification is necessary to take an action only if the PICs are already set up (`setupPIC` was successfully executed), and to not command the same action twice; otherwise the command is to be omitted silently.

- `disablePIC()`: Turn PIC cards off. This is likely to be commanded manually at some point after the end of a VLBI session.

## 6.3 Correlator: TFB Protocols

The TFB is a component of the Correlator Station Racks (128 cards distributed in each Correlator quadrant). The commands which actually adjust the phases are issued by the correlator control computer (CCC) to the digital LOs of the TFB cards. Each quadrant has a dedicated CAN bus (so bandwidth is limited to 1 Mb/s to each quadrant), and these commands can proceed more or less independently. The only shared resource here is the CCC and the overall logic.

These TFB adjustments are delivered in two phases: a `DOWNLOAD_TFB_PHASES` command with the phases to place in temporary storage, and a subsequent command to apply them. Figure 6.1 shows the underlying logic of the TFB.

The download command takes about 220 ms to deliver (12 bits per phase for 2 polarizations on 64 antennas, delivered all at once), while the apply is a short, quick message (about 1 $\mu$s). The phase is actually represented as a 16-bit datum, but the upper bits represent phase wrap (*i.e.* multiples of 360 degrees). Currently the four quadrants are downloaded sequentially; and the command to apply the new phases requires still more time, for a current total of about 1.5 seconds. This will be parallelized to reduce latency in the phasing loop.

The current use of `DOWNLOAD_TFB_PHASES` is tied to the `DelayServer` adjustments and the present usage of the TFB logic. Specifically, the apply logic clears the phase register (*i.e.* uses the `CLEAR ENABLE` signal) and places into the phase offset register a value which is what the LO would have had if no action had been taken. (This is all to support "return to phase" after making LO changes.) For the phasing system, there is no need to change the LO frequency, so it is sufficient for the CCC to remember the last value it placed in the phase offset register and merely adjust it with the incremental values from the phasing system. Thus as each phase update is received (from TelCal or CDP), something like this:

```
// *phase_offset_register* refers to m_tfbPhases[cai][tfb][filter]
```

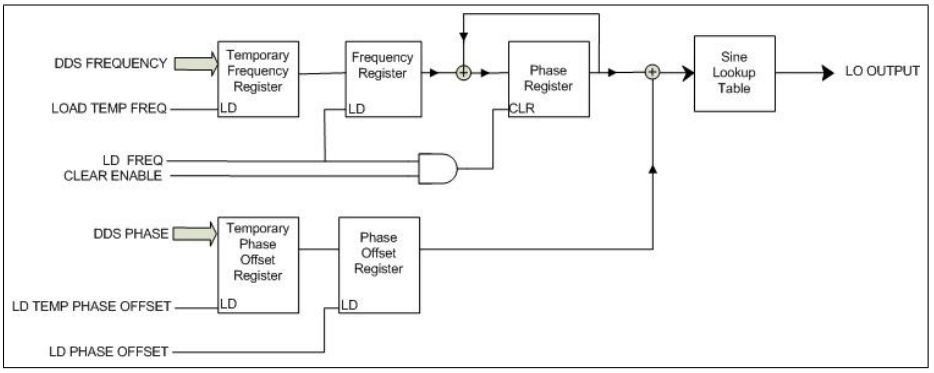


***Figure 6.1:*** *Internal TFB logic. The frequency and phase offset are downloaded to temporary registers, and brought into use by deliberate load commands at precisely determined times. Each clock cycle produces a phase-based lookup into the sine value for the LO output.*

```
// as used in SCC_DownloadTFBPhases
phase_offset_register = previous_phase_offset_register_value;
phase_offset_register += phase_update_adjustment();
```

to populate the `DOWNLOAD_TFB_PHASES` CAN bus message, followed by the new `APPLY_TFB_PHASES` command (which merely moves the new value from the temporary register to the active phase offset register) is all that is required.

Note that the `DOWNLOAD_TFB_PHASES` message goes to a processor which is fully capable of unpacking a shorter message; so that there is a considerably reduction in the 220 ms required to download the phases. *E.g.* if the phasing loop is only delivering one phase per antenna, the 32-fold replication of the same phase to each TFB could be eliminated, allowing a 7 ms transfer time.

As a second comment, the logic to make this adjustment planned for the CCC could also be placed in this processor. (*I.e.* this processor could remember the last phase offset register values applied and merely increment it by new values.)

During the phasing loop TFB phase updates can be triggered by two different means (see figure 5.5):

- The `PhasingController` will issue a `setAppSlow` command (same as described in Section 6.1) at some point after the start of every subscan, containing the latest corrections from TelCal. The CCC will apply these corrections to the TFBs and optionally forward them to the CDP (if in "fast" mode).

- If the "fast" loop if active the CDP will produce additional phase corrections in between TelCal updates. These are returned to the CCC through the `updatePhases` command (or event) and will be applied on a *best effort* schedule to the TFBs. This means that corrections are applied as soon as they arrive, but most other commands (similar to delay corrections from the `DelayServer`) will have higher priority and could cause one round of phase updates to be skipped.

Note that both commands contain a time interval parameter, during which the corrections are valid. Corrections are therefore applied as soon as possible, within that interval. If received after that interval or it was not possible to schedule the CAN command within the interval the corrections are to be discarded and a warning log produced.

## 6.4 Correlator: CDP Phase Updates

Right after defining an array, the CCC will pass on the phase correction algorithm option to the CDP Master (see Figure 5.5), so it knows that this is a VLBI array, and if any fast phase calculations are to be performed.

Subscans will be executed as usual, but with the following modifications:

- Disable all usual WVR corrections. Optionally, also disable residual delay corrections. (Note that the option of enabling/disabling these corrections applies to all quadrants simultaneously.)
- Nodes will flag the data as invalid at the beginning of every subscan (but for the first one), until a `setAppSlow` command with a timestamp is received. Data from that timestamp on is to be unflagged, as it contains the new TelCal phase corrections.
- If a fast phase correction algorithm is to be used, the relevant TelCal data for this is received in the same `setAppSlow` command. Fast incremental corrections are then calculated by the Nodes, gathered by the Master and sent back to the CCC for application in the hardware. These corrections are to be calculated in regular intervals (about once a second) until a new `setAppSlow` command updates the reference information, or the observation finishes (note that this is not limited or interrupted by subscan boundaries, thus corrections should continue being calculated and applied even between subscans).

Note that the CDP can implement different phase correction algorithms, which might be selected through an enumeration parameter. The current scope is described in Section 5.4.1; the simplest case is to calculate WVR corrections, based on online measurements coming from all antennas. A WVR correction algorithm is already implemented in the CDP and currently being commissioned. This algorithm makes the correction on a baseline (between two antennas) basis. For the APP a per-antenna correction (implemented via the TFB commands) is required. The WVR correction is a delay-like correction at each antenna which is equivalent to a linear slope of phases at each spectral point. For the APP, we are interested in the phases at the center of each TFB. And rather than the phase difference between one antenna and another (*i.e.* its baseband peer), we either just want the absolute phase correction itself, or the correction relative to the reference antenna. The former transfers the errors in the WVR at the reference antenna to the phased-sum, but does remove some of the WVR delay; the latter leaves the WVR delay of the reference antenna imprinted on the phase-sum that is recorded. Which choice is optimal for VLBI has yet to be determined (if indeed one choice is optimal under all circumstances).

Finally, we note that the interfaces created are sufficiently general that other algorithms may be implemented in the CDP computers with at most minimal modifications.

## 6.5 Correlator: Status Query Interface

As mentioned before, correlator status queries from other parts of the system are usually done through the `ObservationQuery` interface. Figure 6.2 shows this kind of interactions from other subsystems. The individual cases are defined as follows:

- `getPICstatus(picIds)` : A synchronous call, periodically invoked by the VOM (`VLBIController`). It returns a status structure for the requested PIC(s). This translates into the `GET_PIC_STATUS` hardware command defined in [RD6]. This command is expected to be issued a few times per VLBI scan, *i.e.* on a minutes time scale. At some point the VOM could decide to invalidate the PIC data due to abnormal conditions by commanding `setPICinvalid(true)`.
- `requestPhaseUpdates(PhaseUpdateRequestStruct)` : An asynchronous call containing a request structure specifying phase updates to be published (*e.g.* subset of antennas, quadrants, polarizations, TFB channels, *&c.* ). Right after applying TFB phase updates (see Section 6.3) the CCC will publish the requested values as a `PhaseUpdateEvent` to a Notification Channel. A new request should override the previous one, and requests are cleaned up at array destruction. A phase updates request can be issued by any client interested in receiving updates through the Notification Channel. The main user of this feature would be the "CCC Phase Update GUI", described in Section 8.4. Note that the same logic is used by the `CorrGUI` to subscribe to online updates of relevant CDP processed data.

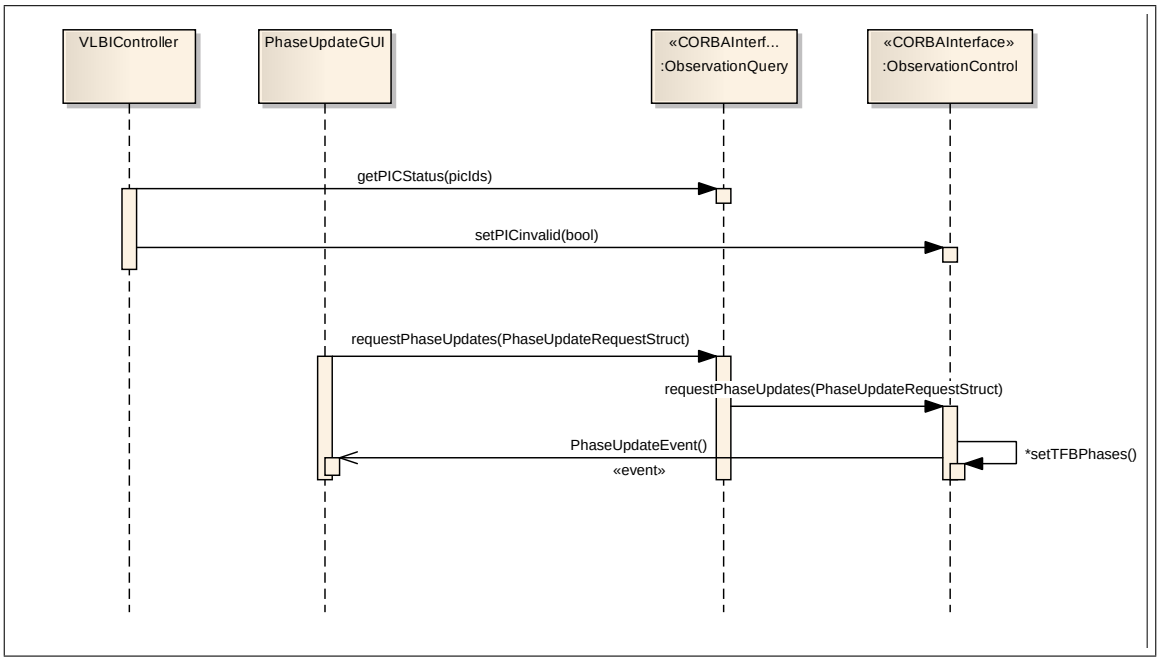


***Figure 6.2:*** *`ObservationQuery` status requests diagram. Each message group represents one possible (independent) request process. Clients include the VOM and the Phase Update GUI.*

## 6.6 Correlator: Simulation Improvements

The APP hardware is expected to be temporarily available in a testing setup. However, testing of all capabilities will only be possible with the full deployment on the baseline correlator. Testing time there is expected to be scarce, both during development/testing and later during maintenance. Simulation capabilities to enable software-only testing is therefore a critical part of the system which needs to be considered from the beginning.

While control subsystem simulation is part of the device generation framework (see Section 7), correlator simulation requires manual additions of capabilities. In principle it is divided into a CAN bus simulator (channeling correlator cards CAN commands to simulation classes) for the CCC portion, and a lag generator (replacing lags normally read out from the correlator hardware output) for the CDP. Both the control and correlator simulations are implemented at the hardware communication level (also eliminating real-time requirements) and therefore transparent for all higher parts of the system.

The correlator simulator is undergoing some improvements (outside of the APP scope) that should facilitate the addition of support for new CCC to hardware commands (LTA, PIC and TFB protocols). Each CAN command is forwarded to a simulation class that performs some customizable logic and return an expected output to the usual CCC software.

On the other hand, the CDP node input (lags and leads) can be either generated as a flat spectra or read from files that contain dumps from real observations (corresponding to the used correlator configuration). However, CCC commands are not fed into the CDP simulation.

The main new simulation feature needed for APP is thus the closure of the phasing loop, *i.e.* applying TFB phase corrections to the simulated correlator data that is sent back to TelCal. TFB phase corrections in the corresponding CCC simulation class can be sent as a `SimPhaseEvent` to a new `SimPhaseProvider` class in the CDP nodes of the same quadrant (identified by an event parameter). Here the corrections would be applied on the spectral data as `[cf, delta]` pairs (TFB central frequency; phase delta). The `SimPhaseProvider` would be implemented following the same idea of current "provider" classes in the `CDPNode` code that "provide" delay and WVR corrections received as events from the CCC.

Therefore, a full integration test scenario is to use this advanced correlator simulation to actually

apply corrections coming from TelCal. The channel average data sent back to TelCal would then contain those corrections, and we should observe a progressive evolution of phase updates. The resulting spectral data should be suitable to be analyzed in CASA as well, although the usefulness of that step would depend on the quality of the initial lags simulation.

# Chapter 7

# Device Implementation Details

With the exception of the PICs, none of the APP hardware will be on the CAN bus; rather they are accessed via Ethernet interfaces. The Back End Central Variable Reference (CVR) device is a reasonable prototype that can be used as a model for implementing the maser, link and recorder devices. The CVR is an Agilent synthesizer (*e.g.* an E8257D) that accepts SCPI commands on that interface. The CVR Control device code implementation uses the Control code generation framework's extension for Ethernet devices, which also provides a corresponding device simulation (by redirecting hardware commands to simulation classes), and a CCL python library that may be used for direct interaction at the command line or via script. Some higher level commands (usually starting with lowercase) may be added for encapsulation. Also, the existing ALMA software infrastructure allows to configure periodic monitoring and storing of generated device monitor points, which can be retrieved by the user.

The three APP devices are sufficiently similar that common code will be used for them (either borrowed from the CVR or copied from one to the other). Note that the CVR opens and retains a socket throughout the device life-time. This is not required for any of the APP devices, and the alternative (connecting the socket as needed) needs to be implemented.

## 7.1 Device: Hydrogen Maser

The Hydrogen Maser is a hardware replacement for the existing Rubidium clock. There are no operational commands to the device. Monitoring consists of periodic access to its monitor points (Section 7.1.1) and monitoring of its drift (Section 7.1.2). Full details on the Maser are to be found in its user guide [RD7]. The software device name will be `Maser`.

### 7.1.1 Maser Monitoring

The T4Science iMaser (currently at Haystack) housekeeping can be read from any linux machine via an http request, *e.g.*

```
printf "GET /monit.htm HTTP/1.0\r\n\r\n" | nc 192.52.61.160 80
# with some parsing:
NDCU815 iM59  01102012 212425   MONITORING RECORD ON 600 sec
 U batt.A[V]   27.661  EB heater[V]    12.358   Pirani heat.[V] 12.219
 I batt.A[A]    0.8525 I heater[V]      6.562   Unused           0
 U batt.B[V]   27.905  T heater[V]      8.223   U 405kHz[V]     11.847
 I batt.B[A]    2.533  Boxes temp[C]   46.704   U OCXO[V]        4.087
 Set H[V]       5.387  I boxes[A]       0.586   24 VDC[V]       24.71
 Meas. H[V]     1.254  Amb.temp.[c]    23.975   15 VDC[V]       14.53
 I pur.[A]      0.459  C field[V]       5.09    15 VDC[V]       15.23
 I diss.[A]     0.46   U varactor[V]    5.576   5 VDC[V]         5.08
 H light[V]     3.024  U HT ext.[kV]    3.467   5 VDC[V]         0
 IT heater[V]   7.275  I HT ext[uA]     6.47    8 VDC[V]         8.01
```

```
IB heater[V]    6.235  U HT int.[kV]     3.501   18 VDC[V]        17.19
IS heater[V]    8.203  I HT int.[uA]     6.714   Unused            0
UTC heater[V] 13.061  H st.pres.[bar]   1.128   Lock              1
ES heater[V]   11.411  H st. heat[V]   14.478   DDS     1420405750.302645
```

The HTML output cannot be parsed with, *e.g.* the `expat` library as the page formatting has bugs. Fortunately, it is not so broken that it cannot be reliably parsed by merely locating the `<td>` and grabbing the key and value pairs. There are defined alarm limits for all of these. The data updates every 10 minutes presently, so monitoring more frequently than that is pointless.

### 7.1.2 Maser Drift

For monitoring the (existing) rubidium clock, there is a counter that counts (125 MHz) clock ticks between each GPS tick. This data is available from the archive (`MASER_VS_GPS_COUNTER`). A sample of data from 4 days in April of 2011 was obtained (at a time when the rubidium clock was the source for the 125 MHz signal and the rubidium clock was *not* locked to GPS). There were 25 system resets in this interval; Figure 7.1 shows the results of processing this data into the traditional VLBI "clock_early" datum (with an artificial offset of 0.0001 s added at each reset to separate the lines which would otherwise overlay in the figure). An online version (good for sanity

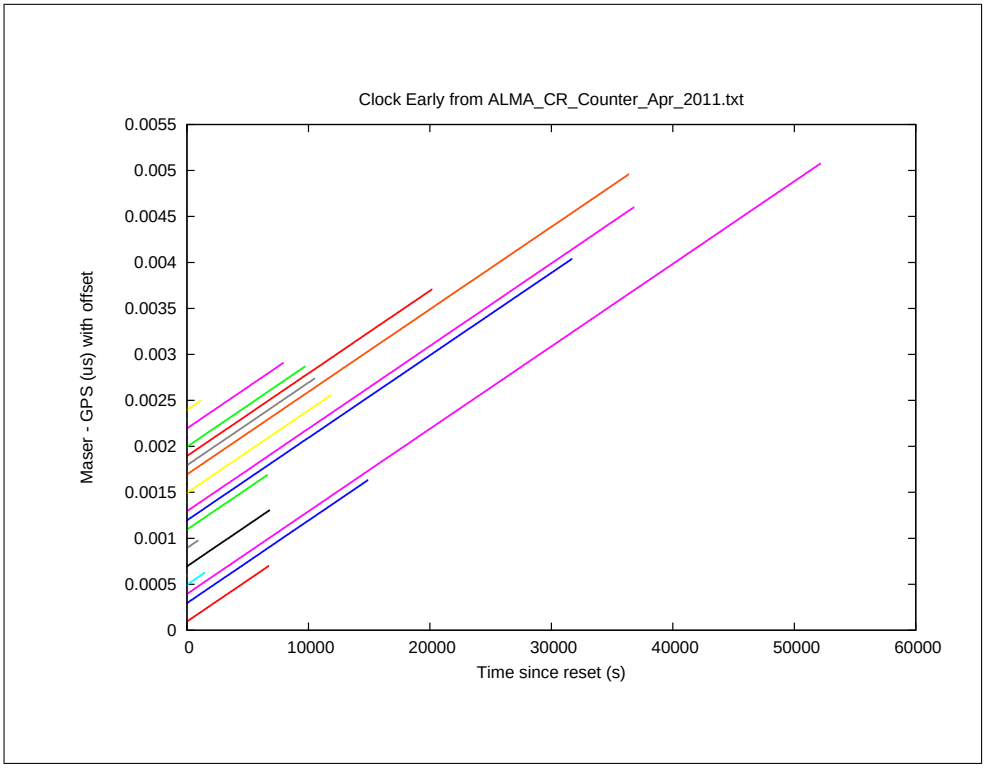


***Figure 7.1:*** *Sample Maser–GPS drift data, see text.*

checking) could be implemented. The ratio of the difference between the newest $N_{new}$ and oldest $N_{old}$ values of `MASER_COUNTER_RESET` with the difference of the corresponding ACS times when these values were acquired is a good estimate of the clock drift:

$$\text{clock\_early} = 8000 \left( \frac{N_{new} - N_{old}}{t_{new} - t_{old}} - 1 \right) \text{ps s}^{-1}$$

## 7.2 Device: Optical Fiber Link System

The Optical Fiber Link System (OFLS) consists of two identical boxes (multiplexors), one at the AOS and one at the OSF. These network devices combine the signals from the 8 PICs onto a optical single fiber to provide a 64 Gbps data link to the VLBI recorders. There is one spare port. Each supports monitoring of its housekeeping, and an operational reset command. Other commands (to *e.g.* change firmware) are described in the user manual [RD8] (current draft). The software device name will be `VLBIOFLS`.

The optical fiber link system provides a simple socket-based Ethernet connection for commands and housekeeping, *e.g.* via telnet to port `5653`. The monitoring command set consists of:

**show_alarm_hex** Show alarms in the system (4 byte response).

**show_alarm_str** Show alarms in the system (verbal response).

**show_system** Show system information (*e.g.* firmware version).

**show_status** Show status of optical modules.

The `show_alarm` provides alarms on the following:

**temperature** OFL inside temperature (which must be less than 70C)

**fan** Cooling fan status (three units 1..3)

**converter** 12V AC/DC converter (two units 1..2)

**sr_port** Status of XFP-10GBASE-SR (local) module (for each of units 1..9: link, signal loss, *&c.* )

**zr_port** Status of XFP-10GBASE-ZR (remote) module (for each of units 1..9: link, signal loss, *&c.* )

**firmware** Firmware status

The commanding set consists of:

**reset** Restart the system.

**set_alarmmask** Indicates which port(s) are spare.

**apl_download_start** Reload the firmware.

The `reset` and `set_alarmmask` capabilities will be supported by the device, but will not be used by the VOM. Reloading the firmware is probably better accomplished via a procedure that bypasses this software device.

## 7.3 Device: VLBI Recorder

There will be four Mark6 16Gbps recorders, each recording the data from the pair of PICs (one baseband, orthogonal polarizations) in one quadrant. Prior to each VLBI session (a night of observing) there will be some offline preparations (described in Section 7.3.2). During the observation the scans will be recorded as described in Section 7.3.3. Following each VLBI scan, a "scan check" operation is typically performed (Section 7.3.4). Finally, for general monitoring of the recorders, the recorders support a general status capability (Section 7.3.5). All activities are carried out through the VLBI Standard Software Interface Specification (VSI-S) summarized briefly in the next section (Section 7.3.1); see the Mark6 user guide [RD9] for full documentation. The software device name will be `VLBIRecorder`.

### 7.3.1 VSI-S Interface

The Mark6 has a control plane that listens on an Ethernet socket (port 14242) for VSI-S commands. An internal data plane is then instructed on what network streams to record to which modules. The machine is configured so that a reboot will take it back to the nominal state with this control plane running.

Commands/queries/responses (respectively) are of the form

```
<keyword> = <field 1> : <field 2> : ... ;
<keyword> ? <field 1> : <field 2> : ... ;
!<keyword> = <VSI return code> : <Mk6-specific return code> : ... ;
```

so methods to exchange command–response (`=`) and query–response (`?`) atoms with the recorder over the socket form the basis of the device. Thus:

```
// send a command
VLBIRecorder.vsi_command( keyword, ... );
// send a request for information
VLBIRecorder.vsi_query( keyword, ... );
// retrieve the response
VLBIRecorder.vsi_response( keyword, ... );
```

will be implemented with appropriate checking for invalid or unsupported keywords. Higher level functions can then be added to facilitate common operations (*i.e.* to keep the details in the device and use appropriate abstractions in the VOM); these are discussed below.

### 7.3.2 Module Preparation

There are a few commands that must be executed manually by an operator when new disk modules are inserted or removed—a procedure will cover this, and the commands can be executed from the GUI as supported by this device. We estimate VLBI sessions will use four 16 TB modules per recorder per observing night, with an exchange of modules between observing nights. (With time, the number of modules will decrease with increasing disk capacity.)

The disks are packaged in modules which are individually packaged in shipping cases with approximate dimensions and weight of 19 in x 13 in x 11 in (48 cm x 33 cm x 28 cm) and 23 lbs (11 kg). These disk modules typically are conditioned before use, and (full) conditioning might take longer than an observing night. So there are effectively two sessions: a conditioning session that runs through all the modules in order, and the actual observing session.

The modules will have been shipped from the VLBI correlator with enough lead time to be ready for the observation. Usually extra modules are provided to allow for a small fraction that fail on-site testing. The operators should be provided with a detailed procedure and a per-session schedule for testing and using the modules. For a typical session:

1. unpack the modules from the shipping boxes,
2. install the modules in the Mark6 recorders,
3. run the test script on each Mark6 (go away for a day)
4. repack the modules: retain passed modules for the session, put failed modules aside
5. repeat steps 1–4 until enough modules for the session have passed.
6. wait for the day preceding the VLBI session
7. unpack the modules from the shipping boxes,
8. install the modules in the Mark6 recorders,
9. (run the observation)
10. repack the modules: mark these modules as “used”.

11. repeat steps 7–10 until the session ends.

Each of these tasks takes a few minutes; so 1–2 FTE hours per observing night are required, plus shipping costs and manpower. The latter consists of staging the module boxes near the Mark6 recorders for the duration of the VLBI session and then transporting the completed modules off-site at the end. Aside from the module labor, running a VLBI observation should be similar to monitoring the existing SFIOM Mode and should incur no extra FTE effort. For reference, the module costs are approximately $65 for the shipping boxes, $500 for the module without disks, and currently 8 2TB disks to fill it cost $900. The software device must support these activities via a simple Python script using the CCL interface. Again, the conditioning is conducted offline, outside of the VOM session.

### 7.3.3 Recording Schedule

The `VEX2VOM` tool (Section 4.2) will summarize the `SCHED` section of the VEX file describing the observation and generate an XML object which describes the schedule of recording for a particular session. The Mark6 has a command (`execute`) to allow the upload of this XML object. Thereafter the Mark6 autonomously records the scan. The XML looks like this:

```
<experiment name=xray21, station=A1, start=2012081050600, end=2012081170000>
    <config ... />
...
    <scan experiment="xray21", source="M87", station_code="A1",
        start_time="2012081060300", duration="600",
        scan_name="xray21_A1_081-0603"/>
    <scan experiment="xray21", source="3C273", station_code="A1",
        start_time="2012081061700", duration="240",
        scan_name="xray21_A1_081-0617"/>
...
    <scan experiment="xray21", source="1921-293", station_code="A1",
        start_time="2012081152700", duration="240",
        scan_name="xray21_A1_081-1527"/>
    <scan experiment="xray21", source="SGRA", station_code="A1",
        start_time="2012081153500", duration="720",
        scan_name="xray21_A1_081-1535"/>
...
</experiment>
```

A schedule in progress may be aborted with a subsequent (`execute`) command (inserting an empty schedule). Finally, making a single scan recording is also useful. Thus:

```
// deliver the schedule
VLBIRecorder.loadSchedule( ... xml ... );
// abort the schedule
VLBIRecorder.abortSchedule( );
// record one scan
VLBIRecorder.recordScan( name, start, duration );
// abort recording
VLBIRecorder.abortScan( );
```

are high-level scheduling methods to be implemented through the lower-level VSI-S interface.

### 7.3.4 Scan Check

Following a VLBI scan, the Mark6 can provide a report of the number of bytes actually recorded, scan name, *&c.* (also shown in Figure 5.2).

```
// scan check
ScanCheck VLBIRecorder.scanCheck( );
```

### 7.3.5 Recorder Monitoring

The recorders are Linux machines; routine monitoring of their internal state is advisable to avoid problems at VLBI session time. The precise details of the status information are not final, but presumably include:

- network byte received per interface
- disk usage
- system housekeeping

and can be accessed by the VOM or GUI via:

```
// scan check
RecorderStatus VLBIRecorder.getStatus( );
```

The actual status call is envisioned to re-use the `execute` VSI-S command, but with a directive to use a canned `status.sh` script which queries current (Linux) system state, parses it and returns the result in a format easily parsed by the software device. It will also make use of the VSI-S listed below.

```
disk_info?      Get information about individual disks in a module
msg?            Get ASCII message associated with return code
mstat?          Get module status
rtime?          Get remaining record time on open group
scan_check?     Quick check of data in recorded scan
scan_info?      Get summary information for recorded scan
status?         Get detailed Mark 6 system status
sys_info?       Get Mark 6 configuration details
```

# Chapter 8

# Graphical User Interfaces

The operator will have new GUI panels for the various parts of the VLBI phasing system, viz covering the timing, VLBI and phasing domains. These will provide status on the progress of the observations and allow some limited commanding capability. The commanding capabilities are either those needed for development which can be left available for the operator, or recorder commands needed when VLBI modules are exchanged.

Since the VOM will be monitoring the successful completion of every VLBI scan, there is no need for it to be explicitly monitoring the maser or the optical fiber link system. If either one fails, there is not much for the VOM to do about it except to abort the observation. The housekeeping from both devices is available to the operator to diagnose and eventually correct the situation.

The panels are designed to take advantage of existing infrastructure, and in some cases be added to existing commonly used operator GUIs. Additional capabilities beyond those described here might be developed to support testing or development. The panels are described in more detail in the following sections.

## 8.1 VLBI Observation Status

The needed information can be added to (or as an extension of) the existing Array Status Panel, and be only visible/active when in VLBI observing mode. A sketch is shown in Figure 8.1.

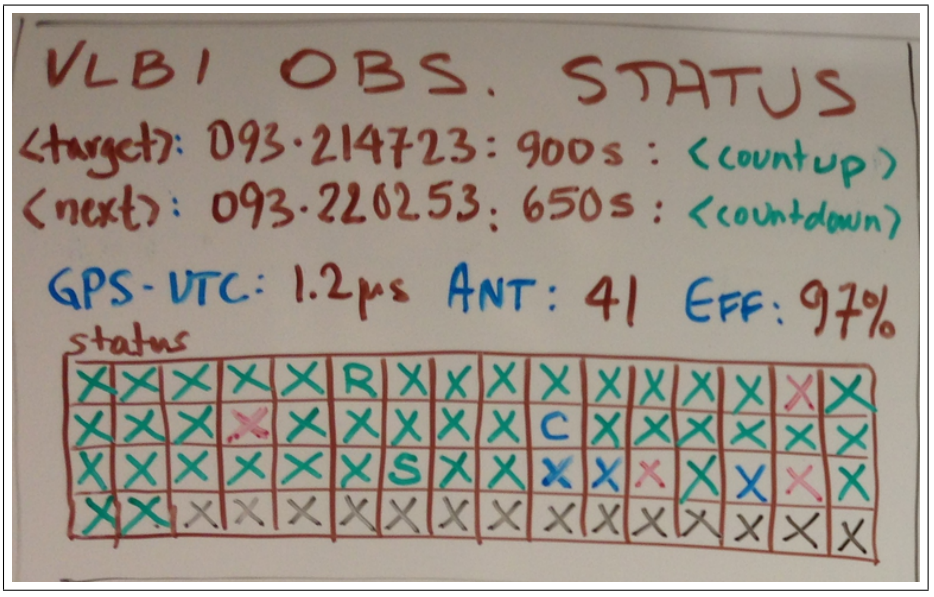


***Figure 8.1:*** *VLBI Observation Status. This sketch shows the top-level observing mode display.*

It shows the current scan (target name, duration) and feedback during the scan—a countdown until or an elapsed counter during. Similar information for the next (VLBI) scan is also shown. In addition the panel shows some overall figures-of-merit. (The GPS–UTC time offset provided by the PICs, the number of antennas, and the phasing efficiency of the array are the most essential ones). It also shows the disposition of the observing mode `subArray` antennas: those included in the `phasedArray`, those included in the `compArray`, or those not part of the `subArray`. In addition the `refAntenna`, the `cai63Antenna` and the `compAntenna` would be indicated.

Graphical methods to drop/add antennas will be available. For example, clicking on the status box will provide a pop-up with appropriate actions for that antenna. The VLBI scan check status (Section 8.2) could also be made available at this level—*e.g.* a green light if all is ok, or a red one to alert the operator to view more detailed information on the other display. Similarly, the scan displays could have a pop-up to allow the operator to skip a scan (or conversely, re-allow a scan that was disabled).

## 8.2 VLBI Hardware Status Panel

This is implemented as an `OMC` plugin showing some of the VLBI device components' monitor points, similar to the current Antenna Status Panel. A sketch is shown in Figure 8.2.

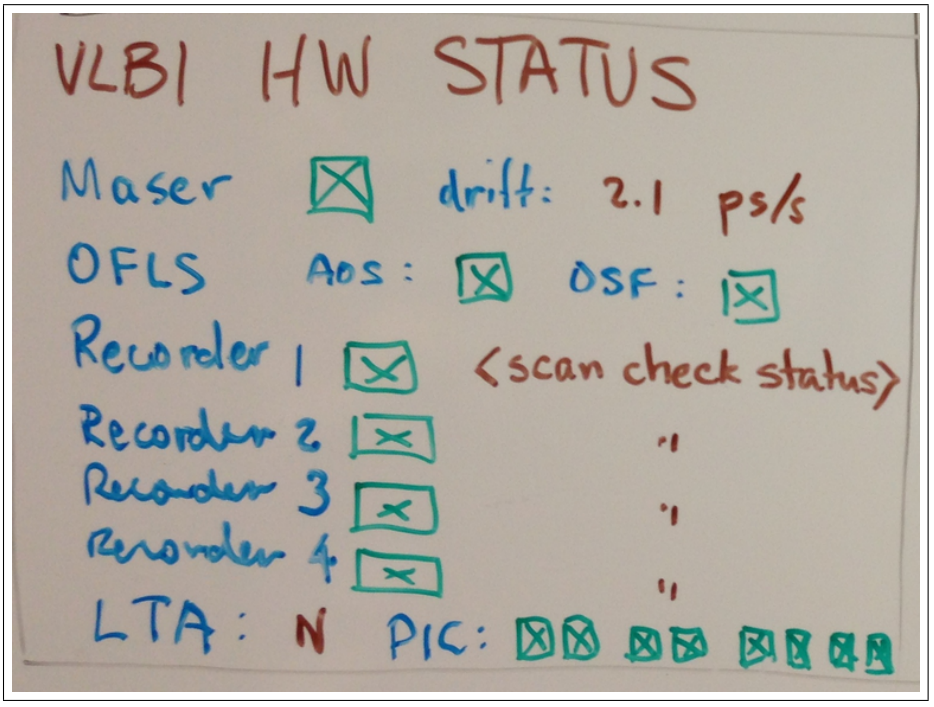


***Figure 8.2:*** *VLBI Hardware Status. This sketch shows the at-a-glance status of the hardward components used by the VOM.*

In all cases, an overall green-or-red type status indicator should be provide based on underlying data. For the maser, we are most interested to see the current drift rate (relative to the GPS system). For the OFLS basic link status is sufficient (but as there are 9 links, there could be 9× as many indicators). For the recorders, the scan check results of the last scan would be useful. The number of antennas supplied to the LTA is useful. For the 8 PICs general status is adequate. Detailed pop-ups for any of the devices might be useful.

For the recorders, a VSI-S commanding pop-up could be provided; although the operator could command from a terminal. Likewise, if the PICs have somehow lost the time, a means to reset them could be provided.

There is a per-PIC validity bit that can be set/cleared for the VDIF data that is produced. Marking this data invalid means that the recorded data will be discarded at VLBI correlation time.

Providing the operator with the capability to set this manually on a per-PIC basis is reasonable, but it should not be easy to access.

## 8.3 TelCal Results Display

This panel would be integrated with the current QuickLook GUI; it uses the results published by TelCal in the tables described in Appendix A. A sketch is shown in Figure 8.3.

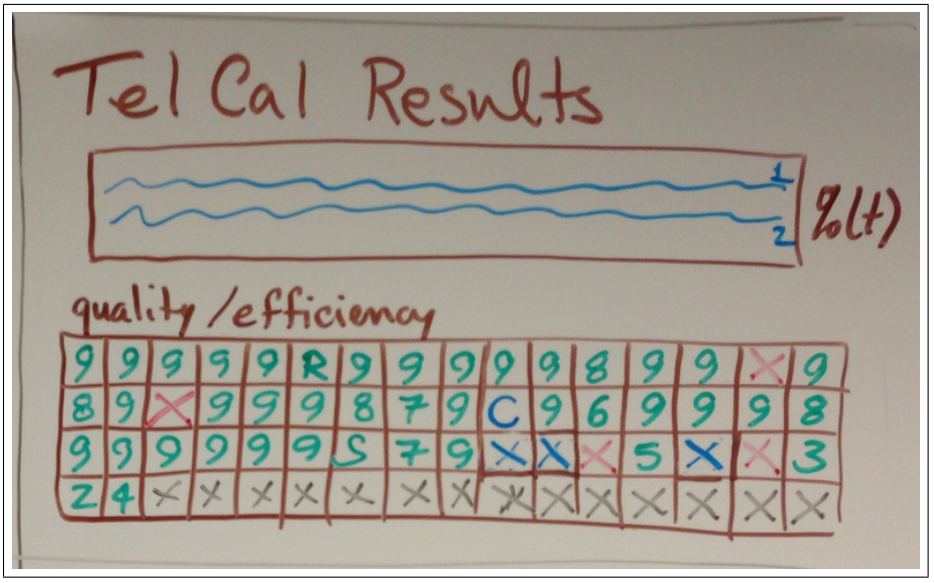


***Figure 8.3:*** *TelCal Results Display. A sketch shows quality/efficiency on a per-antenna basis, together with some capability to provide a time history.*

The main data provided there for this display are antenna quality and phasing efficiency. Having the ability to track some of these with time is useful. This sketch shows numerical summaries for each antenna and presumes an ability to select which data from which antenna is displayed, some control over time, ability to screen-shot or otherwise capture what is seen for future reference or discussion, *&c.* The details of the implementation would depend on screen real-estate. Note that the amount of data is driven by the channel-average configuration, (*e.g.* number of channels and frequency ranges) so most likely that would be presented as well, and the wealth of data from which to select for viewability would need to be managed.

Note that the slow phase updates are passed to the CCC so those could optionally be presented here, or alternatively available in the CCC phase updates panel (Section 8.4).

## 8.4 CCC Phase Updates

This panel makes use of a `CorrGUI`-like request/publishing mechanism (described in Section 6.5) and might be implemented as an additional panel to be started from within the `CorrGUI`. A sketch is presented in Figure 8.4.

As the phase updates (between the slow and fast loops) provide a wealth of data, it is not possible to view all of it simultaneously. However, the operator should be able to identify which data are of interest (quadrant, polarization, TFB channels; all, or only slow or fast) for specific antennas for time-history plotting.

Again, reasonable options to adjust the time-period of the display or the axis scales, hardcopy, *&c.* are not shown at this time.

The sample shown here presumes that one might want to track several channels from one quadrant. However depending on the underlying Java implementation, it might be as easy to pull

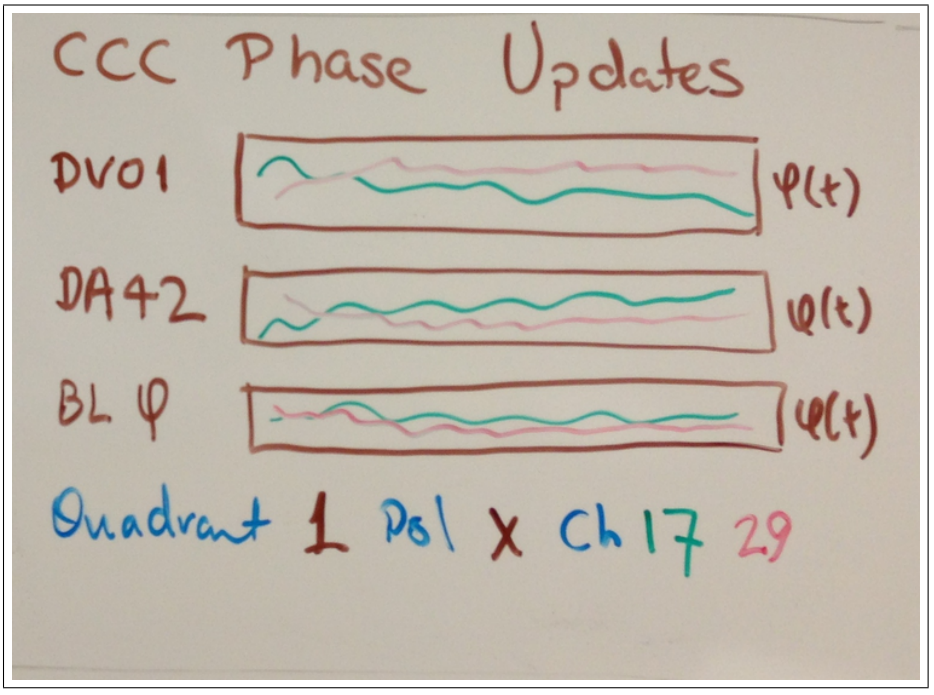


***Figure 8.4:*** *CCC Phase Updates. This panel shows time-history of phase adjustments of selected TFB channels.*

in similar channels from different quadrants. This would be most useful during testing rather than during routine observations.

# Chapter 9

# Contingencies and Fault Tolerance

As long as the VOM does not prevent existing SFIOM from running, it cannot introduce new failure modes for ALMA. That is, it cannot break any hardware and it cannot by itself cause other software to fail. In general the policy during VLBI observations is to continue observing unless an unrecoverable problem emerges. Under special circumstances the operator might decide to abort a VLBI scan, or only skip it at the recorder level (to avoid wasting disk space). The entire session might also be aborted, depending also on the observatory policy. Note that an observation can always be resumed (the script or SB executed again) within the absolute time schedule (the observing mode will skip scans that were scheduled in the past).

A given VLBI session will of course fail if, *e.g.,* the recorders fail to record data or if the phasing system fails to phase-up the array. Such failures are graceful insofar as they lose observing time only, and there is no recourse but to seek operator intervention. (*I.e.* there is little that the VOM by itself can do about them.)

Issues with any of the antennas (from any source) can be solved by dropping the affected antenna from the phased array. Since the number of antennas must be odd, a second antenna must also be discarded (or an idle one re-added from the `comparisonArray` antennas). This decision can be either taken internally by the VOM or overridden by the operator.

Issues with the maser must be addressed prior to starting a VLBI session. If the maser fails, a fall-back to the rubidium clock is a solution for ALMA, but there is no recourse for VLBI. On the other hand, masers of this type are typically used for decades, so this is not a likely problem.

Issues with the optical fiber link system should be rare, and will require operator intervention (reset or power-cycling).

Issues with the phasing system (*e.g.* poor efficiency) will have to be addressed during the commissioning period. The phasing system would have to perform rather poorly indeed, for the ALMA data to be not worth recording.

Issues with isolated scans will result in useless data recorded, but this is not a problem (as long as the issue does not persist through the remainder of the session). The PIC has a command for marking the associated VDIF data "invalid" so that it is cleanly removed from the VLBI correlation. This flag can be (un)set by the operator through a command line or GUI.

During regular ALMA observations, data is flagged or blanked in the CDP based on conditions spelled out in `http://almasw.hq.eso.org/almasw/bin/view/CORR/FlaggingAndBlanking`. Post-processing of the ALMA data to identify suspect times should be done as part of the "station logs" creation process. During VLBI sessions, ALMA data flagging and blanking will work as normal, to be considered by TelCal.

We briefly consider the following conditions that might affect VLBI scans:

**flagging: sigma out of range** this will distort the statistics of the sum, but it is not a serious issue—even at the extreme, the antenna is still producing one-bit data with a useful contribution to the sum

**flagging: TFB scaling factor** at worst, one TFB channel at one antenna will be invalid. This will corrupt one VDIF channel, but the other 31 will still be valid.

**flagging: missed lag zero** The ALMA data will be compromised, but the VDIF should be fine.

**flagging: TE not in hard mode** This will compromise the data from one or two PICs. The data may still be analyzable, so it should not be marked invalid.

**flagging: integration fully blanked** If this means the correlator is not seeing useful data, this might warrant setting the VDIF invalidity bit.

**blanking: missing WVR event** If the WVR data at one antenna is not available and WVR-based adjustments are desired, that one antenna should probably be dropped from the `phasedArray`.

**blanking: explicit blanking event** The affected antenna should probably be removed from the `phasedArray`.

**blanking: missing delay event** The affected antenna should probably be removed from the `phasedArray`.

The ACS logging system will be used to log error and emergency situations, while some info level messages are required for the VLBI station logs. The latter ones could be logged with a special APP audience if they are not interesting for other system users.

ACS also provides an alarm system to make the operator aware of system failures. An alarm set by any component will be enabled, until it is explicitly cleaned up by that component. Therefore, some of the situations described before that require operator intervention might trigger such an alarm.

# Appendix A

# ASDM Tables

Communication with TelCal through the established mechanisms requires the creation of two new ASDM Tables, `AppParameters` described in Section A.1 and `CalAppPhase` described in Section A.2. Documentation on existing types appears in [RD10]. A discussion appears in Section A.3.

## A.1 APP Parameters

On a per-subscan basis the set of antennas to be phased-up and co-added may be adjusted. The signals from one antenna of the array (corresponding to CAI-63) will have been replaced by the phased sum signals. The remaining antennas are available for use as comparison antennas (to estimate the efficiency of the phasing solution). One in particular is distinguished as *the* comparison antenna, and it may be changed as well. This table, shown in Table A.1, documents the process—changes may occur dynamically at the discretion of the operator or via heuristics of the VOM. At present it is envisioned that the `ParameterTuning` interface, rather than this table will be used for more subtle adjustments (*e.g.* antenna weighting strategy). However, the optional data part of the table could be expanded to document such things within the ASDM. Complete descriptions of

***Table A.1:*** *`AppParameters` ASDM Table*

| Name | Type(Shape) | Comment |
|---|---|---|
| *Key* | | |
| `basebandName` | `BasebandName` | identifies the baseband |
| `startTime` | `ArrayTime` | time of application |
| *Required Data* | | |
| `numPhasedAntennas` | `int` | $N_p$, number in phased sum |
| `phasedArrayId` | `Tag` [$N_p$] | phased antennas |
| `refAntennaIndex` | `int` | index (0..$N_p$−1) of the reference antenna in `phasedArrayId` |
| `numCompare` | `int` | $N_c$, number of comparison antennas |
| `compareArrayId` | `Tag` [$N_c$] | comparison antennas |
| `compAntennaIndex` | `int` | index (0..$N_c$ − 1) of the comparison antenna in `compareArrayId` |
| `phasedSumAntennaId` | `Tag` | the phased sum antenna |
| *Optional Data* | | |

the columns follow:

**basebandName** : For ALMA, a baseband pair is the signal path identified by a second local oscillator, and has two polarizations. For the APP, this corresponds to the two PICs in one

quadrant. This will be one of `BB_1` through `BB_4` or `BB_ALL` if all four quadrants are operated identically. The polarizations of each baseband are phased and compared similarly.

**startTime** : The (approximate) time the correlator was commanded to change to the configuration described in this table, or for TelCal to start using a different comparison antenna.

**numPhasedAntennas** : The number of antennas included in the phased sum.

**phasedArrayId** : The rows of the antenna table of the antennas contributing to the phased sum.

**refAntennaIndex** : The index $(0..N_p - 1)$of the reference antenna in `phasedArray`.

**numCompare** : The number of antennas not included in the phased sum.

**compareArrayId** : The rows of the antenna table of the antennas not in the phased sum, which could be used as comparison antenna. The array of available antennas has $(N_p + 1 + N_c)$ members; $N_p$ are in the phase-sum, one *is* the phased-sum, and $N_c$ are not.

**compAntennaIndex** : The index $(0..N_c - 1)$ of the comparison antenna in `compareArray` to be used in efficiency or quality measurements.

**phasedSumAntennaId** : The row of the antenna table of the antenna whose data is discarded in favor of the phased sum. The antenna is also known as `cai63Antenna`. The efficiency is measured through the correlation of this antenna with the `compAntenna` or its peers.

## A.2 CAL APP Phase

This table, shown in Table A.2, provides the results of the TelCal phase calculation on the "slow" timescale loop, *i.e.* at the end of a subscan and based on the data within that subscan following the last "slow" phase adjustment. Complete descriptions of the columns follow:

**basebandName** : For ALMA, a baseband pair is the signal path identified by a second local oscillator, and has two polarizations. For the APP, this corresponds to the two PICs in one quadrant. This will be one of `BB_1` through `BB_4` or `BB_ALL` if all four quadrants are operated identically.

**startTime** : The start of the interval in which the phase solution was calculated, *i.e.* at the beginning of the "valid for TelCal" portion of the subscan.

**calDataId** : Provides a link to the `CalData` table containing information about the data used to produce the result (*e.g.* scan number).

**calReductionId** : Provides a link to the `CalReduction` table which contains information concerning the data reduction.

**endTime** : The end of the interval in which the phase solution was calculated, *i.e.* the end of the last subscan.

**startValidTime** : The phase data shall be considered invalid for application before this time.

**endValidTime** : The phase data shall be considered invalid for application after this time.

**phasingMode** : The mode in which the phasing system is being operated; see Section A.3.

**phasingType** : Indicates one of several possibilities for converting the phase data into TFB commands; see Section A.3.

**numPhasedAntennas** : The number of antennas included in the phased sum.

**phasedArray** : The names of the antennas contributing to the phased sum.

**refAntennaIndex** : The index of the reference antenna in `phasedArray`.

*Table A.2: `CalAppPhase` ASDM Table*

| Name | Type(Shape) | Comment |
|---|---|---|
| *Key* | | |
| `basebandName` | `BasebandName` | identifies the baseband |
| `startTime` | `ArrayTime` | start of phase interval |
| `calDataId` | `Tag` | refers to a unique row in CalData Table |
| `calReductionId` | `Tag` | refers to a unique row in CalReduction Table |
| *Required Data* | | |
| `endTime` | `ArrayTime` | end of phase interval |
| `startValidTime` | `ArrayTime` | time-bound on validity |
| `endValidTime` | `ArrayTime` | time-bound on validity |
| `phasingMode` | `string` | applicable phasing mode |
| `numPhasedAntennas` | `int` | $N_p$, number in phased sum |
| `phasedArray` | `string` [$N_p$] | phased antennas |
| `refAntennaIndex` | `int` | index (0..$N_p-1$) of the reference antenna in `phasedArray` |
| `candRefAntennaIndex` | `int` | index (0..$N_p - 1$) of a candidate (new) reference antenna in `phasedArray` |
| `quality` | `float` [$N_p$] | quality of phased antennas |
| `numValues` | `int` | $N_v$, number of data values |
| `phaseValues` | `float` [$N_v$] | array of phase data values |
| `numChannels` | `int` | $N_d$, number of data channels |
| `numCompare` | `int` | $N_c$, number of comparison antennas |
| `compareArray` | `string` [$N_c$] | comparison antennas |
| `compAntennaIndex` | `int` | index (0..$N_c-1$) of the comparison antenna in `compareArray` |
| `candCompAntennaIndex` | `int` | index (0..$N_c - 1$) of a candidate (new) comparison antenna in `compareArray` |
| `efficiency` | `float` [$N_d \times N_c$] | efficiency of phased sum |
| `phasedSumAntenna` | `string` | the phased sum antenna |
| *Optional Data* | | |
| `numSupports` | `int` | $N_s$, number of supporting data values |
| `phaseSupports` | `float` [$N_s$] | array of supporting data values |

**candRefAntennaIndex** : TelCal may recommend the adoption of a candidate (new) `refAntenna` with this entry. Note that the VOM may not adopt the recommendation.

**quality** : A normalized figure of merit expressing the quality of the solution for each of the `phasedArray` members.

**numValues** : The number of phase data values present in the table; see Section A.3.

**phaseValues** : An array containing the phase data values.

**numChannels** : The number of data channels for which efficiency data is presented.

**numCompare** : The number of antennas not included in the phased sum.

**compareArray** : The names of the antennas not in the phased sum, which could be used as comparison antenna. The array of available antennas has ($N_p + 1 + N_c$) members; $N_p$ are in the phase-sum, one *is* the phased-sum, and $N_c$ are not.

**compAntennaIndex** : The index of the comparison antenna in `compareArray` used in efficiency or quality measurements.

**candCompAntenna** : TelCal may recommend the adoption of a candidate (new) `compAntenna` with this entry. Note that the VOM may not adopt the recommendation.

**efficiency** : An array of normalized efficiencies for the phased sum for each data channel. The `compAntenna` values are to be used for decisions; the other values are advisory.

**phasedSumAntenna** : The name of the antenna whose data is discarded in favor of the phased sum. The antenna is also known as `cai63Antenna`. The efficiency is measured through the correlation of this antenna with the `compAntenna` or its peers.

**numSupports** : The number of supporting data values present.

**phaseSupports** : An array of $N_s$ supporting data values. The presence and use of this array is unspecified, but might include channel average frequencies or supplementary `quality` data as an assist in the implementation.

## A.3 ASDM Discussion

In order to work with the existing TelCal infrastructure, it is necessary to define these two new tables so that phasing activities can be processed by TelCal and documented in the ASDM. Since these tables are small, infrequent and new, the impact on other consumers of the ASDM (*e.g.* CASA) should be nonexistent, and the impact on ALMA (disk space or communications bandwidth) trivial.

Note that aside from these tables, TelCal will receive important instruction through another interface, the `ParameterTuning` interface. These include the `sourceModel` and algorithm to be used for phasing the array and are made on a per-VLBI scan basis.

The design of the phasing system is such as to allow future implementations to exceed the capabilities of the current scope; thus these tables have been defined with an eye for flexibility beyond the current APP project scope. Equally, provision is made to allow some evolution within the current project scope to proceed without invalidating these tables. For the present, all needs can be met with a character string, `phasingMode`, which specifies to all consumers of this table how the data was derived, and how to interpret the data values. As a practical matter, the string token provides a relatively small number of bits of information (see discussion in Section 5.4.1) and could as well be implemented as an enumeration or an `int` with bits defining certain choices in which of the implemented capabilities to use:

**Active/Passive/Model** at least 2 bits

**use-WVR/no-WVR** 2 bit (also: relative or absolute)

**use-RDC/no-RDC** 1 bit

**Simple/Complex** at least 1 bit

and independent of these, an enumerator of different data packing formats used to interpret `phaseValues`. Some obvious choices are:

- one phase per polarization per channel-average per antenna,
- four phases per antenna (*i.e.* a delay-like solution in two polarizations),
- $2 \times 32$-phases per antenna, or
- any of the above, but with additional parameters supplied to allow some secular variation, *e.g.* a slow variation of phase with time.

Generally speaking, TelCal would be providing one phase per channel-average datum, but as there are 32 TFBs per polarization, there are situations where it is desirable to provide more channel-averages than needed, or fewer phase solutions than channel averages. So an allowance for these cases is also in the mix. The channel average frequency spectral window is in any case available to TelCal as input, so it need not be provided in these tables; but to a user of the CalAppTable (other than the main user, control/correlator) the link to the channel average spectral window through the (`calDataId` + `basebandName`) is a little intricate, so the frequencies could be explicit here as optional data (as a component of the `phaseSupports` data which is more generally cast to allow other usage).

The APP project is planned to always operate in a dual-polarization mode, but potentially some projects might choose to work with just one polarization. If so, flagging that is yet another consideration. In addition to that consideration, in the "Complex" case where the CDP nodes are assigned a perform fitting on the "fast" cycle, this table could be used to provide those nodes the expected phases in the source model and, *e.g.* the antenna weighting or the matrix to be used in the least-squares-fit solution process. These notions are discussed in [RD2]; most likely the APP project will only implement the "Simple" mode with one phase per channel-average.

Finally, the `quality` parameter is intended to allow simple heuristics (*e.g.* threhold based) to be used automatically to adjust the optimal set of antennas used for phasing. TelCal can compute this based on RMS residuals of the fit, for example. However there may be other choices. The `phasingMode` would be a place to indicate what method was chosen. (The command choice would be made the `ParameterTuning` interface.)

In this version, antennas are referenced by names (*e.g.* DV01 or DA42), or `antennaId` according to whether those tables are available in the context needed. (*I.e.* they are available for `AppParameters` but not for `CalAppPhase`.)

It is currently envisioned that TelCal would provide calculation of the existing phase error rather then the value of correction to be made. That is TelCal is providing a report of the phase error per polarization per antenna for the CCC to correct. (The alternative of providing the correction is algebraically identical; however this choice is probably less confusing—TelCal tends to report how things are, rather than how to make things otherwise.)

## A.4 Additional ASDM Modifications

In addition to the ASDM tables we need to extend two enumerations: `ScanIntent` should accomodate two more entries: `CALIBRATE_APPPHASE_ACTIVE` (indicating continous phasing on strong target source) and `CALIBRATE_APPPHASE_PASSIVE` (indicating monitoring of phasing for a weak target source). The enumerator `CalType` CalType should have one more entry: `CAL_APPPHASE` (calibration result type, referenced by the ASDM Scan table).

Note however, that we are in the process of reconciling our needs with those which the VLA has already used (see Section 4.1.3, so the precise set of changes is not established at this time.)

# Appendix B

# Source Files

This section lists locations of planned source code changes. Since some of the paths are long enough to be unreadable when printed, a parent directory (`$Dir`) is defined, where `$HEAD` refers to an SVN checkout of some release branch or trunk. In addition, the relevant features defined in the management plan [RD3] are also identified. New paths are marked with "*(NEW)*".

- CONTROL SFI Observing Mode
  - Dir: `$HEAD/CONTROL/ObservingModes/Array/SingleFieldInterferometry`
  - IDL: `$Dir/idl/SingleFieldInterferometryObservingMode.idl`
  - Code: `$Dir/src/`
  - Features: 5.2.1, 5.2.2, 5.2.3, 5.2.4
- CONTROL InterferometryController
  - Dir: `$HEAD/CONTROL/ObservingModes/Array/InterferometryController`
  - IDL: `$Dir/idl/InterferometryController.idl`
  - Code: `$Dir/src/`
  - Features: 5.2.1, 5.2.4
- CONTROL PhasingController *(NEW)*
  - Dir: `$HEAD/CONTROL/ObservingModes/Array/PhasingController`
  - IDL: `$Dir/idl/PhasingController.idl`
  - Code: `$Dir/src/`
  - Features: 5.2.2
- CONTROL VLBIController *(NEW)*
  - Dir: `$HEAD/CONTROL/ObservingModes/Array/VLBIController`
  - IDL: `$Dir/idl/VLBIController.idl`
  - Code: `$Dir/src/`
  - Features: 5.2.3
- CONTROL ExecutionState
  - Dir: `$HEAD/CONTROL/Common/ExecState`
  - IDL: `$Dir/idl/ExecState.idl`
  - Code: `$Dir/src/`
  - Features: 5.2.1

- CONTROL Hydrogen Maser *(NEW)*
  - Dir: `$HEAD/CONTROL/Device/HardwareDevice/Maser`
  - IDL: `$Dir/idl`
  - Code: `$Dir/src`
  - Features: 5.1.1
- CONTROL Optical Fiber Link System *(NEW)*
  - Dir: `$HEAD/CONTROL/Device/HardwareDevice/VLBIOFLS`
  - IDL: `$Dir/idl`
  - Code: `$Dir/src`
  - Features: 5.1.2
- CONTROL VLBI Recorder *(NEW)*
  - Dir: `$HEAD/CONTROL/Device/HardwareDevice/VLBIRecorder`
  - IDL: `$Dir/idl`
  - Code: `$Dir/src`
  - Features: 5.1.3
- CONTROL DataCapturer
  - Dir: `$HEAD`
  - IDL: `$Dir/ICD/OFFLINE/idl/DataCapture.idl`
  - Code: `$Dir/OFFLINE/DataCapturer/src/` *(to be moved under CONTROL soon)*
  - Features: 5.2.5
- CONTROL Hardware GUI
  - Dir: `$HEAD/CONTROL/Common/GUI`
  - IDL: `-`
  - Code: `$Dir/src/`
  - Features: 5.7.1
- CONTROL Array GUI
  - Dir: `$HEAD/CONTROL/Array`
  - IDL: `$Dir/src/alma/Control/ArrayStatusGui/`
  - Code: `$Dir/idl/ArrayStatus.idl`
  - Features: 5.7.4
- CONTROL QuickLook
  - Dir: `$HEAD/PIPELINE/QuickLook`
  - IDL: `$Dir/src/`
  - Code: `$Dir/idl/`
  - Features: 5.7.2
- CONTROL VLBI Utilities *(NEW)*
  - Dir: `$HEAD/CONTROL/Common/VLBIUtils`
  - IDL: `-`
  - Code: `$Dir/src`

  - Features: 5.6.1, 5.6.2, 5.6.3, 5.6.4, 5.6.5, 5.6.6, 5.6.7

- CORR ObservationControl
  - Dir: `$HEAD`
  - IDL: `$Dir/ICD/CORR/idl/ObservationControl.idl`
  - Code: `$Dir/CORR/CCC/ACSIF/src/`
  - Features: 5.3.1, 5.3.2, 5.3.3

- CORR ObservationQuery
  - Dir: `$HEAD`
  - IDL: `$Dir/ICD/CORR/idl/ObservationQuery.idl`
  - Code: `$Dir/CORR/CCC/ObservationQuery/src/`
  - Features: 5.3.2, 5.3.4

- CORR CAN Cmd
  - Dir: `$HEAD/CORR/CCC/CAN_Cmds`
  - IDL: -
  - Code: `$Dir/src/`
  - Features: 5.3.1, 5.3.2, 5.3.3

- CORR SubarrayMgt
  - Dir: `$HEAD/CORR/CCC/SubarrayMgt`
  - IDL: -
  - Code: `$Dir/src/`
  - Features: 5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.5

- CORR CCC Simulation
  - Dir: `$HEAD/CORR/CCC/Simulation`
  - IDL: -
  - Code: `$Dir/src/`
  - Features: 5.3.6

- CORR CDP Master
  - Dir: `$HEAD/CORR/CDP`
  - IDL: `$Dir/IF/idl/Master.idl`
  - Code: `$Dir/Master/MasterImpl/src/`
  - Features: 5.3.5

- CORR CDP Node
  - Dir: `$HEAD/CORR/CDP`
  - IDL: `$Dir/IF/idl/Node.idl`
  - Code: `$Dir/Node/NodeImpl/src/`
  - Features: 5.3.5, 5.3.6

- CORR GUI
  - Dir: `$HEAD/CORR/GUI`
  - IDL: -

  - Code: `$Dir/src/`
  - Features: 5.7.3

- TELCAL DataManager
  - Dir: `$HEAD`
  - IDL: `$Dir/ICD/TELCAL/idl/TelCalDataManager.idl`
  - Code: `$Dir/TELCAL/TelCalDataManager/src/`
  - Features: 5.4.1, 5.4.2, 5.4.3, 5.4.4, 5.4.5

- TELCAL Publisher
  - Dir: `$HEAD`
  - IDL: `$Dir/ICD/TELCAL/idl/TelCalPublisher.idl`
  - Code: `$Dir/TELCAL/TelCalPublisher/src/`
  - Features: 5.4.1, 5.4.2, 5.4.3, 5.4.4, 5.4.5

- TELCAL ParameterTuning
  - Dir: `$HEAD`
  - IDL: `$Dir/ICD/TELCAL/idl/TelCalParameterTuning.idl`
  - Code: `$Dir/TELCAL/TelCalDataManager/src/`
  - Features: 5.4.1, 5.4.2, 5.4.3, 5.4.4, 5.4.5

# Appendix C

# VOM Prototype Interfaces

Here we present some details of the interfaces to supplement their usage as sketched out in the timing diagrams, Figures 5.2, 5.3, 5.4, 5.5, and 6.2. Some structures and parameters are pending to be defined in detail in detail at this point. However, it is expected that implementation of the actual IDL files shall obsolete this discussion, except perhaps as an historical artifact. Existing and new IDL file locations in the code repository are listed in Appendix B.

## C.1 CONTROL: SingleFieldInterferometryObservingMode.idl

*This is an addition to the existing interface.*

```
module Control
{
    enum AppMode {...};

    struct AppScheduleStruct {
        ...
    };

    struct AppPhasingEfficiencyStruct {
        ...
    };

    struct AppVDIFChannels {
        long num_chan_log2;
        long thread_id;
        long station_id;
    };

    interface SingleFieldInterferometryObservingMode :
      ObservingModeBase,
      ObservingModeInterface,
      InterferometricLocalOscillatorInterface,
      PointingSubarrayControllerInterface,
      InterferometryControllerInterface
    {
        void setPhasingController(...);

        void setVLBIController(in AppVDIFChannels channels,
            in AppScheduleStruct schedule);

        void doAppScanSequence(in ScanIntentData[] scan, in ACS::Time startTime,
```

```
            in AppMode mode, in SimpleCallback callback);

        void doCalibrationSequence(...);

        void setAppPhasedArray(...);

        AppScheduleStruct getAppSchedule();

        AntennaSeq getAppAntennas(...);

        AppPhasingEfficiencyStruct getAppPhasingEfficiency(...);

        ACS::TimeInterval getAppTimeOffset();

    };
};
```

## C.2 CONTROL: InterferometryController.idl

*This is a new interface definition.*

```
module Control
{
    interface InterferometryControllerInterface : ACS::OffShoot
    {

        void setAppSumAntenna(in string antennaName, in double xPos,
            in double yPos, in double zPos, in double cableDelay);

    };
};
```

## C.3 CONTROL: PhasingController.idl

*This is a new interface definition.*

```
module Control
{
    interface PhasingControllerInterface : ACS::OffShoot
    {
        void setAppScanParameters(in AppMode algorithmMode,
            in SourceModel sourceModel);

        void subscanStarted();

        void subscanEnded();

    };
};
```

## C.4 CONTROL: VLBIController.idl

*This is a new interface definition.*

```
module Control
```

```
{
    interface VLBIControllerInterface : ACS::OffShoot
    {
        void scanSequenceEnded();

    };
};
```

## C.5 CONTROL: ExecState.idl

*This is an addition to the existing interface.*

```
module Control
{
    interface ExecutionState : Controller
    {
        void doAppScanSequence(in ACS::Time startTime);

        bool isAppPhaseResultAvailable(in double timeout);

        AppPhaseResult getAppPhaseResult();

    };
};
```

## C.6 CONTROL: DataCapture.idl

*This is an addition to the existing interface.*

```
module offline
{
    interface DataCapturer : ACS::ACSComponent
    {
        void sendAppParametersData(in Control::AntennaSeq mask,
            in string refAntenna, in ACS::Time timestamp);
    };
};
```

## C.7 CORR: ObservationControl.idl

*This is an addition to the existing interface.*

```
module Correlator
{
    interface ObservationControl: CorrelatorPrivate::CorrStateModel
    {
        void setAppParameters(in string subarrayId, in Control::AntennaSeq mask,
            in string refAntenna);

        void setupPIC(in AppVDIFChannels channels);

        void setAppScanParameters(in AppMode algorithmMode);

        ACS::Time setAppSlow(in string subarrayId, in Control::AntennaSeq mask,
            in string refAntenna, in doubleSeq phaseSeq, in ACS::Time startTime,
```

```
            in ACS::Time endTime);

        void setPICinvalid(in bool validity);

    };
};
```

## C.8 CORR: ObservationQuery.idl

*This is an addition to the existing interface.*

```
module Correlator
{
    struct PICStatusStruct {
        ...
    };

    struct PhaseUpdateRequestStruct {
        ...
    };

    struct PhaseUpdateEvent {
        ...
    };

    interface ObservationQuery : CorrelatorPrivate::CorrStateModel
    {
        PICStatusStruct getPICStatus(in longSeq picIds);

        void requestPhaseUpdates(in PhaseUpdateRequestStruct request);

    };
};
```

## C.9 CORR: Master.idl

*This is an addition to the existing interface.*

```
module CorrelatorPrivate
{
    interface Master : CorrelatorPrivate::CorrStateModel
    {
        void setAppMode(in AppMode algorithmMode);

        void setAppSlow(in string arrayId, in doubleSeq phaseSeq,
            in ACS::Time timestamp, in SimpleCallback cb);

    };
};
```

## C.10 CORR: Node.idl

*This is an addition to the existing interface.*

```
module CorrelatorPrivate
```

```
{
    interface Node: CorrelatorPrivate::CorrStateModel
    {
        void setAppMode(in AppMode algorithmMode);

        void setAppSlow(in string arrayId, in doubleSeq phaseSeq,
            in ACS::Time timestamp);

    };
};
```

## C.11 TELCAL: TelCalPublisher.idl

*This is an addition to the existing interface.*

```
module telcal
{
    struct AppPhaseReducedEvent {
        ...
    };

    struct AppPhaseResult {
        asdmIDLTypes::IDLEntityRef execBlockId;
        long scanNum;
        ...
    };

    interface GetTelCalResults : ACS::ACSComponent
    {
        AppPhaseResult getAppPhaseResult(
            in asdmIDLTypes::IDLEntityRef execBlockId, in long scanNum);

    };
};
```

## C.12 TELCAL: TelCalParameterTuning.idl

*This is an addition to the existing interface.*

```
module telcal
{
    interface ParameterTuning : ACS::ACSComponent
    {
        setAppScanParameter(in string coeffName, in double value,
            in string arrayId, in boolean on);

        setAppScanParameterAsString(in string coeffName, in string value,
            in string arrayId, in boolean on);

    };
};
```

# Appendix D

# Phase Sum Threshold

A full discussion of the statistics of the 8-bit sum is to be found in [RD6]. Here we point out that the information required for the LTA protocol (viz, to populate the lookup table shown in Figure D.1) may be computed from the number of antennas and the 2-bit state statistics of the ALMA correlator.

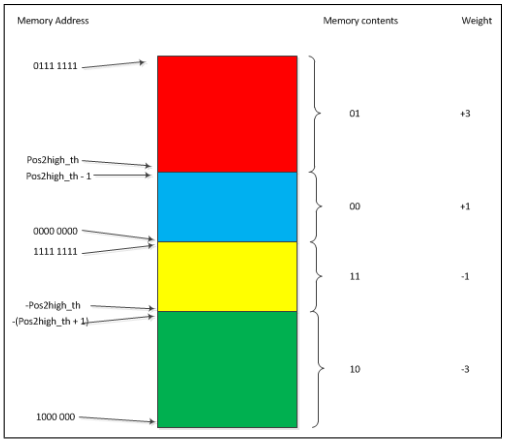


***Figure D.1:*** *LTA Sum 8-bit to 2-bit lookup table, from [RD6]. For odd numbers of antennas, the table is symmetric and well-described by a single parameter,* ***pos2high_th*** *which is* $\approx \mathrm{sqrt}(N_{\mathrm{antennas}})$.

A C implementation of the lookup table is (computed for the optimal combination of state weights of [-3,-1,1,3] and thresholds between $-3/-1$ or $1,3$ set at $\mp 1.0\sigma$, respectively, and is):

```
/*
 * Lookup table for pos2high_th[n], n <= 64, where
 *   n is the number of antennas; optimal if n is odd
 *   values also provided for n even, and single antenna
 *   state probabilities 0.158655 0.341345 0.341345 0.158655
 */
int pos2high_th[] = { 0,
  1,  2,  2,  2,  2,  3,  3,  3,  3,  4,  3,  4,  3,  4,  4,  4
  4,  4,  4,  5,  4,  5,  5,  5,  5,  5,  5,  6,  5,  6,  5,  6
  5,  6,  6,  6,  6,  6,  6,  6,  6,  7,  6,  7,  6,  7,  6,  7
  7,  7,  7,  7,  7,  7,  7,  8,  7,  8,  7,  8,  7,  8,  7,  8
};
```

Note that there is no good solution for an even number of antennas, but that this table supplies even n values for simplicity and operability in sub-optimal conditions or testing.

# Appendix E

# VDIF Header

The VDIF header (see Figure E.1) emitted by the PIC is generated from several data:

- the requested number of TFB data channels,
- the current UTC time, and
- the PIC generating the data.

Several of the header items are driven by time; the rest are configuration information needed to use the data.

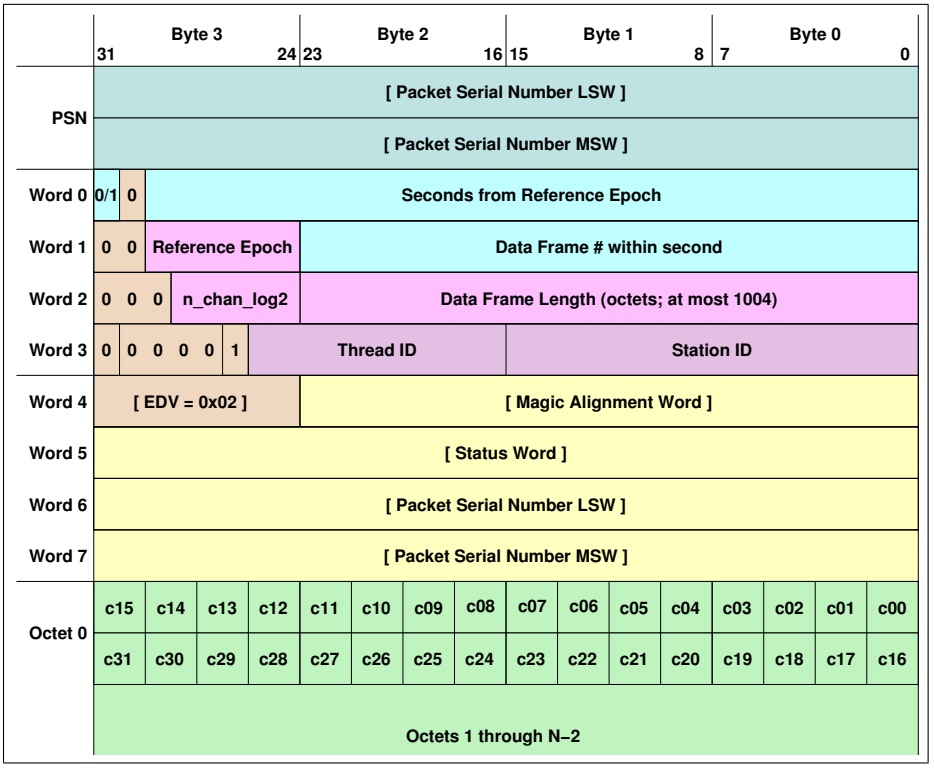


***Figure E.1:*** *VDIF Header structure, extracted from [RD6].*

The packet serial number (PSN) is used to double-check the ordering of packets both for data integrity, as well as to support the VDIF Transport Protocol (VTP). It appears before the packet proper (simplest version of VTP), is duplicated within the packet (for integrity checking of the recorded data), and is incremented by one with each packet. Its value is otherwise arbitrary and

*may* be initialized with zero. The frame counter is incremented with the PSN and is reset to zero every second (at which point the seconds is incremented).

At ALMA packet rates (*i.e.* at most 125,000 packets per second), the PSN will roll-over every 4 million years, so the highest bits could be set for useful debugging purposes; *e.g.*

```
unsigned long psn_initial;
psn_initial  = (0x1F & (year-2000)) << 58;
psn_initial |= (0x1F & month)       << 53;
psn_initial |= (0x1F & day)         << 48;
psn_initial |= (0x1F & hour)        << 43;
```

where year, month, day and hour have their usual calendar meanings and may be taken from the operating system (*e.g.* `gettimeofday()`); and this would produce a unique starting PSN for ALMA's projected lifetime.

The timekeeping in the header is by a count of integer seconds relative to one of 64 epochs identified by Modified Julian Day (MJD) and a frame counter which counts from zero to the number of packets per second minus one. The header is provided with values corresponding to a TE which coincides with a PPS. (This occurs every six seconds.) ALMA timekeeping uses `ACS::Time` which counts seconds from Oct 15, 1582. The epoch and seconds of epoch can then be easily computed from the target `ACS::Time` and a table of epochs, *e.g.*

```
ACS::Time target_TE = getACSTimeOfNextTECoincidentWithPPS();
ACS::Time VDIFepochs[65] = ... ;
int epoch, secs;
for (epoch = 0; epoch < 65 && target_TE < VDIFepochs[epoch]; epoch++) ;
epoch--;
secs = target_TE - VDIFepochs[epoch];
```

The actual table of epochs must be checked against the table used by DiFX (the current prime consumer of VDIF data).

The remaining information is provided to the CCC via a `VDIFchannels` structure:

```
struct VDIFchannels {
    int num_chan_log2;      // 0..5 for 1..32
    int thread_id;
    int station_id;
}
```

The number of data channels is a power of two and provided in the `num_chan_log2` field (*e.g.* 5 for 32 channels, 4 for 16, *&c.* ) and the packet size is given in octets (8-byte units), includes the header size (4) and is derived from this according to Table 3-1 of [RD6], *i.e.* 1,004 for packets with at least 4 channels, and 629 otherwise. The assignment of CAN bus addresses to the PICs in the four quadrants should be done in some fashion where the association of PICs to placement is logical.

The extended data header values (words 4 through 7) are zeroed on input. The PIC will fill them out appropriately for the output packets.

Following the creation of the header, it must be delivered at the appropriate TE; and then a subsequent status query must verify that the header was decoded correctly and *most importantly* that the timing information is correct. At this time the extended header data should be nonzero according to Figure E.1. An extended header data version (EDV) of `0x2` has been reserved for ALMA's use.

# Appendix F

# Phasing Loop Timing

The latency of the phasing loop is the time it takes to apply the next TelCal phase correction in the TFB cards, starting from the end of the previous ALMA scan. The scan timescales were described in Section 3.3.

Detailed studies of the timing have yet to be completed. However, at this point, the timing of several sections of the loop are reasonably well understood. Current estimations were done with 30 to 40 antennas and least-square phase calculation.

With the existing system, the minimum time is perhaps 10 seconds. With some straightforward improvements, this can be reduced to about 8 seconds. The initial phasing study work suggested that 10 seconds was acceptable.

## F.1 Latency of Data Arrival in TelCal

The minimum dump time in mode 13 is 0.512 seconds, but the computing infrastructure (software backend, ACS bulk data transport mechanism and networking hardware) currently only supports an average of about 30 MB/s. There are plans to improve this in the future to 60 MB/s.

Channel average data can be computed with this 0.512-s time resolution.

The data limit that must be respected comes from the following calculation (assuming 62.5 MB/s data rate):

```
        4 correlation products
     2048 spectral points (from 32 TFBs with ~ 1 MHz resolution)
  64*63/2 baselines (at most)
 ******** (multiply together to get)
 16515072 data items /integration /quadrant
          (ignoring overhead bytes)
       8B /data item (Re,Im as 4B floats or equivalent)
 = 132 MB /integration /quadrant (1024*1024B/MB)
 = 528 MB /integration
 = 8.064 sec (at 62.5 MB/s or 500 Mbps)
```

Slightly more time should be allowed for a safety margin. Other data is transferred, but it constitutes considerably less bulk. The spectral averaging capability reduces the data by $2^n$ for some integer $n < 11$. So, this 8.1-s integration time is reduced by $2^n$ to 4, 2, 1, ... if one is willing to sacrifice spectral resolution in the ALMA dataset. (To enable VLBI and decent phasing performance, we certainly are.)

So it is the number of integrations (at least one) per subscan coupled with the spectral resolution which imposes a latency for TelCal to get data to process.

The CDP current overhead to deliver this data is 5 s. This includes processing and packaging time, and transmission from the nodes to the master.

Additionally, the CDP allows to start processing data of the next subscan before the data from the previous one has been completely transmitted to the clients (TelCal and Archive). However,

transmission latencies could propagate to subsequent subscans at large data rates and long enough subscans. We might want to consider extending the inter-subscan time artificially (by design it is close to zero when in VLBI mode) to allow enough transmission time. Doing so within the total time it takes to actually apply the next TelCal results would not impact the phasing system (as this portion is flagged for the next TelCal calculation anyway), only the length of subscans for post-processing from the ALMA Archive.

Therefore, with the sacrifice of all spectral data (retaining only channel average data), the minimum overhead is about 5.5 s (overhead of sending no data plus 0.512 s).

## F.2 Time to Solve

The processing steps in TelCal consist of reading and time averaging the data, obtaining the associated metadata, possibly calculating the source model, running the least square solver and publishing the results. The time-lag introduced by these operations is dominated by the data access, which can take 1-2 seconds. The actual solving process (least-squares-fit) runs on time scales of milliseconds and can be neglected. We expect the latency of the slow loop to be less than 3 seconds from the time TelCal is notified of the BDF completeness until it has published the new phasing solutions.

## F.3 Time to Reply

This is the time TelCal results take to be transmitted back to the observing mode and the correlator (CORBA messages). This time is unmeasured, but reply content is small and should be at the order of tens of ms. Note that this time might include some decision making in the observing mode regarding the phased array composition.

## F.4 Time to Command

The message to download new phases from the CCC to each TFB card requires about 220 ms for delivery. (This is consistent with the required number of bits and the speed of the CAN bus.) There are four independent CAN bus lines in the CCC so these transfers could be executed in parallel, but at present they are not. So this currently takes about 1.5 seconds. Parallelizing this is not hard, reducing the total time required to about 300 ms.

Note that each TFB receives its own phase value, but these are not independent—the phases are either the same for all or “delay-like”. So, the data, and hence the time for downloading could be reduced by a factor of 32 for our needs (less than 10 ms).