ALMA Software Science Requirements: Version 1.0 Report

ALMA Software Science Requirements Committee
R. Lucas, B. Clark, J. Mangum, P. Schilke, S. Scott, F.Viallefond, M. Wright

May 29, 2000

Revision History

Date	Revision	Description
2000-03-07	0.1	Alma Memo 293
2000-05-29	1.0	Reviewed version: Alma Memo 293.1

Contents

1 Introduction **5**

2 Real Time Software **7**

2.1 The basic operation modes . . . 7

2.2 Control Command Language . . . 7

2.3 Graphical User Interface . . . 10

2.4 Observing Modes . . . 11

2.4.1 List of observing modes . . . 11

2.4.2 Sample astronomer input for an observing mode: Single-field Interferometric mapping 13

3 Proposal Submission and Handling **14**

3.1 Basic requirements . . . 14

3.2 Advanced features . . . 15

4 Dynamic Scheduling **16**

4.1 Requirements . . . 16

4.2 Implementation . . . 16

5 Operator Interface **18**

5.1 Operator Model . . . 18

5.2 Operator Responsibilities . . . 18

5.3 Operator Interface Software . . . 18

6 Data Pipeline **19**

6.1 Goals . . . 19

6.2 Requirements . . . 19

6.3 Functionalities . . . 20

6.3.1 Array Calibration . . . 20

6.3.2 Calibration of Interferometer Data . . . 20

6.3.3 Calibration of the Total Power measurements . . . 20

6.3.4 Imaging . . . 21

6.4 Interaction with other Actors . . . 21

7 Archiving **22**

A Script language **23**

A.1 Special sections . . . 24

A.2 Control structures . . . 25

A.3 Describing a source . . . 26
A.4 Describing the correlator setup . . . 27
A.5 Describing a set of observations . . . 28
A.5.1 Antenna . . . 28
A.5.2 Subreflector . . . 29
A.5.3 Receiver and IFs . . . 29
A.5.4 Basebands . . . 30
A.5.5 Correlator . . . 31
A.5.6 Real-time data processing . . . 31

1 Introduction

This document intends to give an overview of the science-driven software requirements of the ALMA project; it is a snapshot of a work in progress that is intended to be broadly reviewed. It will be followed in a few months by a more detailed report where we will consider in detail the requirements of the various components of ALMA software.

The operation of ALMA will have to deal with a larger variety of projects than previous instruments: on one hand at long wavelengths (1-3 mm) due to the high sensitivity and quality of the site, and a long experience with millimeter-wave interferometry, we can predict with reasonable certainty the observing modes that will be used, the relevant observing strategies to schedule the instrument, and the data reduction techniques. On the other hand at the highest frequencies ($\sim 300\mu$m) no array has been operational yet; we plan to rely on techniques such as radiometric phase correction, fast phase switching and phase transfer between frequency bands, that have been demonstrated, but not applied with the operational scale that we foresee for ALMA. We thus will have to combine in the software a high level of automation, needed to deal with the large information rate that will be available, with a high level of flexibility at all levels to be able to develop and implement new observing methods and reduction procedures. For simple projects the astronomer with little or no experience of radio techniques should be able to use the instrument and obtain good quality results; however experts should easily be able to perform experiments we do not even foresee today.

The expert user/developer will need to be able to send direct commands to the instrument through a simple, easily editable command language (Section 2.2). Atomic commands in a script language will directly send orders to the basic software elements controlling the hardware: antenna motion, instrument setup, or transmitting parameters to the data processing (pipeline). The script language will support loops, structured conditional tests, parametrised procedures, global variables and arrays ... These scripts, once fully developed and tested, will evolve into the basic observing procedures of the instrument.

The general user will need more user-friendly graphic interfaces to many components of the system (Section 2.3). They will propose several templates, corresponding to the available observing modes, and provide a simple way to pass astronomy parameters to the basic observing process, and to the corresponding data reduction procedures of the pipeline. Input parameters will preferably be expressed in terms of astronomical quantities, which will be translated into technical parameters by sophisticated configuration tools.

In Section 2.4 we give a list of the basic observing modes and examples of templates.

Proposal submission will be in two phases, the first before proposal evaluation, the second to provide information needed for the actual scheduling and observation. The tool that will have to be provided for this process are described in Section 3.

We believe that dynamic scheduling is an essential feature of the instrument and should be installed from the very beginning of its operational life. Though the site is undoubtedly one of the best for submillimeter observations, it will only be usable at the highest frequencies for a fraction of the time; to improve the total efficiency we must be able to make the best use of all weather conditions, by selecting in quasi-real time the project most suited to the current weather and to the state of the array. This means we should always be able to observe a given project in appropriate weather conditions. This philosophy can be extended to the point where a given project can change its own observing parameters according to variations in observing conditions (such as atmospheric phase rms). In Section 4 we explain how these two levels of dynamic scheduling can be implemented and what are the requirements on software.

The whole real-time system will be under control of a telescope operator, through a specially designed interface. This must provide an overview of what observation is occurring, the state of the instrument, and observing conditions on the site, and should enable the operator to react to any unexpected event (Section 5). A general monitoring interface must be also accessible through the network.

The instrument should produce images, aiming to be final for most projects, even when projects are spread over several sessions and configurations, and/or include short/zero spacings. For this purpose an on-line pipeline is required (Section 6). It will include calibration of the array itself, to reduce measurements of baseline, delay offsets, and determine pointing models during specific sessions. During standard observing sessions reference pointing and focusing measurements will have to be reduced, with fast loop-back of results to the observing process; the phase fluctuations on the phase calibrators must be evaluated, with a feedback to both the real time process and the scheduler (Section 4). Calibration will be applied on-line and maps/datacubes will be produced according to data processing parameters input by the observer. Single-dish observing sessions will also be reduced on-line. The pipeline must be able to reduce on line the quasi totality of the data, which is expected to be produced at an average rate of 3 MB/s, with a peak rate of 30 MB/s for some observing sessions.

For most projects the data pipeline will produce results in a form suitable for quality evaluation, and astronomical processing, hopefully leading to fast publication. Uncalibrated *uv* data will be archived together with the calibrations curves and the resulting images. The archive should enable fast access to the observing parameters and full reprocessing of the data set with improved processing algorithms (Section 7).

A general requirement is that the various parts of the system should be developed in a highly consistent way, from the very beginning of design; they may however be installed progressively, provided the critical elements are implemented first.

2 Real Time Software

2.1 The basic operation modes

The actual control of ALMA observations will be run by the ALMA operations staff. There will be basically four modes of operation, with increasing levels of automation:

1. Through a **technical interface**, allow/deny technicians complete access to all the control points available on each device, for debugging and maintenance purposes. The system should include such a facility, under control of the operator, for determining whether the device is to be used for observing or is withdrawn for maintenance.

2. Control the array in **manual mode**. This mode will allow direct control on the hard-real-time system and on the quasi-real-time system (see below). This will provide a way to test all the functionalities of the array, as well as to develop and debug new ways of observing.

3. Start astronomical observations or array calibrations in the **interactive mode**. This will pass the actual control of observations to the interactive astronomer (guest or staff), through a **graphical interface**. This interface will allow control of the observing process, with automatic feedback from the data pipeline. The astronomer and/or the operator will have the possibilities to interrupt a sequence of events at any time, to modify the observing strategy, to switch to a different mode to perform needed but unforeseen calibrations, or to switch to the dynamically scheduled observing mode. While the manual mode is mainly intended for instrumental tests and procedure development, the interactive mode is aimed for observations using the standard modes, but requiring interactive observing (e.g. targets of opportunity for which a fast reaction time is essential).

4. Pass control to the **dynamic scheduler**, which will select among its queue the observing project the most suitable for the current observing conditions, and run it in the most efficient way, using up-to-date environment parameters from both the real time system and the data pipeline. Though scheduled in a different way, the available observing modes will be the same as for interactive scheduling; in fact the astronomical parameters will have been input by the astronomer (PI/CoI), with the same graphical interface as above, but used long before the actual observing. Projects requesting to be scheduled at a precise time, like VLBI, or observation of comets, occultations, .. etc. will be included here but with a top priority in the required time window.

For ALMA we need the possibility of dividing the antennas in several **sub-arrays**: for instance, assigning some antennas to an interactive observer or to the dynamic scheduler, and others to operations staff for pointing model determination, baseline determination, or engineering tests. Sub-arrays will be operated simultaneously through different operation sessions, each sub-array being in any of the four above modes. he antennas will be allocated to those sessions by the operator with highest priority to maintenance and array calibration sessions, since they condition the general performance of the instrument. De-allocation of an antenna from an observing session may require some calibrations to be performed before the antenna is handled to highest priority operations. The allocation of antennas to sub-arrays/sessions will take into account the hardware constraints imposed by local oscillator control (up to 4 different simultaneous LO setups), and by the correlator (up to 16 different correlator sub-arrays).

In the next two subsections (2.2, 2.3 we describe the basic requirements on the manual mode of operation, and on the interactive mode. The requirements on the dynamic scheduler are detailed in a specific section (4).

2.2 Control Command Language

In this mode the astronomer will be given direct real-time control on the instrument (or part of it), either by typing simple commands, or more frequently by running pre-edited scripts. Because the real-time system cannot be worked upon without stopping observing, it is imperative that it be as reliable as possible, and sufficiently flexible to accommodate the inevitable evolution of observing modes as the instrument becomes better understood.

Entities to be controlled are:

- the hardware, controlled through what we call the hard-real-time software, exercising detailed control of the antennas, receivers, local oscillators, correlators, ... etc.; this include commands to initiate antenna motion to the source and tracking at the requested speed, and actual requests for integration being sent to correlators.
- the quasi-real-time software, or software that must be run only when observing conditions (including time) are known, e.g. to compute source coordinates in the antenna system, prepare LO and IF filter settings according to the requested frequency and Doppler tracking parameters, ...etc. but also to reset some observing parameters, such as integration times, collimation and focus corrections according to the atmospheric data or to data pipeline products;
- the data pipeline itself, to which basic data must be passed such as which data reduction software/procedure should be used, and some input parameters which are not present in the data header information, and which cannot be guessed from the data records themselves.

The key to flexibility in the observing system is that the constructs that it deals with must be as close to the elementary hardware capabilities of the instrument as possible. This must be balanced with the need that the input to this system must be sufficiently simple to be read by expert astronomers: they should be able to decide by just looking at a script whether it will cause the instrument to do what the programmer had expected, without wasting any telescope time.

For this reason we require that the commands should be expressed in a simple command language, for which keywords are as much as possible words in a commonly understood language (we may agree on English). The number of single-character control operators must be strictly minimal.

Desirable features in the language built-in functionalities include:

- non-ambiguous abbreviation of keywords should be accepted
- macros for abbreviation of frequently typed sequences
- procedures to which parameters may be passed
- definition of variables and arrays, with numeric or character content
- evaluation of expressions, including built-in functions
- conditional execution facilities
- loops
- error recovery facilities
- interruption facility in procedure execution

Several scripting languages with these functionalities are actually being used at various observatories. The actual choice will mostly be based on software design considerations.

Foreseen use of variables/built-in functions is the ability to react on environment data such as meteorological data, phase monitor data, sidereal time, hour angle, source elevation, but also on pipeline output such as pointing collimations, detected calibrator flux, noise, flux and snr in the output map, ...

The exact balance of functions between observer's interface, quasi-real-time system, real-time system, and data processing pipeline is in the end likely to be dictated by constraints first seen in the detailed design of the software systems. However, it is a useful planning tool to have a first guess at the capabilities of the real-time system available, not only for designing the real-time system but also for designing the preceding tools and the archive data formats. We provide this first guess in an Appendix (A), in the form of a scripting language which might control the array. Note that this language is not really intended for direct use by astronomers, nor does it provide functions necessary for the array operators to control the observing. However, as an interface in the middle of what might otherwise be regarded as a very large black box, it is hoped that it can provide a planning tool for some of the other critical elements of the software system.

A few of the more difficult decisions for this scripting language/interface may be briefly noted:

- First, the baseline design for the ALMA local oscillator system requires fairly sophisticated choices to be made about the lock points for first and second local oscillators. The implication is that, although one can make a continuum observing file that can be given to the real-time system at any time, any useful spectral line file must be prepared for a specific date. This implies that the final LO/correlator setup can only be made at the actual time when an observing session is started. Advance warnings will have been given to the observer when running the Observing Tool, with predicted ranges for the LOs, and eventual restrictions. Second, the flexibility of the correlator hardware will be somewhat restricted because utilizing the full flexibility of the correlator would be technically difficult, and lead to more complication in the real-time system than desirable. Thus, we recommend that options to support different bandwidths or spectral windowing in the two halves of a baseband pair (usually used for opposite polarizations at the same frequency) be given a low priority.
- The suggested interface puts into the real-time system precession for J2000 to date. It is suggested that all reasonable corrections be put into the real-time system so that milliarcsecond level astrometry can be done differentially. It should include, eg, the latest IAU nutation, earth tides, general relativistic corrections and diurnal aberration. Interpolation of positions for solar system bodies would be supported, along with the phase correction for the first order parallax. Proper motion calculation for galactic objects should also be taken into account.
- Although most observing control will be done through the observer's tool, a couple of non-elementary concepts are supported by the proposed interface, in order to make the input scripts more readable for software debugging purposes by reducing the occurrence of repetitive text. These are first, macros, so that long descriptions of setups need not be repeated, and second, loops.
- Operating a subset of antennas as a sub-array for calibration etc., as mentioned above is a clear necessity. It also seems useful to us to provide provide also sub-arraying inside an observing project, permitting an observer to divide the antennas allocated to him into independent groups. Perhaps the most common use would be to detach three or four antennas to do tipping curves to evaluate the atmosphere, while the main body continues to make astronomical observations. In that case, if those antennas have to be handled in a different observing session, proper software connections should allow the results to be tested in the main session.

2.3 Graphical User Interface

The Graphical User Interface (GUI) is now the de facto standard in the commercial world. It is proposed that all interaction with the ALMA system will be through GUIs, except for the low level scripting that controls the real-time system.

Because GUIs are a fundamental part of the presentation of the instrument, they should be implemented with a common style and reusable components. Many interfaces will be required to the instrument and its data flow for different purposes. Administrative access occurs in the scheduling phase, engineering access during operations, and scientific access during an archive search. This implies that a variety of people using different computing platforms from different locations will access the instrument. As much as possible, the GUIs should support multiple platforms and remote access. All GUIs should have an embedded standardized help system so that documentation is always available.

Support for observing commands and sequences will be handled with an Observing Tool that must be able to produce either scripts or object sequences for the real-time engine, both outputs resulting in the same observations when executed. Integration with source catalogs, image databases, and spectral line catalogs are essential to create a high level tool. The Observing Tool can be used by an astronomer to prepare an observing script, or by the telescope operator to produce an updated script for a project.

The correlator complexity presents particular problems that require a GUI for most non-trivial setups. The Correlator Configuration Tool can be run stand-alone or be invoked by the Observation Tool.

A list of some of the major GUIs can serve as a model for interaction with the data flow. Details of some of these interfaces are found in other sections.

- Preparation of observations:
  - Proposal preparation tool
  - Archive search tool (historical project search)
  - Observation simulator
  - Observation tool
  - Correlator configuration tool
- Real time control and monitoring
  - Operator array control
  - Data cube viewer
  - Monitoring
    - Antenna based UV data
    - Live tables
    - Multi-variable plots versus time
    - Scatter plots
  - Engineering device control
  - Project status visualization tool
  - Data cube viewer
- Archive data extraction interface

2.4 Observing Modes

In this section we describe the observing modes of the instrument. This list cannot, by nature, be made definitive. Among our requirements is the fact that these modes can be tuned, and new modes be developed, without involving deep changes in the system: i.e. by editing scripts in the real time control language. When fully checked, these modes will be available through GUI interfaces to the general user.

2.4.1 List of observing modes

All these modes involve a sub-array. Any number of sub-arrays, working simultaneously on any different modes, different projects (and different observing frequencies) are possible.

Interferometric

Single Field Classical mode for interferometers. Alternate between on-source cross-correlation scans and phase calibration scans. Switching can be quite fast if needed, to follow atmospheric phase variations.

Multi Field Mosaics Same as above, but on-source cross-correlation scans are taken at nearby, pre-defined positions in the sky. Alternate between on-source and phase calibration scans.

On-The-Fly Mosaics The on-source cross-correlations are taken while the antennas drift on the sky along a pre-defined pattern. Every so many scans, phase calibration scans are done.

Phased Array Normally used for VLBI. On-calibrator cross-correlation scans are needed to calibrate the phases. Flexible control over which antennas are used for the phased array must be provided, to optionally eliminate antennas which are deeply shadowed, or which have other problems.

Total Power

On the Fly Mapping The on-source auto-correlations or total power readings are taken while the antennas are driven across the sky in a pre-defined pattern. Every so many scans, a reference auto correlation on blank sky may be needed.

Position Switched Mapping Mainly on-source auto-correlation scans. Every so many scans, a reference auto correlation on blank sky (off-scan) is needed. The optimal integration time on off source scans depends on the number of on-source positions and the integration spent time on each one. Occasional amplitude calibrations (auto-correlations on hot and cold loads).

Frequency Switched Mapping On-source auto-correlation scans. The frequency is switched to a slightly different value to obtain the reference scan. Useful if no clean reference field can be found, or if the receivers/atmosphere stability does not allow for the antennas to be switched to a different position. This mode can be combined with on-the-fly mapping.

Special observations Some observing modes will need mode detailed studies:

Pulsar observations

Solar flare tracking.

Project Calibrations We first give calibration modes that may have to be done for a given project. Some will be done once when the project is started or ended; some will be done at regular intervals during a session. This interval will appear as a parameter in the description of observing.

Amplitude calibration Needed for all astronomical observations: Short auto-correlations on blank sky, hot and cold loads. Used to measure system temperatures at the observing frequency. Blank sky to be done as often as needed to follow variations in atmospheric transparency; loads need to be done only if receiver performance varies. One may need to store the results for the source and the phase calibrator separately (for fast switching phase calibration).

Gain Calibration

Phase Independent phase calibration is needed for most interferometric and phased array observations. Can be done frequently if needed, to correct for atmospheric phase fluctuations. One observes a point source (phase calibrator), near the project source. Some projects may need several different calibrators. The observation may be done at the observing frequency or at a different (low) frequency band.

Amplitude In many cases the phase calibrator is strong enough to be useful to calibrate the gain amplitudes. In that case its flux has to be referred to primary flux calibrators (see 'flux calibration' below).

Bandpass calibration Needed for most interferometric observations. One observes a strong point source (calibrator), in the observable sky. The integration time will vary according to the project needs. The same calibrator must be observed both at the observing frequency and at the frequency band used for phase calibrations.

Pointing/Focus calibration Needed for most astronomical observations. Done by cross-correlation scans, either on-the-fly or five-point scans on pointing calibrator, at low frequency, every 10-30 min. Extrapolated pointing parameters from last few pointing calibrations need to be fed back into antenna control. One may also be able to self-calibrate the pointing/focus from the observed source itself, if it is strong enough. At the low frequencies, pointing/focus calibrations may not be necessary.

Flux calibration Used to set up the absolute flux scale, by observing a source of predictable continuum flux (could be in total power or interferometric mode).

Polarization calibration This can probably be derived from phase calibration observations.

Array Calibration We list here observing modes that are used, at time intervals, to calibrate the array itself. The results are useful to all projects.

Pointing calibration session A set of pointing calibrations on several pointing calibrators all across the sky. Needs to be done only after some antennas have been moved. Has to be checked periodically. Occasionally one should measure the relative pointing of the different frequency receivers, by using either a strong planet in total power mode or a strong quasar in interferometric mode.

Baseline calibration session Done by cross-correlation scans on several calibrators all across the sky. Needs to be done only after some antennas have been moved (but some antennas which have not been moved have to be included in the sub-array). One may observe for some time with antennas the positions of which have not been precisely measured, provided that these positions are known for the final reduction.

Delays calibration Needed for most interferometric observations. One observes a strong point source (calibrator), in the observable sky, for a very short time to measure the relative delays to the antennas.

Beam shape calibration Holography measurements on cosmic source, to monitor beam shape and focusing of antennas as a function of elevation.

2.4.2 Sample astronomer input for an observing mode: Single-field Interferometric mapping

```
Section: Field_info
   Name:      IRAS 15194-5115
   System:    EQUATORIAL J2000
   Longitude:  15h 19m 21.90s
   Latitude:  -51d 15' 19.0''
   Beam_size: 0.2''
   Field:     60''
   Map_Cell:  0.03 x 0.03''
   Map_Size:  2048 x 2048
   Weighting: Natural
   Taper:     None

Section: Spectral_info
   N_Bands: 4
   Names:         HCNv0     HCNv1     HCNv2      Continuum
   Frequencies:   xxx GHz   yyy GHz   zzz GHz    350.000 GHz
   Velocities:    -15       -15       -15        0
   Velocity_ref:  LSR       LSR       LSR        OBS
   Widths:        100 km/s  100 km/s  100 km/s   1 GHz
   Resolutions:   0.1 km/s  0.1 km/s  0.1 km/s   5 MHz

Section: Calibration
   Phase_calibrator:             closest  &&  flux > 0.1Jy
   Phase_calibrator_frequency:   90 GHz
   Phase_calibrator_rms:         10 degrees
   Phase_correction:             yes
   Bandpass_calibrator:          best
   Bandpass_phase_rms:           2 degrees
```

Comments:

On these inputs the observing tool should react, e.g. :

- Estimate the size of the array configuration, giving some beam information (ellipticity, sidelobe level)
- Compute first LO and second LO frequencies, converting bandwidths and resolutions, if provided in velocity units, into available correlator modes,
- Estimate output data rate
- Give all sorts of warnings, like the following:

```
OK, with the available setups, you will get resolutions of 0.06, 0.06,
0.06 km/s, 8 MHz, rejection of objects in the skirts of the beam by
about 28 db, an integration time smearing in the outer parts of your
map corresponding to a 6\% loss of peak flux for a point source, and a
data output rate of 37 Gbytes per hour.

128MHz bandwidths will be used, but the upper 5MHz and lower 8MHz will
not be available all year round to allow for Doppler tracking, ...
```

3 Proposal Submission and Handling

Proposal submission has two sides: it should be easy and convenient for observers (including observers unfamiliar with the peculiarities of a given instrument) to write and submit proposals. From the ALMA side, it is important that proposals arrive in a form that makes it easy to store the relevant information and to process them – that includes screening for collisions with other proposals or already observed projects, checking for technical feasibility, and scientific reviewing. We try to combine these two requirements, and take a two step approach: *basic requirements*, which are features that the first version of the proposal preparation tool should have, and *advanced features*, which are convenient, but not necessary at first. However, the software should be written with these requirements in mind, and easily allow adding them.

Proposal submission will occur in two phases: phase one, in which the proposal is submitted for review; and phase two (after acceptance of the proposal), in which the necessary data for writing an observing script are submitted. Phase one requires that, beside the scientific justification, all the relevant technical informations to judge the feasibility of the proposal are present.

3.1 Basic requirements

- Proposals should be submitted electronically.
- All technical input should be stored in a database, and should therefore be computer readable. There should be enough information to provide the proposer (and the reviewer) with a detailed idea of the technical feasibility of the project. This would be easiest if a web-based proposal preparation tool performed certain calculations, i.e. of time required under average conditions, fraction of time available at the site for the specified requirements (e.g. phase stability $<$ xy degrees at a specific frequency), etc.
- The technical data input for proposals should have several layers:
  - Basic interface for non-experts: input in terms of proposal goals
    - angular resolution, largest structure (instead of configurations),
    - source Flux and S/N or RMS (instead of observing time),
    - line frequencies and desired resolution in km/s (instead of correlator setup),
    - image fidelity needed (instead of specifying particular calibration strategies).
  - This translates into observing mode, configurations, observing time, correlator setup, all of which should be checkable (and changeable).
  - Expert interface: one should be able to override the basic mode and specify ALL parameters manually.
- Scientific justification could be postscript or pdf add-on. We don't think this needs to be computer parsable, if all the technically relevant data are available in digital form an electronically submitted anyway. People should be free to use their favorite text processing system to prepare the text and to include figures.
- Basic checking for conflicts against a database of already conducted observations should be done at time of submission to give instant feedback to the proposer. This database should also be accessible for interactive searching prior to proposal writing. People shouldn't be wasting time on proposing things that have been done already.
- The proposal preparation tool should allow storing of intermediate stages to local disks, to enable trying out different parameter settings.
- Proposer should specify what is needed for real-time checking of data quality. In some cases, breakpoints could be defined, after which an observations is stopped and only resumed (possibly in modified form) after examination of the data obtained so far by the proposer.

- The proposal should be detailed enough to judge technical feasibility and time, but does not need to go to all the detail necessary for observations. The observing script should be prepared after acceptance of the proposal with a tool that is a superset of the proposal preparation tool.

3.2 Advanced features

- There should be a simulator (attached to the proposal preparation tool) to produce S/N, dirty beams and, for suitable source models, maps with the desired configurations. There should be a database of real data as input (e.g. if one wants to map a source in HCN(4–3), one could use an already existing HCN(3–2) map as input), and easy-to-construct models by defining basic source components: disks, Gaussians, tori etc. One should aim at getting something like the NCSA Astronomy Digital Image Library for ALMA and linking this to the simulator tool. This depends on the policy for data archives to become publicly accessible.
- Links to databases like SIMBAD or NED should be available to define coordinates. On the basis of a map (from survey databases, or from databases of previous observations) one should be able to define the area to be mapped interactively with a mouse. It would be useful to use existing ALMA (or other) observations as an input data base for defining observations.
- There should be graphical tools for e.g. correlator setup, such as those in use at BIMA and SMA. This should be linked to existing line surveys for a variety of sources, to be able to place correlator units visually at interesting regions, or to simulated spectra based on physical models. This means a line database (such as the JPL catalog) should also be linked to the tool.

4 Dynamic Scheduling

4.1 Requirements

In order to make best use of ALMA we need to change the observing schedule according to the system status and the weather conditions. For example, if the weather is not good enough for the current project to succeed, then we make better use of the array by scheduling a project which can make use of the current conditions. We might call this *level 1* dynamic scheduling, where only the time allocation is changed, but the observing procedures are specified within the previously prepared observing scripts for each project. *Level 2* dynamic scheduling allows system control of the observing procedures (calibration sources, integration times and calibration intervals, etc) depending on the weather and system parameters.

A flexible, dynamic schedule accommodates both **fixed queue** and **interactive** observing. Time critical observations are scheduled by giving them high priority at the appointed time. Interactive observations are made more productive by the use of **breakpoints**; the user takes control of the telescope for a time interval after which control is returned to the scheduling program to determine the best project to observe.

The user may also wish to modify the observations as the results become available from the on-line pipeline analysis and image production. An interactive observer (local or remote) can make these decisions in real time or may wish to dynamically schedule observations based on pre-determined outcomes from the data. For example, if source A is detected, then spend more time on source B, or to change the mosaic pattern or frequency band.

4.2 Implementation

Assimilating some of VLT terminology, we might implement flexible scheduling using observing blocks which are specified by the observer. The observing blocks can be scheduled dynamically, interactively, or in a fixed queue. Generally an observing block should be self contained, and be able to be calibrated independently, although it might use system calibrations such as J/K scales or bandpass. To accommodate level 2 dynamic scheduling, the parameters such as calibration intervals in an observing block could be variables. Even the calibrators might be selected in real time. A versatile scripting language is needed so that observers can control the system to a greater or lesser degree according to their expertise and wishes.

ALMA will be able to measure the phase coherence across the array and find the best calibrator and integration times to use on-the-fly. Such observing parameters, determined in real time, can be used in a dynamically determined schedule. Thus the description of the observations and post-processing of the data must allow for calibrations and observing parameters which are determined at the time of the observations.

Most observing scripts are described in terms of various calibration cycles, mosaics, source switching, frequency switching, etc. A table driven observation is more flexible. The table could specify the 'state' of the instrument for a basic integration cycle. The state includes the source(name, coordinates, velocity etc), polarization, frequency, pointing position, correlator configuration, integration time etc. These state parameters describe the instrument from the point of view of the observer and do not include system parameters such as delay centers etc.

The observer controls the instrument interactively by requesting a particular state, or can build a script which steps or loops through various tables. The tables should be ASCII files which the user can build with more or less help from friendly 'expert' programs which know how to sequence observations. Standard tables can be used for many observations, rather like subroutines (or macros). The tables can be read and edited, and archived as a record of the observations.

An expert dynamic scheduling program (level 2) will become the default mode of observing, unless an expert observer wants to take interactive control of the array.

Dynamic scheduling can be implemented by allowing the system to select some of the 'state' parameters. E.g. instead of specifying the source name for a phase calibrator, the observer could specify

cal=phase RMS 5deg, or some such thing, as the requirement for a phase calibrator, and allow the system to select the best calibrator dynamically. The users retain control of which parameters are set by the system and which they specify directly. A close analogy is taking photographs with a camera which allows either manual or automatic setting for each of its controls. For interactive observations, the human observer can search for the best calibrator, integration times, etc, and then use them. If the observation is running from a script, then the observations can still be optimized and scheduled dynamically with feedback from the data and the weather if desired.

5 Operator Interface

5.1 Operator Model

Location: The Operator can either be located at the site or at the Operational Support Facility (OSF) in San Pedro de Atacama.

Skills: ALMA Operators should have experience with computing and electronics, with a much higher weighting toward the computing skills.

5.2 Operator Responsibilities

The ALMA Operator will be responsible for the smooth operation of the ALMA by performing the following functions:

1. Tends to the daily observing schedule, which may involve an appropriate level of interaction with the dynamic scheduling system. For example, the Operator should review the observing schedule for his shift and contact the appropriate staff regarding possible problems with this schedule.
2. Ensures overall safety at the site by monitoring work schedules of engineers and technicians working at the site. For example, if a telescope is taken out of service for maintenance or repair, the Operator is the main contact point for information regarding the progress of this repair work.
3. Takes care of "routine" operational tasks, which may include the acquisition and analysis of:
   (a) baseline determination measurements which follow antenna reconfiguration;
   (b) pointing measurements;
   (c) flux calibration measurements.

5.3 Operator Interface Software

In order to support additional (remote) monitoring of the array system, the Operator interface software should be made available from any location. A good operational model for such a system would be any of the remote monitoring systems available at the existing millimeter array and single antenna facilities (OVRO, BIMA, NRAO 12 Meter, PdBI). This interface should include:

1. a basic "current array status" display, which includes information such as current pointing position for the array (in a variety of coordinate systems), receiver status, and correlator configuration;
2. a weather display indicating the current site conditions;
3. a video display from cameras at strategic locations on the site (an extreme would be to place a video camera at each telescope pad location);
4. a "current integration" display.

Note that the monitor information within this interface will need to be accessible from everywhere.

6 Data Pipeline

6.1 Goals

The main motivation for processing ALMA data in quasi-real time is to optimize the scientific efficiency of the array. The instrument will be dynamically scheduled (Section 4), so an evaluation of the data quality must be available very soon after data taking, using visibilities in quasi real time, in order to allow switching projects if the current one is not matched to the actual observing conditions. Though readily available information on the array behavior can be obtained by monitoring atmospheric data (such as water content, and it fluctuations), more valuable information can be obtained by monitoring the atmospheric phase itself (using phase calibrators). Finally it is important to be able to determine quite soon if a project's goals are being attained, and for this a first step is naturally to calibrate the data as completely as feasible, and evaluate the quality of that calibration. The instantaneous u,v coverage being reasonably good, pipeline data processing can include not only calibration but also imaging using, if possible, the best solution to the inversion problem: diagnostic tools could eventually select between competitive methods. Another motivation is naturally to make the instrument more accessible to first-time users by producing images in a quasi-automated mode.

The required data quality level will be specified by the astronomers in their proposals, so that the output of the data pipeline can be used to decide when a project is completed. This can be e.g. on the basis of a certain rms noise level at a certain spatial resolution, a dynamic range, the achieved angular resolution or eventually the translation of these into more technical specifications such as rms phase uncertainties on calibrators, bandpass calibration accuracy, tolerance on side lobe levels in the synthesized beam, etc...

The pipeline must be able to process systematically the quasi totality of the measurements obtained with the array in a fully automated procedure. Its output will constitute a data archive with rather homogeneous properties. However their quality will not necessarily be optimal: human intervention will often be required to enhance the quality of the output. These final results should also be archived too, but in an other base of reduced data. Comparison between these two databases can be very useful to optimize the observing and data reduction procedures. These two databases, being often of much more modest size than the raw database, will be much more easily manageable and accessible through the Internet. This should maximize the use of ALMA observations, e.g. for the preparation of new projects by the proposing astronomers or, when they become publicly accessible, for direct scientific use in a different astronomical context than that of the original proposal.

Pipeline data processing will also enhance the efficiency of interactive observing, either by the astronomer if so requested in the proposal, or by the staff during technical time. The data pipeline will not only provide to the astronomer the possibility of adjusting the observing strategy following the results in quasi-real time, but also of running projects more efficiently in focus with their scientific objectives. High level specifications could actually be given during Phase 2 of the proposal submission procedure. To illustrate this, let us consider a proposal with the following requirements: some wide region of the sky must be imaged in the continuum in the 1.2 mm window; this wide field imaging must reach a certain specified rms sensitivity; all compact sources found above 5 σ in that image have to be imaged in pointed mode, one field per source, at higher frequency down to again a certain sensitivity limit such that their spatial morphology can be investigated. Such a high level of specifications implies the need of high level measurement tools as part of the data pipeline such as a source extractor to blindly find sources and their positions; in this case the observing procedure will include several observing modes (mosaicing, pointed observations, multi-frequency) and it will be set dynamically, on the basis of results obtained by the data pipeline during the sequence of observations.

6.2 Requirements

- The pipeline must be able to process all the data coming from the array; it must not constitute in any case a bottle-neck in the data flow.

- The pipeline must operate in fully automated mode for processing from raw data up to deconvolved images. Although this fully automated processing may not lead to optimum quality reduced data, it will constitute the basis for an homogeneous database of processed data.
- The pipeline must also be able to operate in manual mode for technical development, inspections by expert engineers and astronomers on duty but also at the level of the PI/CoI astronomers, for projects which request interactive observations. Requirements (input pipeline parameters) may be given at the stage of the elaboration of the proposals by the astronomers.
- The pipeline must process in quasi real time the observing blocks but, in the imaging stage, it must also be able to include all the observing blocks previously obtained since the project has started. This may mean calibrated data obtained in different array configurations.
- The pipeline must interact with various packages, including the Dynamic Scheduler, the actual observing program, various high-level tools for visualization, and the data transmission (compression/decompression, formatting).
- The pipeline must operate through a readable and comprehensible data reduction script. A high level tool may be available to interactively build such a script, but for automated processing it must be automatically generated from a template, on the basis of the scheduler status and of the collected actual observing procedures. For debugging it should also be able to operate at the command line level. Scripts will be composed of basic components such as, pointing, focus, phase calibration, etc... Some of these components will be activated sequentially but other in parallel (in particular at the mapping stage). It must be able to process in parallel sub-arrays. The observing script will be attached to the data in the archive.
- The pipeline should be run either at or near the telescope, in quasi real time, or later, at places where the official archives are kept (see Section 7).

6.3 Functionalities

6.3.1 Array Calibration

The basic components in the array calibration are the baseline calibration, the pointing, and focus determinations. They must be back fed as soon as possible to be taken into account by the real time system. The operator must have the flexibility to suspend a sequence of activities such as those described in a data reduction procedure and to resume from that state (plus the manual modifications) when the activity was suspended. He or she may change e.g. to an other pointing source or phase calibrator. These actions may have effects on the ongoing observing procedure. It must be possible to modify the level of interaction at any time from fully automated continuous processing to prompts requesting inputs with sensible visible defaults up to prompts with no default (i.e. fully manual).

6.3.2 Calibration of Interferometer Data

The basic components here are phase and amplitude calibrations and their interpolation between calibrator observation. Results (e.g. rms phases and seeing information) must be back fed both to the scheduler and the observing processes. The pipeline must also be able to self calibrate the data when possible.

6.3.3 Calibration of the Total Power measurements

For continuum projects the data pipeline must subtract the atmospheric contribution, in a way that depends of course on the actual observing mode. For line data it must subtract measurements obtained on an OFF position if needed, normalize by gains to scale the data into temperature units; it must also subtract spectral baselines. The pipeline must be able to grid the data for imaging. It must display the results at the various stages. If these total power measurements are obtained by a sub-array while the other antennas are used for the cross correlations for the same target, the calibrations should be able to proceed in parallel such that when imaging both data sets are ready to be combined when the imaging stage begins.

6.3.4 Imaging

The pipeline must produce continuum and/or line images of the calibrated data obtained so far. These images must be visualized, interactively in the case of interactive observations. The pipeline should also be able to compare redundant data (obtained simultaneously or not) to better assess the data quality. It must be possible to feed these interactive measurements back to the scheduler or to the observing process, if relevant. The images should be deconvolved using the most appropriate algorithm; it is desirable to allow several algorithms to compete in case of complex images for which there is no guaranty of a single optimum algorithm. The imaging pipeline must be able to produce images with inclusion of zero and short spacings. In any case it must return information about the robustness of the results in these cases where a unique method is not available.

6.4 Interaction with other Actors

The pipeline interacts with a number of actors in the system. It also plays an important role in the sequence diagram for the array activity. The following actors interact with the data pipeline:

The **user(s)**, such as the PI/CoIs who will most frequently be in their home institute. If they want to go beyond the informations automatically provided by the scheduler, they may inspect the output of the pipeline. Their control on pipeline input parameters may be specified at the proposal stage. Fully interactive observations should indeed be well justified because they have a strong impact on the dynamic scheduling, with a risk of lower overall efficiency.

The **operators**, the **astronomers on duty** and **engineers** close to the array will use the pipeline as one of the tools to control the behavior of the system in general and to check that the automated procedures lead to rational sequential events. Because of their expertise they should have the privilege to modify some of pipelines parameter specifications initially provided by the user.

databases: the array will feed automatically the raw data into a low level database. At the same time it will feed the pipeline with these raw data. The pipeline will output its results into one or two databases of much lower volume, the first one with results obtained systematically by fully automated pipeline procedures, and a second one used when manual interaction has been necessary. Manual interaction should proceed on a parallel system to avoid breaking the fully automated procedure, in order to guarantee the homogeneity of the database obtained with fully automated processing.

Other Software Functions: The pipeline will interact with a number of other software functions, such as:

- Basic analysis tools: visualization, measurement tools, diagnostic tools...
- system analysis: telescope efficiency optimization etc.
- interactive tools used by the operators, engineers and astronomers
- the actual observing program,
- the dynamic scheduler.

7 Archiving

The archive should enable astronomers to easily use data which has already been obtained with ALMA. This includes being able to judge the quality of the data, to produce new images from parts of the uv-data which have not previously been analysed, and to re-analyse data using new data reduction algorithms. The archive should produce data products which are requested by the user. An astronomer unfamiliar with the project should be able to easily find out what data is available in the archive. This might involve successive inquiries to the archive to find: i) what sources have been observed, ii) what spectral lines, velocity coverage and velocity resolution and continuum frequencies and bandwidth was used, iii) inspect the images which were produced by the on-line pipeline, iv) make new images from the archived uv-data. The data in the archive must be easily accessed in a number of ways, e.g. by date, project, source, frequency, spectral line.

Clearly, the archive needs to include the raw uv-data, the calibration data and the images produced by the on-line pipeline. In the case of irreversible on line corrections, such as atmospheric phase correction on time scales shorter than the integration time of the uv-data, it is desirable to keep both the corrected and the uncorrected uv-data (the French solution).

It would be useful to include in the archive the observing scripts, the data reduction scripts used to produce the images in the on-line pipeline, and a description of the scope and goals of the observations. Since new data reduction algorithms will be developed and existing code and algorithms will evolve, the original scripts might not be either usable at a later date, but they will serve as a record of how the data was produced and reduced at the time the data was taken. The description is very useful at later date, when a user unfamiliar with the project can read the description, rather than doing reverse engineering on the observing parameters to deduce what the information the data might reveal. To enable the use of alternative calibration routines, the uv-data within each observing block should be identified by the function which was intended (phase calibrator, target source, coherence calibrator etc.) Generally an observing block should be self contained, and be able to be calibrated independently, although it might use system calibrations e.g. Jy/Kelvin scales and bandpass.

The post-processing of an observing block can be described by the use of objects, and methods. Objects include such things as phase calibrators, amplitude calibrators, coherence calibrators, bandpass calibrators, source(s) and bands (including both multiple observing frequencies and spectral windows). The calibration objects can be selected at the time of the observation, so the detailed data reduction should follow from the actual observations, and not from a previously prepared observing script. The methods specify how to use the phase calibrator objects to apply phase calibration to the target sources, etc. New algorithms might be developed, so that archived data can be re-analysed using using new methods on the previously defined objects within the observing blocks. One could think about inheritance, so that an observing block which does not have a specific bandpass calibrator, could use a default bandpass defined at the system or project level. The designation of the calibration objects by the observer is used by the pipeline post processing and in the calibration of current or archived data. i.e. we don't draw any distinction between data written in close to real time, and data recovered from the archive at a later date; both can be post processed using the chosen methods to deliver the requested data product: "raw" or calibrated uv-data, selected spectral windows and spectral resolution, synthesized images and beams, or deconvolved images.

The software to retrieve the data should check whether they are actually public before forwarding them to astronomers outside the original team of investigators of a given project.

There may be several archives which hold all or subsets of the data. The principal archives should be easily accessed by users in the USA, Europe, and Japan (elsewhere ?). There also needs to be an archive which is easily accessed from the ALMA site, both to buffer the data from the telescope and to provide data for on-line imaging of multiple array configurations.

A Script language

A Suggested Script Language for the ALMA Real-Time System
B.G. Clark
February 2000

This document is intended mainly as a list of the atomic items needed to describe the state of the array. In order to do so in a concrete fashion, it is necessary to include a few control concepts and other programmatic conventions. This are seriously affected by choices made in the course of the implementation of the interface, and this document should not be construed as specifying these things; alternate approaches of equivalent power are certainly acceptable.

I am taking the approach that one describes an observation, and then executes it, rather than the approach that the script should describe a sequence of timed commands, which is also a viable approach, but in my opinion a little less desirable.

In the material below, I find it convenient to speak in terms of macro expansion. (A parameterless procedure is an alternative way to look at the functionality; the two are indistinguishable except in implementation.) In the examples I shall use '`&`' to indicate that the following symbol is a macro, to be replaced by its defining text wherever it is found. I think requiring all macros to be defined at the beginning of the script is a reasonable requirement, and nesting of macros to arbitrary depth useful.

In the examples and definitions below, I shall use the character '`!`' to indicate that the remainder of the line is a comment. (There need to be two types of comments - one which is visible when the script is printed, one which is posted to the attention of the array operator at observe time.)

`{ }` enclose a block of text to be operated on as a unit.

Most information is conveyed by statements of the form

```
<parameter> = <constant>
```

Constant types are:

- strings (in examples below, blank delimited and not containing "hot" characters. We probably need a quoting convention so source names, for example, can contain hot characters.)
- integers (as found by `scanf("%d")`)
- double precision (as found by `scanf("%lf")`)
- angles - we use, for example, `03h19m48.16s`, `41d30'42.1"`
- times - I think we need three formats
  - LST - as, eg `01h10m00s` or `01:10:00`
  - UT - I suggest MJD & fraction, eg `51543.75`
  - Intervals - I suggest `.+3` (three seconds from beginning of scan or block) `.+3:10` (190 seconds from beginning of scan) etc.

Al Wootten's remark that observers may want to detach just a few antennas to run off and do a tipping curve was a possibility that had not occurred to me, and, although it might be done in any of several ways, is elegantly handled by user definable sub-arrays, and has pushed me into defining a script that supports them. Most of the control structures below are derived from that requirement.

There will need to be a carefully thought through specification of the scope of parameter settings (I would model it on the scope of variable names in C), which I will ignore for the moment.

A.1 Special sections

Each file should have a special section describing the proposal and observation block (OB hereafter). Say,

```
$HEADER
    {
    parameters with proposal code, proposal contact person, addresses,
    phone numbers, etc., comments helpful to operators at run time.
    }
```

A special symbol is needed to define a macro. As I mentioned above, I think requiring them to be early in the file is a good idea for readability purposes. The observer's tool should include a library of macros, which can be searched for useful macros to be extracted and put in the script file. A few macro's could be usefully built into the real-time system, for example the list of antennas with a specific set of hardware.

```
$MACRO <macroname>
    { a body of text, not containing a reserved word,
    that appears in the script when the macroname, preceded by a &,
    appears
    }
```

There are two items which I suggest deserve a special status, for different reasons. The correlator setup is very fundamental to the system. It, in particular, determines the output format. I think it is useful for the real-time system to have a list of the correlator setups that it may be required to setup for, and especially for the data reduction programs to have a list of the possible data formats they may have to deal with.

Source descriptions are really tables. For a mosaic, the table is a list of offsets from the reference position, or perhaps preferably, a list of positions to be included in a mosaic organized around the reference position. For a planet, asteroid, or comet, the source description is a table of position versus time. (This differs from the mosaic table in that it may be queried at fractional rows, with the expectation of receiving an interpolation.)

Because these two items are quite different I shall return to them individually below, here remarking that they will look something like this:

```
$SOURCE <sourcename>
    {
    a body of text setting parameters to set up an observation of a source.
    These parameters may appear only in a $SOURCE| block.
    }

$CORRELATOR <setupname>
    {
    a body of text describing the correlator setup.  These parameters may
    appear only in a $CORRELATOR block.
    }

$MESSAGE <messagename> <messageaddress>
    {
    A block to pass information to other programs, especially the pipeline
    reduction program in their native tongue, passed on but not interpreted
    by the real-time system.
    }
```

A.2 Control structures

As mentioned above, it seems desirable to give the astronomer the capability of organizing subarrays of his own. A subarray is defined as a collection of antennas (note: not baselines) whose data is output together. (That is, baselines between antennas in different subarrays are not processed.) Not all of the data may be meaningingful, for instance if some antennas are missing their full complement of baseband digitizers, but the data are output together, and the validity of that data is handled elsewhere.

Each subarray is conceived as being an entity, with the entity scanning the script until it encounters its own description of the sequence of observations, and then proceeding to execute that sequence with the time steps described therein. A facility is provided to move antennas from one subarray to another, here conceived as having the subarray to which the antenna belongs yield it to another.

Each description of the sequence of observations would look something like this:

```
$OBSERVE [<sub-array name>]
    {
    description of observations, with time steps delimited by '/'
    }
```

It seems worthwhile to provide looping at the script level, to keep the size of the script small and readability good. The minimum construct for looping would be to provide two qualifiers, which could be applied either to simple observations (delimited by '/') or blocks (here shown delimited by ...):

```
    REPEAT(<integer number of repeats>)

    UNTIL(<time (either constant or relative)>)
```

In addition, it is occasionally necessary to specify some parameters differently for some antennas. So we have a block of text ignored by or read only by a specified antenna list.

```
    { text not containing '/' } [NOT]FOR ( <list of antennas> )
```

To transfer antennas from one sub-array to another we have the special function

```
    TRANSFER (<sub-array name>, <list of antennas>)
```

An example of the simplest form of `$OBSERVE` block is below. We presume that `setup1` is a macro which organizes frontend switches, LOs, and the correlator, and `sgra*`, `nrao530` and `1749-28` are the names of `$SOURCE` blocks.

```
$OBSERVE
    {
    &setup1
    source = nrao530 / UNTIL (15:30:00)
{
source = sgra* / UNTIL (.+1:20)
source = 1749-28 / UNTIL (.+40)
} REPEAT (60)
    }
```

This runs to source `nrao530`, observes it until 15:30 LST, then starts a set of two minute cycles, with 80 seconds on `sgra*` and 40 on `1749-28`, which continue for the next two hours.

For a more complicated example, suppose the observer wants to assign a few antennas to do a tipping curve for four minutes every 30 minutes. There is a `$SOURCE` block with name tipcurve.

```
$MACRO sgracycle
    {
    source = sgra* / UNTIL (.+1:20)
    source = 1749-28 / UNTIL (.+40)
    }
$MACRO mytipsubarray {2 7 9 17 23}
$OBSERVE main
    {
    &setup1
    source = nrao530 / UNTIL (15:30:00)
{
{&sgracycle} REPEAT(13)
TRANSFER (tipping, &mytipsubarray)
{&sgracycle} REPEAT(2)
} REPEAT(4)
    }

$OBSERVE tipping
    {
    &setup1
    source = tipcurve / UNTIL(.+4:00)
    TRANSFER(main, &mytipsubarray)
    }
```

The first subarray is activated, and does its 13 cycles of source switching. At this point the tipping subarray antennas are transfered to the other subarray, which has been inactive to this point. After the tipping curve is completed, the antennas are transfered back, and the subarray again becomes inactive.

A.3 Describing a source

All the hard decisions in designing a script are whether a function should be provided in the real-time system, or if it is part of the observing preparation program. I suggest below some details for the `$SOURCE` blocks which have a good deal of software in the real-time system. What this complexity gets you is that first, if the operator finds it necessary to suspend operation of the observe file and then restarts it, more sensible results are obtained, and second, a greater variety of observe files may be written in a day-independent fashion (that is, that the file run today may be used again tomorrow without passing through the observe file preparation software again).

`$SOURCE` blocks come in several forms. The first is the simplest and probably the most common.

```
$SOURCE <source name>
    {
    type = STAR
    ra = <J2000 right ascension>
    dec = <J2000 declination>
    calibratortype = <NON | PHASE | AUXILIARY | FLUX | BANDPASS | POINTING>
    [label = <text to be passed to processing programs>]
    [processing = <list of $MESSAGE blocks>]
    }

$SOURCE <source name>
    {
    type = MOSAIC
```

```
    ra = <J2000 reference right ascension for object>
    dec = <J2000 reference declination for object>
    tracking = <SMOOTH | JUMP>  ! antenna pointing at phase reference or
! moving smoothly to next posn during obsn
    [label = <text to be passed to processing programs>]
    [processing = <list of $MESSAGE blocks>]
    centers = <ordered list of triads: ra, dec, dwell-time>
    }
```

This type of source block declares a table of locations, which are then accessed by an explicit or hidden variable, such that each time the source is visited, a new entry in the table is pulled out and used for the phase tracking center. Jumps, based on the durations specified here need to be synchronized so that they begin precisely at the end of correlator integrations. (As should any change of source.) I have suggested positions, rather than offsets, because I always have trouble remembering whether they should be added to the reference position to the the instant position, or vice versa.

```
$SOURCE <source name> { type = PLANET [label = <text to be passed to
    processing programs>] [processing = <list of $MESSAGE blocks>]
    position = < ordered list of tetrads: UTTime, geocentric ra, dec,
    distance > }


$SOURCE <source name>
    {
    type = AZEL
    [processing = <pointer(s) to information used by postobserving processing>]
    [processing = <list of $MESSAGE blocks>]
    centers = < ordered list of triads: elevation, azimuth, dwell-time >
    }
```

This is used to position antennas to a given AZEL location, or set of locations.

The set of locations is intended for the use of this for use to generate data for a tipping curve, or for holography with a transmitter. The other common uses for az-el coordinates - parking the antenna for technician access or putting the array into stow position - will likely have convenient commands in the operator and technician interfaces, and are less likely to be done through the observing script.

A.4 Describing the correlator setup

I am suggesting here that parameters be specified on a baseband pair basis. This forcloses, for instance, different spectral resolutions in the two basebands. One can think of cases in which that is useful, but they do not seem to me to justify the complexity in the real-time system. The suffix X indicates which of the four baseband pairs is indicated. I have no particular opinion about whether this should be A, B, C, D, or 1, 2, 3, 4 or [1], [2], [3], [4] or whatever.

```
decimationX = <1 | 2 | 4 | 8 | 16 | 32>
   ! Sets the total bandwidth analysed

crosscorrelationX = <YES | NO>
   ! NO is the multiple single dish mode

polarizationX = <YES | NO>
   ! four vs two products per BB pair (even if, due to switches, some are silly)
```

```
integrationX = <integration time in seconds>
   ! the u,v FITS for the VLBA correlator requires the same integration time
   ! in all basebands, which I here propose to relax.
   ! having a baseline specific integration time is a big complication in
   ! the real-time system (and elsewhere also), which I do not believe is
   ! justified by the reduction in output rate one would realize

nmbrlagsX = <16 | 32 | 64 | ... | 2048>
   ! one may want to process fewer than the full number of lags the hardware
   ! provides, to exchange spectral resolution for data volume or data rate.

archiveX = <list of channel numbers to be passed to archiving system>
   ! We probably need a mechanism to discard the ends of the spectra
   ! to get data output rates and volumes down.  This is the most
   ! general way of describing this, and pretty easy to implement.
   ! However, I can imagine the programmer of a 'filler' program looking
   ! like the famous Muench painting when asked to implement this in all
   ! its generality.  In practice, one would presumably have a small number
   ! of spectral windows.  But having the full generality in the real-time
   ! system does not seem harmful to me.
```

A.5 Describing a set of observations

Each subarray has an associated block which describes the sequence of setups and sources as a function of time.

For use by a "control panel" type front end, the real-time system should be arranged to accept all blocks at the front of the file, and then prepare itself to receive the `$OBSERVE` block a record at a time, and to skip to the new record when it arrives.

I group things by the devices to which they apply for reading convenience - the groupings have no functional significance.

A.5.1 Antenna

The real-time system would have a database of the parameters of the antenna pointing model - things like azimuth axis tilt, axis perpendicularity, encoder centering, azimuth encoder zero offset, and, for each feed, the collimation offsets. It is a little unclear to me what of these need to be brought out to script level. It is clear that the collimation offsets need to be, and they are the only ones shown here. These script offsets are to be added to the ones from the internal database. If, as seems likely, the initial pointing model needs work, (eg having temperature dependent coefficients, or a higher order harmonic to describe bearing runout), I would incline to changing the internal database directly, rather than bringing the quantities out to the script for playing around purposes.

```
azcollimation = <collimation offset in arcminutes>

elcollimation = <collimation offset in arcminutes>

refractivity = <refractivity in N units>
   ! if refractivity is 0, calculate it from on-line weather instruments

wrap = <R | L | N>
```

```
   ! If there is a choice of azimuth wrap, choose CW, CCW, Nearest, resp.

reference = <NO | * | receivercode>
   ! Apply reference pointing collimation errors from named band
   ! (from current band if *)
```

A.5.2 Subreflector

Again, there would be a database of the subreflector model parameters, containing for each antenna and receiver, the required focus setting, focus vs sin(elevation) coefficient, focus vs temperature coefficient, x,y focus position, x vs cos(elevation) coefficient. Again, quantities in the script are added to those in the database.

```
focus = <focus offset in mm>

xfocus = <x direction translation in mm>

yfocus = <y direction translation in mm>

temperature = <temperature, C>
   ! if temperature is 0, use on-line outside thermometer

calibration = <ON | OFF | SWITCHING>
   ! Subreflector center calibration device
   ! As I understand it, calibration scheme still undecided.
   ! Might require a few more codes (ie thermal calib might need ONHOT, ONCOOL)

! parameters need to be added for control of nutating subreflector if one
! is eventually decided on; I understand one is not in the baseline plan.
```

A.5.3 Receiver and IFs

```
receiver = <receivercode>
   ! this selects what collimation parameters to use, and specifies
   ! which receiver the LO power is routed to.  (My understanding is
   ! that only one receiver at a time receives LO power.)

firstlo = <first LO frequency in GHz>

phasemod1 = <NONE | SINE | WALSH >
   ! specifying the functional form of the phase modulation imposed on
   ! the first Local Oscillator.  May also want to specify the order of
   ! the Walsh function, although letting it be implied by the number of
   ! available antennas is probably OK.  The suffix 1 indicates that it
   ! is applied to the first LO.
   ! I include this technical detail in the script because I'm quite
   ! unsure that we can a priori make the right choice that will work
   ! for everything.

phasemodrate1 = <modulation rate in Hz>
   ! Basic modulation rate (repetition of the Walsh functions, or lowest
   ! harmonic for the sine waves).  Each antenna will have a sequency or
```

```
   ! harmonic number of it's own.
   ! If WALSH is selected, there will be interesting constraints on permitted
   ! rates

firstlodoppler = <OFF | SET | SETREF | TRACK | TRACKREF >
   ! SET = set LO at scan beginnings
   ! TRACK = track continuously during scan
   ! REF = set LO at mosaic center; otherwise at mosaic offsets
   ! If not OFF, firstlo frequency is barycentric

dopsource = <source descriptor | CURRENT>
   ! Use given source descriptor to calculate doppler, rather than current
   ! source (to observe bandpass calibrator at same frequency as main field)
```

A.5.4 Basebands

There are eight basebands, which must be operated in pairs, because there are only four copies of the second LO module. These pairs are usually to be used to select polarization pairs, but more exotic configurations are possible. Because of this pairing, it is convenient to describe basebands in terms of baseband pairs. Note that, in electronic terms, we are talking about pairing two baseband converters from **different** downconverter blocks. It's implementer's choice whether you want subscripts or a separate name for each baseband command. Issue bypassed here by writing, eg, `BBselectXY`, wherein X specifies the member of the pair, and Y specifies the pair. The above could mean `BBselectA1` or `BBselectA[1]` or `BBselect[1,1]` or `BBselect[0,0]`, depending on how you want to name or number things.

In particular, the `BBselectXY` command is a problem, in that not all combinations are possible (the switch structure is flexible, but not infinitely so), and not all possible combinations are useful (eg selecting different first LO sidebands for the two members of a pair). An alternative approach would be tied more closely to the structure of the switches, so that it would not be possible to specify an impossible or even a non-useful combination.

```
BBselectXY = <OFF | UV | UH | LV | LH>
   ! Selects the sideband and polarization of the signal fed to the named
   ! converter.  (I suppose R,L should be permitted as well as V,H, in case
   ! somebody has a smart idea about a quarter wave plate.)

secondloY = <second LO frequency in GHz>

phasemod2Y = <PASS | WALSH | SINE | NONE>
   ! for some purposes it may be desirable to have the second LO remove
   ! the phase modulation of the first LO and impose a different one of
   ! its own.  (In particular, with the photonic LO scheme, it may well
   ! be convenient to use sines for sideband suppression and Walsh functions
   ! to cancel sampler offsets).  PASS lets the first LO modulation pass
   ! unchanged; NONE removes it, and does not impose a different one.

phasemodrate2Y = <modulation rate in Hz>

phaseoffsetY = <arbitrary phase in degrees to be added to 2nd LO phase>

secondlodopplerY = <OFF | SET | SETREF | TRACK | TRACKREF >
   ! SET = set LO at scan beginnings
   ! TRACK = track continuously during scan
   ! REF = set LO at mosaic center; otherwise at mosaic offsets
```

```
   ! If not OFF, secondloY frequency is barycentric
   ! dopsource (above) would also apply here
   ! if first LO doppler is not OFF, this only applies difference


phasecalibY = <OFF | AUTO | CALIB | POWER | WVR>
   ! source of phase calibration information
   ! AUTO indicates using current observations to phase up immediately
   ! (called Autophasing at the VLA) - principal use for VLBI
   ! CALIB extracts phases from last calibrator scan, again principally for
   ! VLBI
   ! POWER uses power level from receiver in use with a coefficient to convert
   ! to phase due to water vapor
   ! WVR uses a special water vapor radiometer being considered
   ! More codes, and combinations of the above, may be needed

deltaDelayXY = <Delay in nanoseconds to add to system offset value>

FIRfilterXY = <Code for set of filter tapweights>
```

A.5.5 Correlator

It is insufferably confusing unless all baselines in a subarray have the same values of this set of parameters. They are therefore forbidden to occur inside a `{...} for(..)` block.

There would be something to be said to make this stuff into a special `$CORRELATOR` block, like `$SOURCE`. For one thing, it would make it very natural to supply ahead of time to a 'filler' program a list of all settups to be expected, which would make it easier to write than having to have it be ready for anything at all times.

A.5.6 Real-time data processing

Again, forbidden in for-blocks.

```
fftY = <YES | NO>
   ! to get delay functions instead of spectra for special purposes

digitizationCorrectionY = <YES | NO>
   ! suppress digitizatin correction for higher output rate

statisticsCorrectionY = <YES | NO>
   ! apply correction for non-ideal digitizer levels

smoothing = <NONE | HANNING>

tsysCorrectionY = <YES | NO>
   ! apply correction derived from calibration hardware

scaleXY = <arbitrary number to multiply output by>

corrphasecalibY = <OFF | AUTO | CALIB | POWER | WVR>
   ! Apply phase calibration after correlation instead of to 2nd LO
```

```
postintegrationY = <number of LTA integrations to be averaged before
                   archiveing>
   ! Only useful if corrphasecalibY is not OFF

archivebothY = <YES | NO>
   ! Only useful if postintegration is specified
```