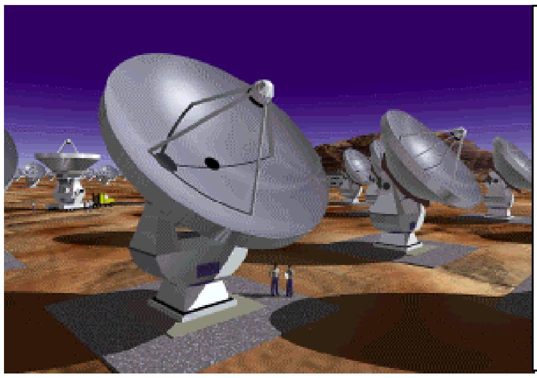


Atacama Large Millimeter Array	Alma-NNNNN Issue: 1.0 2000-09-19 *MEMO* **Gianluca Chiozzi**

ACS and AMS Kitt Peak 2000 Test

MEMO

Gianluca Chiozzi, Birger Gustafsson, Robert Karban

ESO

Ron Heald

NRAO

Alain Perrigouard

IRAM

Doc Number	Alma-NNNNN
Issue	1.0
Date	2001-02-07
Owner	Gianluca Chiozzi (gchiozzi@eso.org)

Approved by: Date: Signature:

Change Record

ISSUE	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS
1.0/prep1	2000-09-19	G.Chiozzi		First draft before K.Peak test
1.0/prep2	2001-01-26	G.Chiozzi		Update after K.Peak
1.0/prep3	2001-02-05	R.Heald		Entire document revised
1.0	2001-02-07	G.Chiozzi		Final release

Table of Contents

1 SUMMARY

The Kitt Peak 2000 Test was a joint project of ESO (G. Chiozzi, B. Gustafsson, R. Karban, B. Jeram, P. Sivera), IRAM (A.Perrigouard) and NRAO (R. Heald), with the collaboration of R. Lemke (AIRUBMP).

The project goal was to evaluate the ALMA Common Software (ACS) 0.0 prototype as a baseline for the ALMA antennas, with particular emphasis on the Test Interferometer.

In parallel to the development of the ACS prototype, A.Perrigouard (IRAM) and R. Heald (NRAO) developed a prototype Antenna Mount Software (AMS) based on the ACS prototype.

The ACS and AMS software components were integrated in Socorro during November 2000, and the whole system was installed on the 12-meter antenna (12m) at Kitt Peak, Arizona, for the final test during the period 1 to 8 December 2000.

The basic idea of the tests was to repeat as much as possible of the Kitt Peak 1999 test. That test was based on the ESO Common Software (ECS) and the VLT Telescope Control System (TCS). The 2000 test would substitute the prototype ACS for the ECS, and the prototype AMS for the VLT TCS.

For the tests the AMS was required to provide the same interface as the VLT TCS so that components of the 1999 test software could be used together with the new software to allow making a direct comparison.

This document describes the various aspects of the project, defining the scope, architecture, milestones, deliverables, test plan, test setup, planning and responsibilities.

The document is concluded with a report from participants on the suitability of ACS for the ALMA project.

The document is divided into chapters. Chapter **2** is an introduction. Chapter **3** defines the requirements for the tests. In chapter **4** the standard 12m hardware and software is described. Chapter **5** describes the architecture of the Kitt Peak 2000 TCS. Chapters **6** and **7** describe the Kitt Peak 2000 system configuration. Chapter **8** is the actual test report.

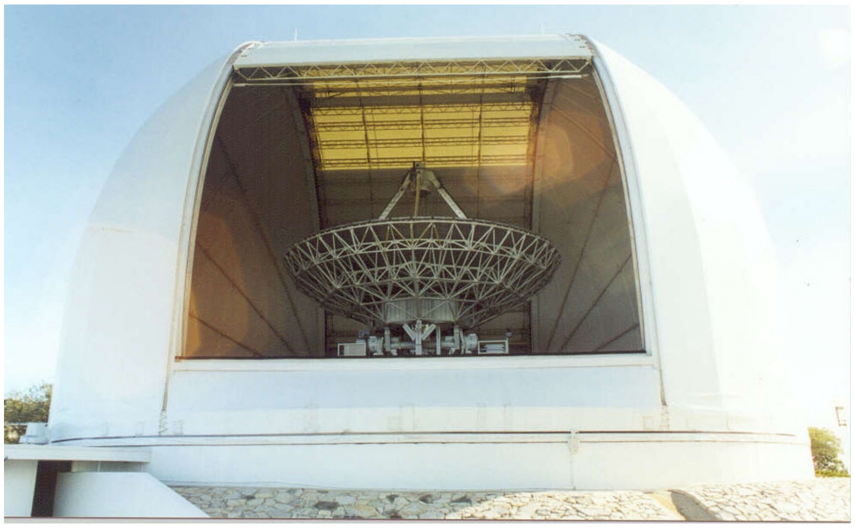


Figure 1 - The Kitt Peak 12m Antenna

2 INTRODUCTION

2.1 Scope

It is assumed that the reader is familiar with Telescope Control Systems (TCS) in general. Knowledge of ALMA Common Software, the VLT TCS and the Kitt Peak 12-meter antenna is an advantage. Furthermore the document uses core UML notation to graphically enhance the understanding of the system under design.

This document is not only meant as a report of the Kitt Peak 2000 test, but also as an introductory reading to the AMS software and as an example of test procedure/integration procedure.

2.2 Glossary

This document employs several terms, abbreviations and acronyms to refer concisely to an item, after it has been introduced.

The main reference for such definitions is the online ALMA Software Glossary (http://www.mma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm).

What follow are definitions specific for this document and not already part of the ALMA Software Glossary.

12m	**The 12-meter antenna.**
AIRUB	**Astronomisches Institut der Ruhr-Universitaet Bochum**
AMS	**Antenna Mount Software developed using ACS.**
ECS	**ESO Common Software. The software used for the VLT project.**
ESO	**European Southern Observatory**
IRAM	**Institut de RadioAstronomie Millimétrique**
KPAM	**Kitt Peak Antenna Mount API. Low-level C API to the 12-meter antenna control electronics.**
LCU	**Local Control Unit. A real-time computer running the VxWorks OS.**
NRAO	**National Radio Astronomy Observatory**
TCS	**Telescope Control Software.**
VLT	**Very Large Telescope. An optical instrument designed and built by ESO.**

2.3 Stylistic Conventions

The following styles are used:

bold
in the text, for commands, file names, pre/suffixes as they have to be typed. Also used to highlight words.

italic
in the text, for parts that have to be substituted with the real content before typing. Also used to highlight words.

`teletype`
for examples.

<name>
in the examples, for parts that have to be substituted with the real content before typing.

2.4 Naming Conventions

All Software Modules that are created during this project has the prefix **kp**. **kp** stands for **K**itt **P**eak.

This applies to all new modules that were created, as well as to VLT software modules that were modified for the purposes of the test, e.g. **kp**trk, **kp**trkws, **kp**prs, etc.

An exception is the **ams** module that is more general and forms the beginning core for software to be used in the ALMA Test Interferometer.

3 REQUIREMENTS

In the following paragraphs we will define the basic requirements that should be fulfilled at the end of this project.

3.1 ACS basic features

The ACS basic features shall be integrated with the Kitt Peak software. These features are:

- **Distributed Object / Properties and Characteristics design pattern**
- **Logging**
- **Error System**
- **Testing**
- **Graphical User Interfaces**
- **CMM Archive**
- **Software Engineering practices (i.e. directory structure, Makefile, build procedures)**

The goal is to build on top of the Antenna Motion API (the kpam module) a layer that provides all relevant functionality using the ACS features. It shall not be necessary to modify the original Antenna Motion API to achieve this.

3.2 Optical tracking

The 12m has a small optical telescope mounted at its prime focus that is normally used to deliver a first pointing model for tracking radio sources. A CCD is connected to the optical telescope to deliver the images to a display in the control room. A reference point is marked near the center of the display.

The 12m shall be able to track optically. Optical tracking is defined as, using the optical telescope, keeping the target at the reference point of the display.

The pointing and short-term tracking errors shall not be worse than the currently achieved values, i.e. a pointing accuracy of about 5 arc-second on the sky.

To achieve this, the AMS shall be integrated with the VLT pointing model software, already tested in 1999. The AMS consists of the needed drivers and servo-loop software that drives the axes of the antenna.

The VLT pointing model software is used to point the antenna to suitable targets in the sky, to evaluate a pointing model, and to install it on the antenna.

The goal is to demonstrate the ease of integration of the ACS with completely different hardware and software.

3.3 ACS tutoring

In the course of this demonstration project the software group developing AMS shall become familiar with ACS software, procedures, standards and software engineering practices. At the end of the project they shall be knowledgeable of the ACS prototype major parts, i.e. ACS development environment, configuration management, testing, LCU applications and GUIs.

4 *STANDARD* 12-METER TCS

This chapter describes the normal 12m TCS. The system's primary responsibilities are processing commands received via a network socket connection from the control workstation and to return replies, performing coordinate conversions of input positions, closing the antenna position loop, and recording the antenna positions during On-The-Fly (OTF) observations.

4.1 Hardware

The following hardware boards are located in the TCS LCU (VME chassis) that is commonly called "Tracker":

One CPU board Motorola MVME147
The Motorola MVME147 CPU board (based on the Motorola 68020 processor) is used to run the tracking software.

One Matrix Corp. MD-DAADIO-8NAP DAC card
This card is given two 16-bit position errors, one for each axis. It converts each error value to a +10v to -10v signal that is fed directly to the axis servo motor amplifier. A VxWorks driver for the card was provided by Matrix

Two VME Microsystems VMIVME-2511 parallel input cards
Each of these cards reads the 24-bit position encoder value for a single axis. 12m personnel wrote a VxWorks driver for the card.

One K-SYSTEMS IRIG-B time base generator card
This board provides UTC down to tens of microseconds. The board must be provided with an IRIG-B signal if it is to provide accurate time. Without the signal the board runs "open loop" on an internal oscillator. Notice the board is unaware of the year and it must be obtained from another source. 12m personnel wrote a VxWorks driver for the card.

One SAM bus card
This card supports the SAM bus that propagates certain signals in real-time through out the 12m system. Each device that needs this real-time information is connected to the bus. The bus consists of several bits. One bit of particular interest is used to indicate whether the antenna is on or off source.

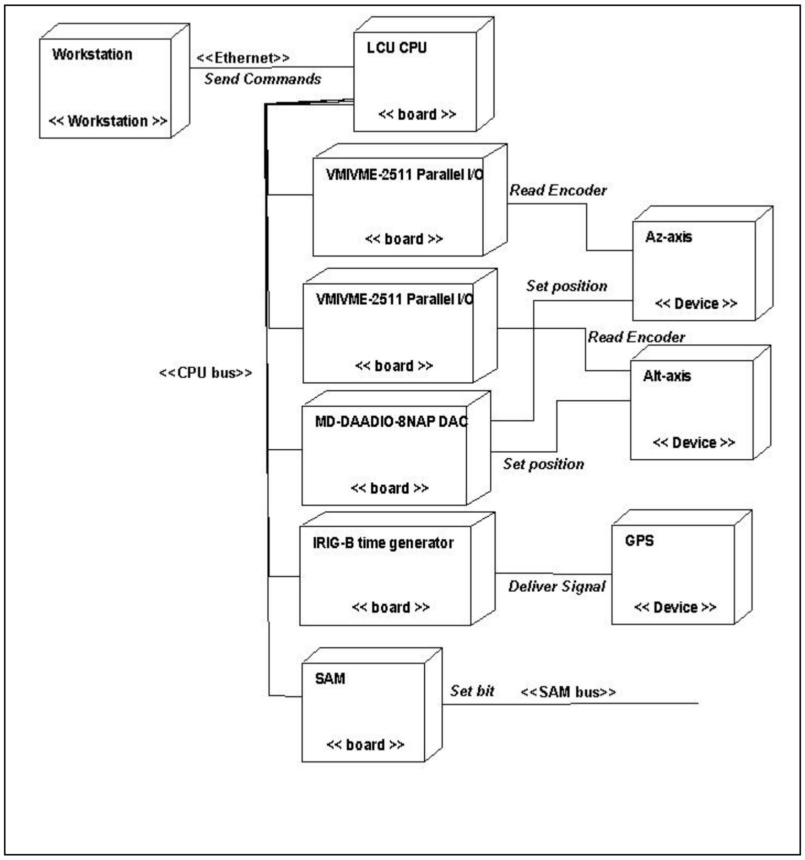


Figure 2 - 12m Tracking Hardware

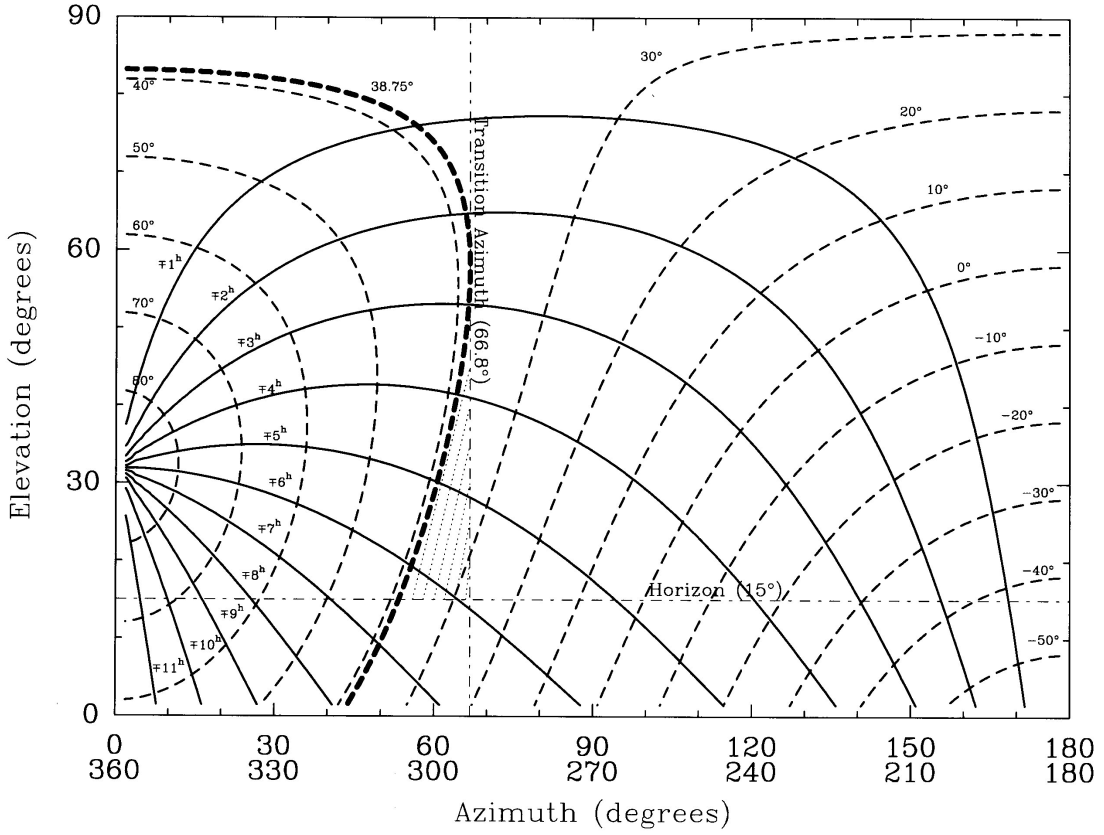
4.2 Software

Coordinate System

The 12m conventions for azimuth position are 0° is north and 90° is east. The Park position for the antenna is elevation 90.0° and azimuth 0.0°.

Azimuth Limits

The azimuth axis cannot move through the antenna mechanical wrap located at about 66°. When this position is reached the TCS must reverse direction to move the antenna to the other side of the limit where tracking can resume. While the axis moves to the new positions, the SAM bus bit "Off Source" is raised and data taking is stopped. The wrap is shown in Figure 3 as the heavy dashed line.



Local hour angle and declination versus azimuth and elevation for the 12m. The hatched region indicates the range of source declination where the telescope will undergo a 360° azimuth transition while tracking a rising source.

Figure 3 - Local hour angle and declination versus azimuth and elevation at the 12m

Position Control

The architecture of the TCS consists of the following major parts:

CommandInterpreter: a task to receive commands via a socket connection

Tracker: a task that does all necessary astronomical calculations

PositionLoop: a task that keeps the antenna axes under position control

The Tracker task provides periodically (0.1Hz – 0.2Hz) coefficients to the position loop that describe the azimuth and elevation motion, plus a timestamp. The coefficients determine the position, velocity, and acceleration at any particular moment for the azimuth and elevation axes.

The Position Loop task calculates the target position 100 times a second (every 10 milliseconds). It does this using the coefficients from the Tracker task and time retrieved from the **IRIG-B** board. It then calculates the difference between the current position (read from the **VMIVME-2511** board) and the desired position. The resulting position error is passed to the **DAADIO** board where it is converted into an analog signal and fed to the servo motor amplifier. The Position Loop implements only pure positioning without PI feedback.

The Position Loop task also monitors the actual position error and raises a SAM bus bit if the tolerance (a configurable value) is passed. This action notifies other applications if the target has been lost.

5 ARCHITECTURE OF KITT PEAK 2000 TCS

This chapter elaborates the architecture chosen for the test system. Notice all documentation is kept under configuration control in the module **kpdoc,** where you will also find this document.

The system includes both the new TCS based on ACS, and the software used for the 1999 test based on ECS and the VLT TCS. It is possible to switch dynamically between the two systems via software commands.

The simultaneous availability of the two systems allows to:

- Use the KP1999 software to verify the status of the 12m compared to last year.
- Compare the performance of the two systems.
- Use components of the KP 1999 software to enhance the test of the KP 2000 system. This allows performing tests otherwise not possible with only the software developed for this test. In particular, the VLT pointing model software is used to acquire and evaluate the pointing model to be installed in the 2000 system. The VLT software drives the ACS TCS using normal ESO Common Software commands through a protocol converter.

The complete system is built and installed using the **kptcsBUILD** package (version 1.34 is used to build the final system). This package uses the **pkgin** installation tool that is part of ACS 0.0.

The file `kptcsBUILD/config/kptcsBUILDINSTALL.cfg` contains the complete list of modules with the respective archive versions and is included in an appendix.

The **kptcsBUILD** module contains also the ECS real time database (environment definitions) and the ACS configuration database for both workstation and LCU.

5.1 ACS Telescope Control Packages

The following package diagram shows the architecture of the TCS based on ACS.

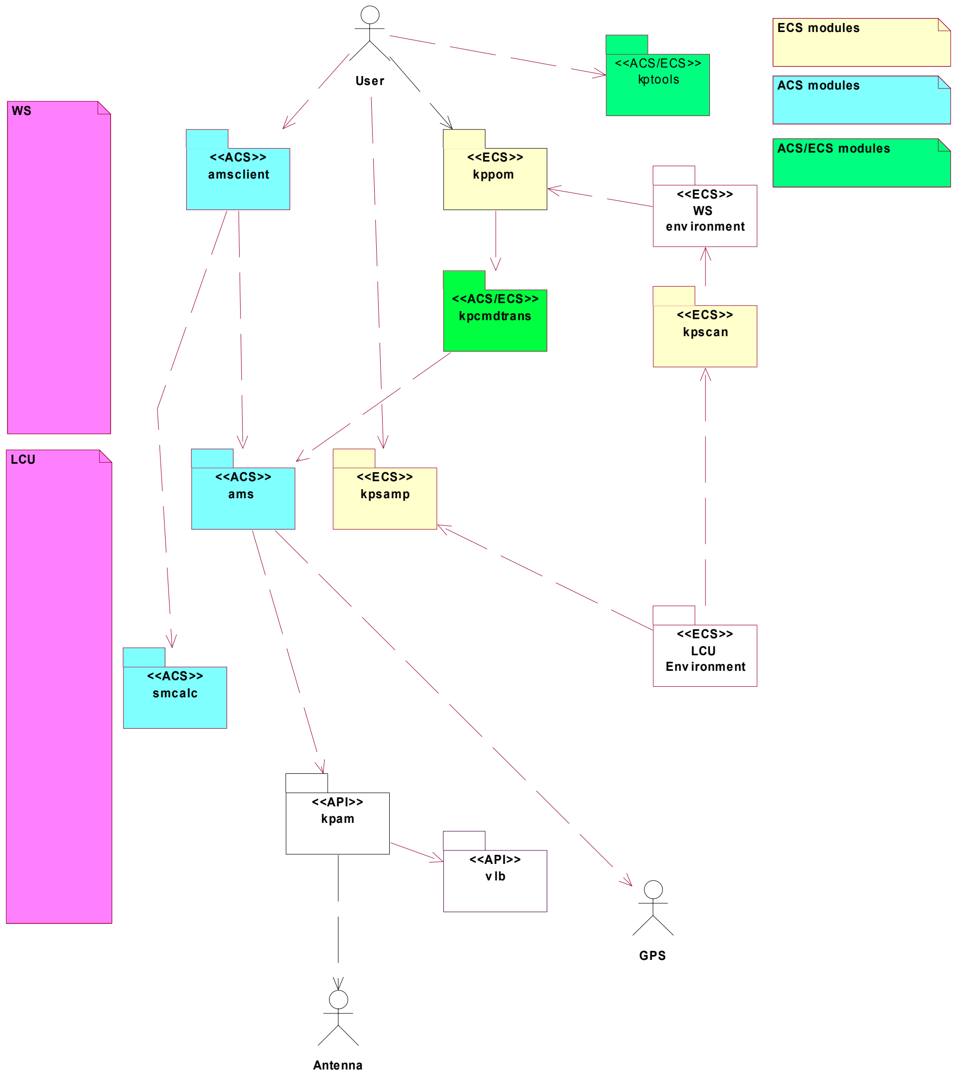


Figure 4 - ACS TCS package diagram

Each package corresponds to one software module kept under configuration control in the CMM archive (except for the LCU and workstation environments that are represented here to describe the system but that are not real packages).

There are 4 types of packages:

1) **<<ACS>>** packages are purely developed using ACS 0.0 and are main subject of the test.

2) **<<ECS>>** packages are "*legacy*" packages, developed for the 1999 Kitt Peak test using the ECS. These packages are re-used here to provide features that could not be implemented with ACS 0.0 because of lack of time or because the ACS prototype did not provide needed features.

3) **<<ACS/ECS>>** packages are "*glue*" packages specifically developed to provide an interface between <<ACS>> packages and <<ECS>> packages and hide from <<ECS>> applications the fact that they are interacting with <<ACS>> applications.

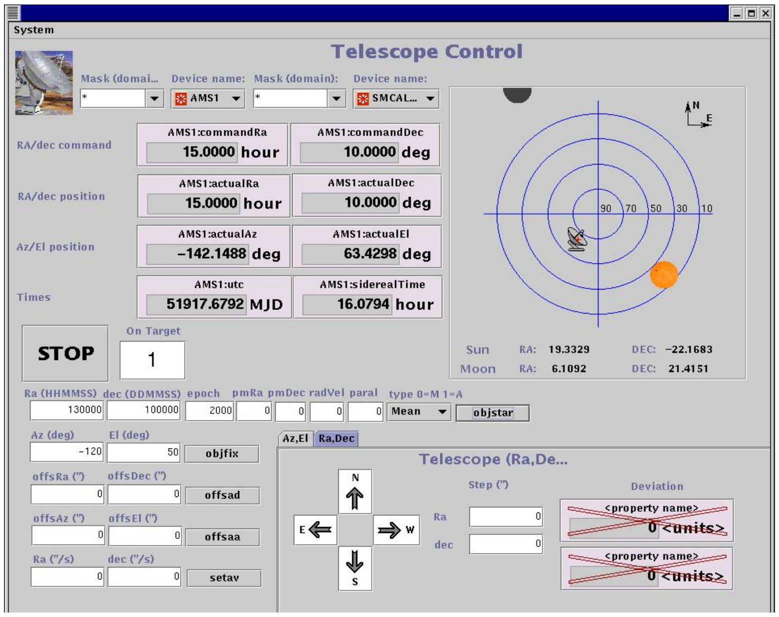
4) **Other packages** do not fall in any of the previous categories.

Packages to the right of the box marked "WS" run on a Linux workstation, and packages to the right of the box marked "LCU" normally run on a LCU, but can be run on a Linux workstation for testing and simulation purposes.

In the following paragraphs we describe the packages and how they interact.

<<ACS>> amsclient

This package provides the user interface to the TCS, and is used by the operators to interact with the system. It uses the ACS user interface packages and tools (specifically the Abeans Java classes), and was developed with Visual Age for Java on Linux.



AMS1:commandRa 15.0000 hour
AMS1:commandDec 10.0000 deg
AMS1:actualRa 15.0000 hour
AMS1:actualDec 10.0000 deg
AMS1:actualAz -142.1488 deg
AMS1:actualEl 63.4298 deg
AMS1:utc 51917.6792 MJD
AMS1:siderealTime 16.0794 hour

On Target 1

Sun RA: 19.3329 DEC: -22.1683
Moon RA: 6.1092 DEC: 21.4151

Ra (HHMMSS)	dec (DDMMSS)	epoch	pmRa	pmDec	radVel	paral	type 0=M 1=A
130000	100000	2000	0	0	0	0	Mean

Az (deg) -120, El (deg) 50
offsRa (") 0, offsDec (") 0
offsAz (") 0, offsEl (") 0
Ra ("/s) 0, dec ("/s) 0

Step (") Ra 0, dec 0

Figure 5 - AMS User Interface

<<ACS/ECS>> kptools

This package contains a few simple startup/shutdown scripts that are used to activate the new ACS-based TCS or the old ECS-based TCS.

<<ACS>> ams

This package controls the antenna and normally runs on the LCU. The package implements the AMS Device. It receives target coordinates from the user interface, performs all needed astronomical calculations (including pointing model corrections), and sends reference positions to kpam.

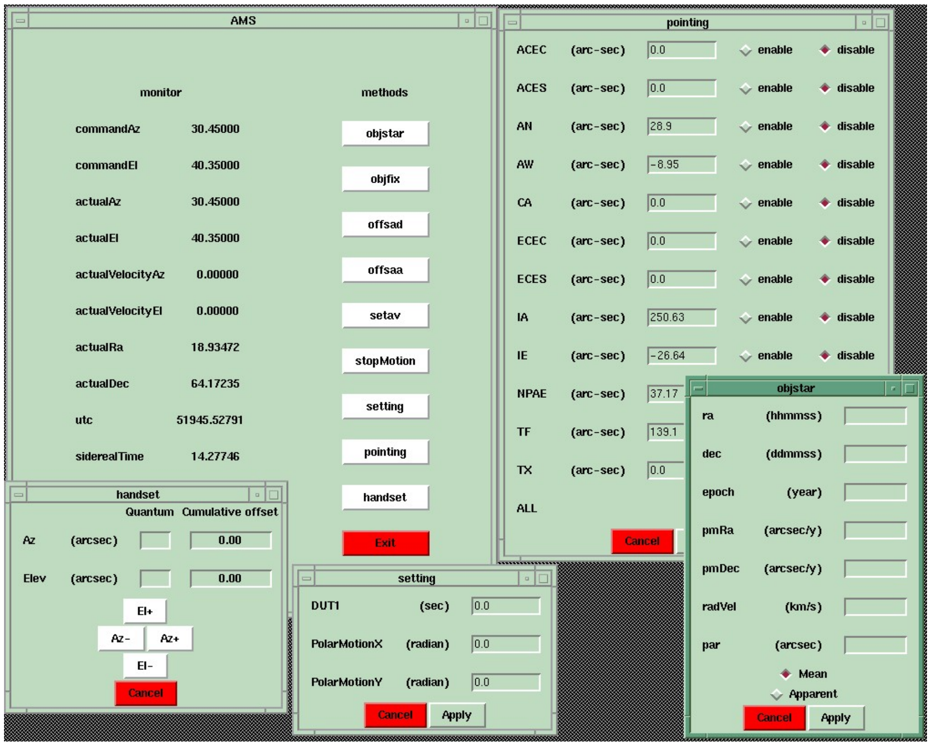
A foreground loop performs in real time the conversion from astronomical coordinates to positions and velocities in horizon coordinates. A background loop is responsible for calculating slowly varying, but computationally intensive parameters, at a low frequency.

All astronomical calculations are based on the **slalib** package, from Pat Wallace at Starlink. The implementation of the pointing modeling corrections is based on the terms and algorithms described in the TPOINT software, also from Pat Wallace.

The **ams** package was explicitly designed to provide as much as possible the same interface as the **kptrk** package that plays the same role in the TCS of the 1999 test.

The commands and properties provided by ams are listed in the IDL in the appendix.

The **ams** package also includes a Tcl/Tk client (Figure 6) that allows access from a user interface to all IDL interfaces of the AMS Device. This client was developed as an engineering interface, and to demonstrate the capabilities of a Tcl/Tk client in the ACS environment.



monitor	
commandAz	30.45000
commandEl	40.35000
actualAz	30.45000
actualEl	40.35000
actualVelocityAz	0.00000
actualVelocityEl	0.00000
actualRa	18.93472
actualDec	64.17235
utc	51945.52791
siderealTime	14.27746

ACEC	(arc-sec)	0.0	enable [✓ disable]
ACES	(arc-sec)	0.0	enable [✓ disable]
AN	(arc-sec)	28.9	enable [✓ disable]
AW	(arc-sec)	-8.95	enable [✓ disable]
CA	(arc-sec)	0.0	enable [✓ disable]
ECEC	(arc-sec)	0.0	enable [✓ disable]
ECES	(arc-sec)	0.0	enable [✓ disable]
IA	(arc-sec)	250.63	enable [✓ disable]
IE	(arc-sec)	-26.64	enable [✓ disable]
NPAE	(arc-sec)	37.17	
TF	(arc-sec)	139.1	
TX	(arc-sec)	0.0	

		Quantum	Cumulative offset
Az	(arcsec)		0.00
Elev	(arcsec)		0.00

DUT1	(sec)	0.0
PolarMotionX	(radian)	0.0
PolarMotionY	(radian)	0.0

ra	(hhmmss)	
dec	(ddmmss)	
epoch	(year)	
pmRa	(arcsec/y)	
pmDec	(arcsec/y)	
radVel	(km/s)	
par	(arcsec)	

[✓ Mean] Apparent

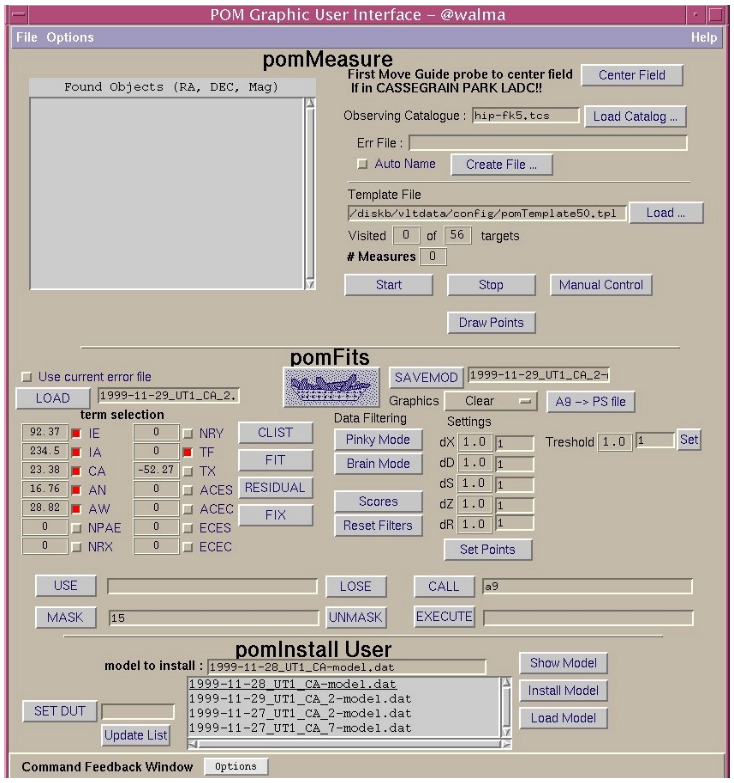
Figure 6 - TCL/TK AMS Engineering User Interface

<<ACS>> smcalc

This package implements the SMCALC Device that is responsible for periodically calculating the position of sun and moon. It runs normally on the LCU, and its results are used only for display purposes.

<<ECS>> kppom

This package is the pointing modeling software used on the VLT and for the 1999 tests. It actually consists of the **kppom** CMM module and of a set of user interfaces that are part of the **kpgui** CMM module.



First Move Guide probe to center field
If in CASSEGRAIN PARK LADC!!

Found Objects (RA, DEC, Mag)

Observing Catalogue : hip-fk5.tcs

Err File :

Auto Name

Template File /diskb/vltdata/config/pomTemplate50.tpl

Visited 0 of 56 targets

Measures 0

Use current error file

1999-11-29_UT1_CA_2.

1999-11-29_UT1_CA_2-

Graphics Clear

term selection

92.37	🟥 IE	0	NRY
234.5	🟥 IA	0	🟥 TF
23.38	🟥 CA	-52.27	TX
16.76	🟥 AN	0	ACES
28.82	🟥 AW	0	ACEC
0	NPAE	0	ECES
0	NRX	0	ECEC

Data Filtering

Settings

dX	1.0	1
dD	1.0	1
dS	1.0	1
dZ	1.0	1
dR	1.0	1

Treshold 1.0 1

CALL a9

MASK 15

model to install : 1999-11-28_UT1_CA-model.dat

1999-11-28_UT1_CA-model.dat
1999-11-29_UT1_CA_2-model.dat
1999-11-27_UT1_CA_2-model.dat
1999-11-27_UT1_CA_7-model.dat

Figure 7 - POM main User Interface

This software works un-modified with both the ACS-based and ECS-based TCS.

Using the user interface provided by this package it is possible to:

- Retrieve from catalogues candidate targets over the visible sky
- Point the antenna in sequence to the selected targets
- Measure the pointing error (this must be done manually on the 12m since no software for controlling the CCD is available)
- Build a pointing model using the TPOINT software from Pat Wallace
- Install the pointing model in the TCS

<<ACS/ECS>> kpcmdtrans

This package is a protocol converter that makes transparent to **kppom** the fact that it is talking to a system based on ACS instead of ECS. Whenever a process of the **kppom** package sends a command to trkwsControl, it is translated to the equivalent AMS command and routed to AMS.

The kpcmdtrans process consist of two parallel threads:

- An ACS thread that handles all CORBA communication
- An ECS thread based on the evhTASK class that handles all ECS communication. This thread registers itself in the ECS environment as the trkwsControl process and effectively replaces this in a transparent way for applications.

<<ECS>> kpscan

This package implements a limited replacement for the "scanning system" for ECS under Linux, as the complete system is not yet available. The "scanning system" mirrors values in the ECS real time database on the LCU into the workstation database.

In some cases the ACS TCS takes care of mapping values into the workstation database that are expected by the **kppom** software but that are not directly provided as such by **ams**.

When using the ACS TCS, the **kpscanAMS** process periodically reads all needed values from the LCU ECS real time database, making any necessary intermediate calculation and unit conversion, and finally writing the values in the workstation ECS real time database.

This package exploits the fact that in the current implementation ACS relies on the ECS real time database for the configuration of characteristics, and that properties also write the current value there.

<<ECS>> kpsamp

This package implements a limited replacement for the "sampling system" for ECS on Linux, as the complete system is not yet available. The "sampling system" samples at high frequency (in our case up to 20 Hz) a few values (up to 4) in the ECS real time database on the LCU and generates files to be used for later data analysis.

ACS 0.0 does not yet provide such a system, and is foreseen for the final implementation.

This package exploits the fact that in the current implementation ACS relies on the ECS real time database for the configuration of characteristics and that properties also write the current value there.

<<API>> kpam

This package provides an antenna motion C-language API to interact with the 12m control electronics. The package includes the hardware drivers described in the section “Standard 12-Meter TCS” above. The package is used by both the ACS and ECS TCS implementations. The idea is, that to use a real ALMA antenna, this package would be replaced by an equivalent package for the ALMA antenna and no other software changes would be needed.

The package is essentially unchanged from the version used in 1999 except some work was necessary to port it from the MVME167 (68040 processor) CPU to the new MVME2604 (PowerPC processor) CPU.

The following figure shows the API and the related drivers.

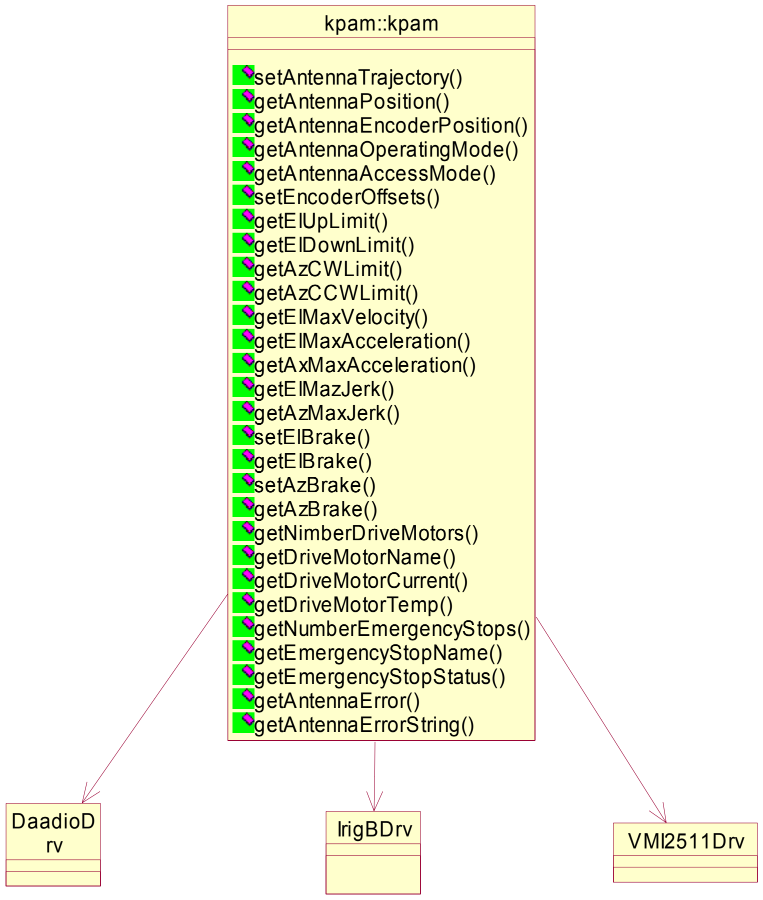


Figure 8 - kpam class diagram

The function setAntennaTrajectory() controls the antenna motion, and takes as input four polynomial coefficients for each axis plus a timestamp. The coefficients determine the position, velocity, acceleration and jerk at any particular moment for the azimuth and elevation axes.

The polynomial is

$$p = p0 + p1 * (t_{now} - t_{0)} + p2 * (t_{now} - t_{0)}^2 + p3 * (t_{now} - t_{0)}^3$$

where p is the position at time t_{now}; t_0 is the timestamp, and p0, p1, p2, p3 are the coefficients. In the current implementation p3 is always zero, as it's not used.

The package contains the position loop task that runs at a frequency of 100Hz (every 10 milliseconds). This task determines the current time t_{now} by reading the **IRIG-B** clock, and evaluates the polynomial for each axis to obtain the desired position. Then the task computes the difference between the antenna current position read from the **VMIVME-2511** board and the desired position. The resulting position errors are passed to the **DAADIO** board where they are converted to an analogue signal and then fed to the servo motor amplifier. The position loop implements pure positioning without PI feedback.

The position loop task drives both axes (az,el) and therefore needs a set of coefficients for each axis. In the current implementation the task is started at boot time.

Access to the **IRIG-B** board is hidden from other packages in the ECS by the **tims** module. This module provides a common set of features, like the function **tims**GetUTC() for obtaining UTC, to access hardware-specific features.

Also in the package is an clock task that starts when the application gets the INIT command. This task, among other things, sets the vxWorks clock to the **IRIG-B** time every 30 seconds. This is often enough to keep the vxWorks clock well synchronized to the IRIG clock as the maximum drift of the vxWorks clock is about 10 millisecond in 90 seconds.

In Simulation mode the IRIG clock task initializes the time from the walma workstation clock and then runs "open loop" from only the vxWorks clock.

<<API>> vlb

This small package provides several functions used only by <<API>>kpam. They came from the VLBA and are unchanged from the originals. They have been extracted from the <<API>>kpam package because they never change.

5.2 ECS Telescope Control Packages

The following package diagram shows the architecture of the TCS based on ECS.

No changes have been introduced to the software tested in 1999, except those mentioned in the **kpam** package.

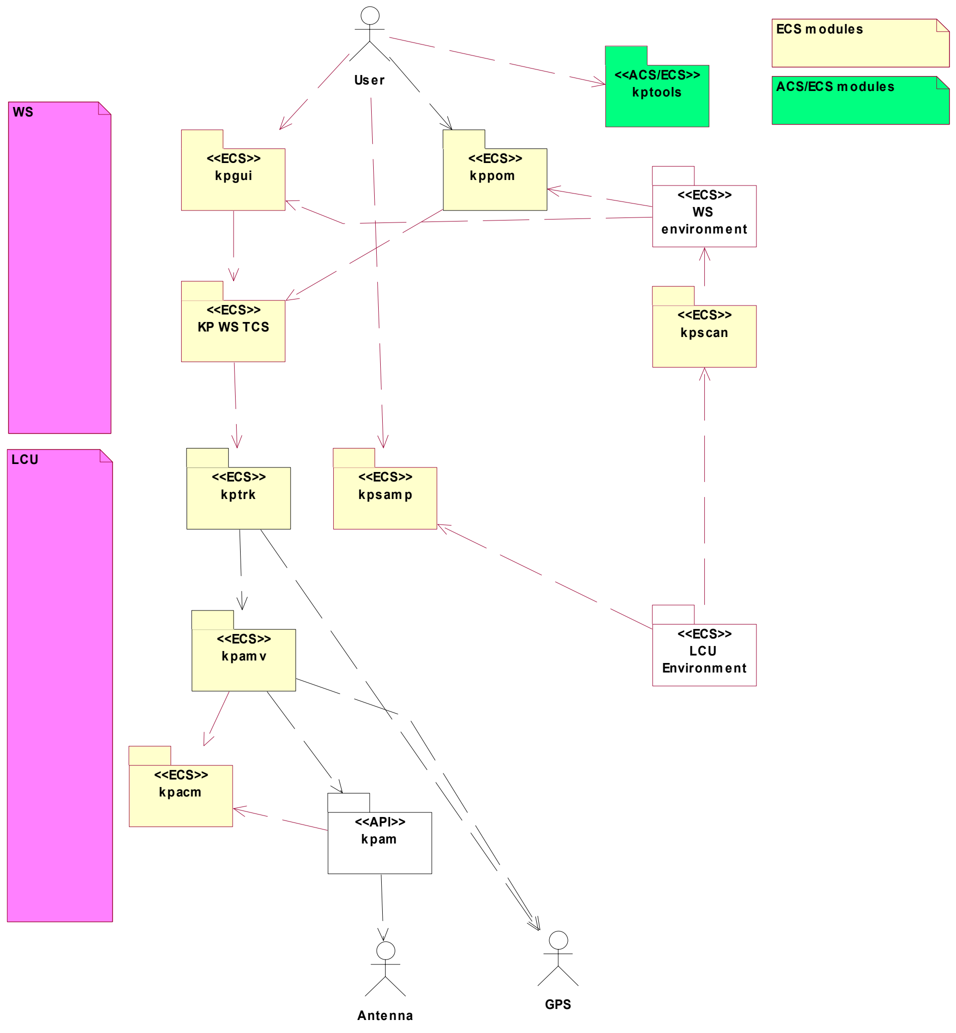


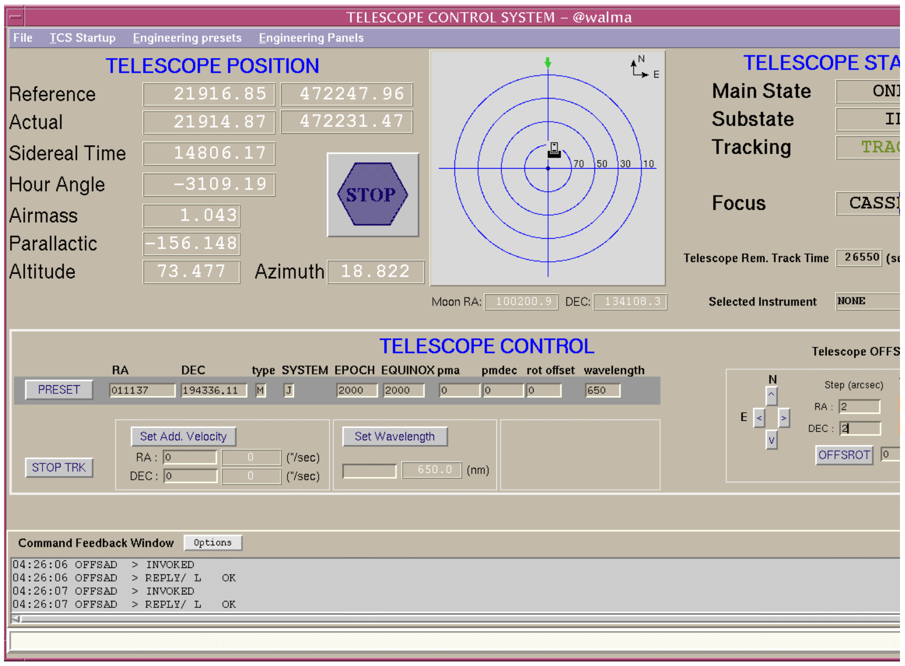
Figure 9 - ECS TCS package diagram

In the following paragraphs we describe the packages not already fully described in the previous section and how they interact.

<<ACS>> kpgui

This package contains a subset of the VLT TCS user interfaces stripped down to those implemented on the 12m.

The main panels are the antenna control panel kpguiTCS and the antenna status panel kpguiStatus, shown in the following pictures.



TELESCOPE POSITION

Reference 21916.85 472247.96
Actual 21914.87 472231.47
Sidereal Time 14806.17
Hour Angle -3109.19
Airmass 1.043
Parallactic -156.148
Altitude 73.477 Azimuth 18.822

Moon RA: 100200.9 DEC: 134108.3

TELESCOPE STA

Main State ONI
Substate II
Tracking TRA
Focus CASS
Telescope Rem. Track Time 26550 (se
Selected Instrument NONE

TELESCOPE CONTROL

RA	DEC	type	SYSTEM	EPOCH	EQUINOX	pma	pmdec	rot offset	wavelength
011137	194336.11	M	J	2000	2000	0	0	0	650

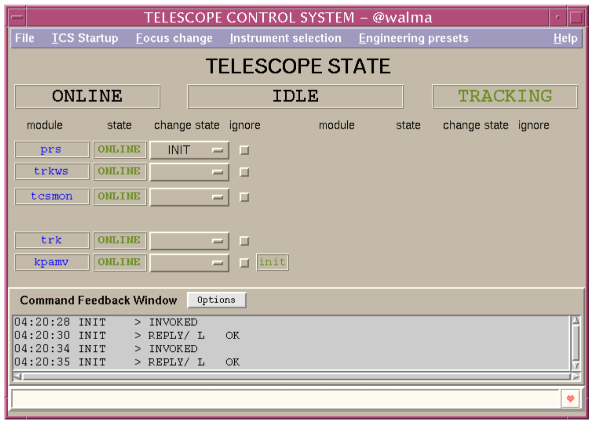
RA : 0 0 ("/sec)
DEC : 0 0 ("/sec)
650.0 (nm)

Telescope OFFS
Step (arcsec)
RA : 2
DEC : 2

Command Feedback Window

```
04:26:06 OFFSAD  > INVOKED
04:26:06 OFFSAD  > REPLY/ L   OK
04:26:07 OFFSAD  > INVOKED
04:26:07 OFFSAD  > REPLY/ L   OK
```

Figure 10 - TCS Main Control Panel



TELESCOPE STATE

ONLINE IDLE TRACKING

module	state	change state	ignore	module	state	change state	ignore
prs	ONLINE	INIT					
trkws	ONLINE						
tcsmon	ONLINE						
trk	ONLINE						
kpamv	ONLINE		init				

Command Feedback Window

```
04:20:28 INIT    > INVOKED
04:20:30 INIT    > REPLY/ L   OK
04:20:34 INIT    > INVOKED
04:20:35 INIT    > REPLY/ L   OK
```

Figure 11 - TCS Status Panel

<<ACS>> KP WS TCS

Most of the coordination software of the VLT TCS runs on the workstation

This package contains a stripped-down version of this software and consist of a number of software modules**: kptcs, kpmsw, kptif, kptrkws, kpprs, kptcsmon, kposf, tcscam**.

For more details on these modules we refer the reader to the VLT documentation. We only discuss the **kptrkws** module, as it is the most important, since it provides the interface between the high-level coordination software and user interfaces and the LCU real time control components.

Look at the documentation of the KP 1999 test for details on the changes that have been applied to the VLT modules to adapt them to the 12m.

<<ACS>> kptrk

The **kptrk** package implements the tracking algorithms to control the antenna and is functionally equivalent to the **ams** module for the ACS-based TCS.

It receives target coordinates from the user interface, performs all needed astronomical calculations using **slalib** (including pointing model corrections), and sends reference positions to **kpamv** using the interfaces defined in the **kpacm** package.

A fast foreground loop and a slow background loop perform in real time the conversion from astronomical coordinates in reference positions and velocities in horizon coordinates. The background loop is responsible for calculating at a low frequency the slowly varying, but computationally intensive parameters.

kptrk needs the following modification with respect to the standard VLT trk package to be compatible with **kpamv**:

Support two axes at the same time. The backward calculation from encoder values to reference (az,el) values has to be supported for two axes. This means it must provide the backwards-calculated encoder values (corrected for pointing model) that are in turn provided for the final backward calculation to ra, dec on the workstation done by **trkws**.

kptrk must provide (calculate) the coefficients of the polynomial required by the Antenna Motion API. This is done in a specific acmSendRefs function provided by **kpamv**.

If the kpamvSendrefs function uses the analytical formulae, **kptrk** has to pass additionally to (az,el) the latitude. Still the same function will be used, but the interpretation of its values changes.

<<ACS>> kpamv

This package implements a standard ECS LCU server application to interface ECS with **kpam**. It provides the standard ECS features, like database access, commands, scanning and GUIs.

The template for this application is the **citmp** module that is available in the ECS-CMM archive.

The **citmp** module provides a minimum standard ECS LCU application that is used as a starting point for any new LCU application.

The name of this package means **K**itt **P**eak **A**ntenna **M**otion adapted to **V**LT TCS.

Kpamv implements wrappers for the relevant functions and the corresponding commands of the LCU kpamvServer application.

Two functions require special attention:

kpamvSendRefs
This function is an interface function to the **kptrk** module. From the (az,el) values that are delivered by **kptrk** it derives the coefficients that are required by **kpam**. Its prototype is defined in acm.h (module kpacm) as *acmSEND_REF*. It complies with the standard **acmSendRefs** function that is the usual interface to the axis driving software.

The usage of the setPoints parameter is slightly modified. The member *posRot* contains the latitude that is needed to calculate the coefficients. The az, alt values are in the range of 0 to 2PI.

The meaning of *action* and st*ate* remains the same:

acmACTION: The action trk wants to perform:

acmACTION_STAR
Move to a new target and start tracking. The state is acmSTATE_PRESETTING until a the position error (difference between commanded and actual positions) is below a the threshold (configured in the database). After that the state switches to acmSTATE_TRACKING

acmACTION_FIXED
Move to a new target and stay there. The state is acmSTATE_PRESETTING until a the position error (difference between commanded and actual positions) is below a the threshold (configured in the database). After that the state switches to acmSTATE_IDLE

acmACTION_UPDATE
Updates the positions of the previously sent new target.

acmACTION_STAR_OFFS
Some axes need to handle this differently if the new position is supposed to be an offset. To provide this specific command the database of kptrk must be configured differently.

kpamvReadEnc
This function is an interface function to the **kptrk** module. It returns the current Encoder positions in the range of 0, 2PI. Its prototype is defined in acm.h (module kpacm) as *acmREAD_ENC*

Calculation of the polynomial coefficients

Two methods for calculating the coefficients are considered:

1. Using the analytical formulae of the (az,el) polynomial, i.e. (az,el) as a function of ha,dec. In this case **trk** needs a modification (see the section on **kptrk** above). This method works only with constant ha,dec. Therefore **trk** must be changed significantly to support differential tracking and pointing modeling.
2. Buffering actual (az,el) values, produced by **trk** and calculating the coefficients needed by **kpamv**.

The current implementation is based on 1.

kpamv implements wrappers for the relevant functions and the corresponding commands of the LCU kpamvServer application. In addition to the standard commands, the following table shows the commands that are specially implemented and their respective actions:

Command	Parameters	Action
INIT		**Initialise kpamv/kpam and spawn position loop and monitoring task** **(and stops a previously running ones)**
SETTRAJ	**azPos, elPos, azVel, elVel**	**Set the coefficient of the (az,el) polynomial motion (unit is degrees)**
GETTRAJ		**Get the coefficient of the (az,el) polynomial motion (unit is degrees)**
GETPOS		**Get current Axes positions in degrees**
GETAZLIM		**Get azimuth limits (unit is degrees)**
GETELLIM		**Get elevation limits (unit is degrees)**
SIMULAT	**kpamv, ant, clk**	**Starts simulation of the Antenna (drives), the IRIG Clock (clk) or kpamv (all)**
STOPSIM		**Stops any simulation and goes to state loaded**

Application specific commands of kpamv

The module is able to simulate the behavior of the real hardware, i.e. when the current encoder position is requested, it returns a position that complies with the set (az,el) trajectory.

A specific branch in the LCU ECS real time database is created that contains the relevant information of the antenna motion software

A monitoring task is running that polls, e.g. the current encoder position, and writes the results in the database to fill the DB with values that are written by the **kpam**.

The **kpamv** interface can be used at any time, i.e. also while the system is running under the control of the ACS TCS, to directly interact with the antenna control electronics.

This allows the ECS tools, like the Command Engineering Interface ccseiMsg or the Engineering Database Browser ccseiDb, to be used at any time. That is convenient since ACS 0.0 does not provide equivalent tools that are foreseen for the final system.

It will also use the standard interfaces defined in the **acm** module to interface in a standard way with **kptrk**.

kptrk needs the following modification to be compatible with **kpamv**:

Support two axes at the same time. The backward calculation from encoder values to reference (az,el) values has to be supported for two axes. Now only one axis is supported. That means it has to provide to backwards calculated encoder values (corrected for pointing model) that are in turn then provided for the final backward calculation to ra,dec on the workstation, done by **trkws**.

kptrk has to provide (calculate) the coefficients of the polynom required by the Antenna Motion API. This is done in a specific acmSendRefs function provided by **kpamv**.

If the kpamvSendrefs function uses the analytical formulae, **kptrk** has to pass additionally to alt/az the latitude. Still the same function will be used, but the interpretation of its values changes.

<<ECS>> kpacm

This package just provides a standard set of interface definitions.

The interfaces are implemented by the **kpamv** package and used by the **kptrk** package to pass to it (az,el) reference positions.

6 TEST AND DEVELOPMENT CONFIGURATIONS

Before the KP 2000 test period, the ACS and the TCS software was developed at different sites and with different hardware and software configurations.

The minimal necessary configuration consists of a Linux workstation, where all software can run, using the AMS in simulation mode.

In the period August-December 2000 these configurations were used for development with intermediate releases of both ACS 0.0 and TCS software.

6.1 Socorro

The installation in Socorro is the most complete. It was used for AMS development and for the final system integration before going to Kitt Peak

- Linux development workstation
- SUN workstation for VxWorks cross development
- LCU for software simulation consisting of only the MVME 2700 CPU board
- LCU for hardware simulation consisting of the MVME 2700 CPU boards, all 12m interface boards, and the 12m antenna hardware simulator.

6.2 ESO

The installation at ESO is similar to the one in Socorro, but does not include the 12m antenna hardware simulator. It was used for ACS development, ECS TCS integration and AMS testing.

- Linux development workstation

- SUN workstation for VxWorks cross development
- LCU for software simulation consisting of only the MVME 2700 CPU board

6.3 IRAM

The installation at IRAM consists only of a Linux development workstation. It was used for AMS development and ACS testing.

6.4 IJS Institute Ljubljiana

The installation at the IJS Institute in Ljubljiana consists only of a Linux development workstation. It was used for ACS development and testing.

6.5 AIRUB Bochum

The installation at AIRUB Bochum consists only of a Linux development workstation. It was used for ACS testing.

7 KP 2000 SYSTEM CONFIGURATION

7.1 Hardware

The following deployment diagram gives an overview of the hardware setup at the 12m with more details in the following subsections.

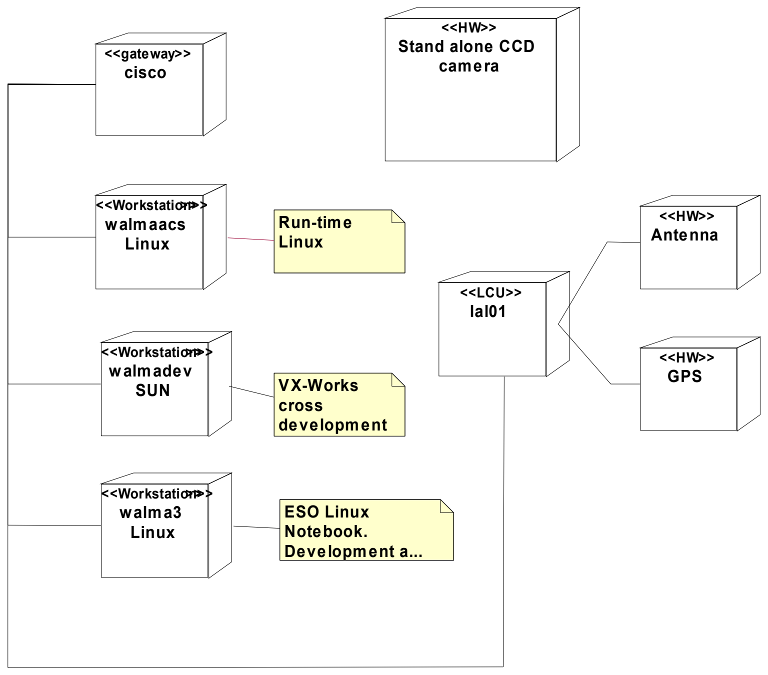


Figure 12 - Kitt Peak 12m test computer configuration

The following hardware was used during the test:

- Workstations and terminals
  - Linux workstation (walmaacs) for control
  - Sparc Ultra 1 (walmadev) workstation for cross development

  - ESO Notebook with Windows NT (walmapc1) and Linux under VmWare (walmapc2) for extra development station and spare.
- LCUs
  - Tracking LCU (lal01) with MVME2604 CPU and all hardware of the standard 12m TCS (described in section 4.1)
  - Spare LCU (lal02) for additional offline tests

7.2 Software

The following software was used during the test:

- Linux Red Hat 6.2
  - ECS Software Release NOV2000 Linux pre built June 2000
  - ACS 0.0, as from the ACS 0.0 tape, patched and built according to the procedure described in section 0. This includes the following packages, provided as tools:
    - ACE 5.1.4
    - TAO 1.1.4
    - Orbacus 4.0.3
    - Jdk 1.1.7
    - Swing 1.0.3
    - Combat 0.6.1
    - KPTCS software built according to the procedure described in section 0
- SOLARIS 2.6
  - VxWorks 5.4 (Tornado II)
  - ECS Software Release NOV2000 CCSLite
    - ACS 0.0 for SUN, built according to the procedure described in section 0.
    - KPTCS software built according to the procedure described in section 0

8 KP 2000 TEST REPORT

8.1 Test Organization and Execution

This section was written prior the test period and was used during the test as a checklist. During the test, notes about test execution and actual duration was added in the text boxes and the table columns *Comments* and *Results* were completed.

Assumptions

Planning of test cases, as well as estimates of time necessary to perform the tests, are based on the assumption that the antenna is fully operational when the test period starts. Thus it is assumed that encoders, motors, cable wraps, GPS, CCD (optical telescope), and all other parts of the antenna are performing according to specifications. The aim of the test period is to test the KPTCS software on a working antenna.

Summary of test sequence

The basic strategy is to use a bottom-up approach. Each test step requires that the previous step be performed successfully; at least to the extent that it impacts the following tests.

The sequence of tests is designed with the aim of gradually incorporating higher-level control parts. For the LCU, a set of tests is performed that verifies the complete LCU (hardware and software) before including other, coordinated tests.

The tests are composed of a number of “test blocks” (with one chapter per block in this document). The test blocks are divided in smaller test sets, and finally, a number of individual test cases are defined. There are test cases covering normal execution and various error cases. Each test case has a unique identifier.

The first steps in the test procedure are a repetition of the tests performed in 1999 with the ESO Common Software. They are used to assess the status of the antenna hardware and electronics, and warranty that the results obtained with this test are comparable with the previous test.

The test sequence can be summarized as follows:

- Install hardware and verify connections
- For the tracking LCU with ECS software:
    - Test basic software, interfacing to hardware
    - Test basic position control
    - Axis tracking module
- Antenna motion with ACS software:
    - Tracking modules
    - Integration tests
    - Pointing tests
    - Tracking tests

The table below is a summary of duration of the planned activities, assuming “ideal” conditions.

The total time allocated to the test is 8 days and 8 nights.

In the following sections, where tests are specified, there is also an estimated duration of the test. It might be necessary to modify these parameters, subject to availability of the antenna.

Task	Duration (hrs)
System installation (0)	**12**
Test antenna motion software with ECS software (8.3)	**12**
Test tracking software with ECS software (8.4)	**12**
Test tracking software with ACS (8.5)	**18**
Create pointing model (8.6)	**12**
Perform tracking tests (0)	**12**

The first four activities can be performed during daytime. So with 4 people performing the tests, work can be done in shifts effectively using 24 hours per day.

The first 4 activities will need a total 50 hours (about 2 days).

The last 2 activities have to be done during nighttime, in total 24 hours corresponding to 3 to 3 ½ nights. The 3 mornings can be used for data processing, development, debugging etc.

This should leave about three additional days for other tests and of buffer.

Activity logging

Everybody making tests at Kitt Peak shall produce a test log for each day at Kitt Peak. This log, which is a plain ASCII-file, is the most important record of the test period. A properly maintained test log is also the basis for the test report.

Test logs are used for the update of this report during and after the test period. The suggested procedure is to store log files in the directory ~almamgr/DAILY_LOGS. The name of the log is <yyymmdd>.<3-letter initials>; e.g. 20001203.rka for the logfile of Robert Karban for December 12th 2000. Hand-written notes are acceptable, since they will also be integrated in this report. Every action must be logged with a time stamp, a reference to the test identifier and any results/problems. Any modifications to hardware or software (change of code, configuration, data, etc.) must be logged. Any problem that cannot be immediately solved shall get an identifier, P<incremented number>. This is needed as a reference for future analysis. Furthermore, the test log may have pointers to log files, SPRs, data files etc. In such a case the reference (e.g. a logfile) shall be copied to the directory of the day.

Modifications

In principle it is prohibited to perform any changes whatsoever in $INTROOT or $VLTDATA directly! All changes, including environment configurations, shall be made in the module directory and then installed. This is to ensure that no modifications are lost.

However, in a few cases exceptions can be allowed for $VLTDATA, for practical reasons such as config files and for database definitions. A change made directly in $VLTDATA is by definition temporary and will be lost when the system is re-built. To become permanent, such a change must be reflected back in the correct module by manually copying file(s) from $VLTDATA to the module. Great care must be taken that modifications are not lost.

Modules that are checked out from the archive for modification shall be put in the directory ~almamgr/CURRENTLY_MODIFIED.

If a new (already archived) version of a module has to be installed, the module shall be cmmCopied to ~almamgr/TCS_SOURCE/TCS, whereas before the previous version of the module is renamed to module.<version>.

General considerations

Environmental preconditions of each test scenario shall be checked before the execution of the test.

In case of unexpected errors during the execution, the test shall be stopped. A decision to continue or to correct the error shall be taken. In case of important corrections, all tests previously performed shall be repeated. No tests involving antenna motions shall be allowed to continue if any such error persists.

8.2 Preparations

Purpose: Workstations and LCUs will be prepared in Socorro, transported to Kitt Peak and installed on the first day of testing.

Allocated time: 12h of hardware and software installation work at Kitt Peak

Pre-requisite: Hardware operational

Test execution:

Start of test: 2000/12/01 09:00

Actual test duration: 24h

Comments: The installation of the system on Kitt Peak required more time than foreseen because of an unexpected hardware problem. The back plane of the lal01VME chassis had a shortcut between two pins. This short never produced problems with the MVEM147 CPU (68020) board used normally on the antenna, but caused a severe malfunction when used with the new MVME2604 CPU (PowerPC) used for this test.

Preparations in Socorro

8.2.1.1 Installation of KPTCS modules

Test the installations of all KPTCS modules in $INTROOT on the KP Control Model. The purpose of this is to test installation procedures, and to check that all modules can be installed properly.

8.2.1.2 Installation of environments

- Test installation of LCU environment lal01 (Kitt Peak environment)
- Test installation of workstation environment walma (Kitt Peak environment)
- Installation of LCU environment lal01 (Control Model environment)
- Installation of workstation environment walma (Control Model environment)

8.2.1.3 Run test sequences

All test sequences shall be executed with the Kitt Peak Control Model (all or partially simulated Hardware).

8.2.1.4 Installation of KPTCS environment

Before shipping the hardware to Kitt Peak a full Installation of the Kitt Peak TCS configuration shall be done.

Preparations at Kitt Peak

8.2.1.5 Installation of Hardware

Install LCUs and workstations in 12m Control Room

Establish connection between LCU and walmaacs workstation, as well as between the walmaacs workstation and the local net and internet (for guide star catalog and cmm access)

Verify accessibility of CCD image

The names and/or IP addresses of workstations and LCUs were modified in Kitt Peak according to the assignment give to us.

In the final configuration we installed:

- Linux TCSW workstation: walmacs, 140.252.87.99
- SUN Development workstation: walmadev, 140.252.87.97
- VxWorks TCS LCU: lal01, 140.252.87.252
- VxWorks Spare/Test LCU: lal02, 140.252.87.96
- ESO Laptop, Win NT: walmapc1, 140.252.87.94
- ESO Laptop, Linux: walmapc2, 140.252.87.95
- IRAM Laptop, Linux: walmapc3, 140.252.87.85

The last day of testing, i.e.the 7th of January, the names of the LCUs were changed respectively lal01 to lal1c and lal02 to lal1t, to comply with a simplified installation procedure based on the **pkgin** software installation module.

8.2.1.6 Installation of KPTCS modules

- Verify that walma environments (workstation and LCU) start up correctly

Re-install KPTCS from scratch

8.3 Test Antenna Motion Software with ECS Software

Purpose: This test verifies the correct operation of the basic axis control software in the configuration used in 1999.

Allocated time: 12h

Pre-requisite: Hardware operational

<u>Test execution:</u>

Start of test: 2000/12/03 11:00

Actual test duration: 10h

Comments: This test was supposed to be performed first with each axis independently, with brakes engaged the second axis. However, the current version of the kpam module does not support this and both brakes must be released to allow antenna motion.

- **LCU lal01 is re-booted before starting the test**.
  When the LCU is rebooted, all ECS software processes are automatically loaded, started and set in STANDBY mode. All control loops are consequently open, but the system is ready to receive commands.
- **Workstation environment walma is running**.
  Is this is not the case, to start the environment issue the command:

```
> vccEnvStart -e walma -t QSEMU
```

Check the ACS and ECS documentation for more details on starting/stopping environments.

Commands are sent directly to the **`kpamvServer`** process on the LCU lal01 using the ESO Common Software tools:

- `msgSend` or `ccsei` engineering user interface
- `ccseiDb` Database Browser to browse the values stored in the LCU Real Time Database.

Check azimuth positioning (AZ-1)

Test Id	Command	Expected Result/Status	Comments	Result
AZ-1-1	**STOP**	**OK**		✓
AZ-1-2	**OFF**	**OK**		✓
AZ-1-3	**STATUS**	**LOADED/ Idle**		✓
AZ-1-4	**INIT**	**OK**	**INIT command was allowed only once. Code fixed.**	✓

AZ-1-5	**STATUS**	**LOADED/ idle**		✓
AZ-1-6	**ONLINE**	**OK**		✓
AZ-1-7	**STATUS**	**ONLINE**		✓
AZ-1-8	**SETTRAJ**	**az=-90** **el=90**		✓
AZ-1-9	**GETTRAJ**	**az=-90** **el=90**		✓
AZ-1-10	**GETPOS**	**az=-90** **el=90**		✓
AZ-1-11	**SETTRAJ** **azPos=-100** **elPos=90**	**az=-100** **el=90**	**Both axes must have brakes released, otherwise antenna moves to position set by first SETTRAJ command, then does not move any more.** **Code fixed.**	✓
AZ-1-12	**GETTRAJ**	**az=-100** **el=90**		✓
AZ-1-13	**GETPOS**	**az=-100** **el=90**		✓
AZ-1-14	**SETTRAJ** **azPos=66** **elPos=90**	**az=66** **el=90**	**just before limit**	✓
AZ-1-15	**GETTRAJ**	**az=66** **el=90**		✓
AZ-1-16	**GETPOS**	**az=66** **el=90**		✓
AZ-1-17	**SETTRAJ** **azPos=-293** **elPos=90**	**az=-293** **el=90**	**just before limit** **Antenna started to flip between –275 and 0 and did not react to new SETTRAJ commands.** **This problem was found also in 1999, but the fix went lost.** **Code fixed.**	✓
AZ-1-18	**GETPOS**	**az=-293** **el=90**		✓
AZ-1-19	**GETTRAJ**	**az=-293** **el=90**		✓

AZ-1-20	**GETPOS**	**az=-293** **el=90**		✓
AZ-1-21	**SETTRAJ** **azPos=25** **elPos=90**	**az=25** **el=90**		✓
AZ-1-22	**GETTRAJ**	**az=25** **el=90**		✓
AZ-1-23	**GETPOS**	**az=25** **el=90**		✓
AZ-1-24	**SETTRAJ** **azPos=-40** **elPos=90**	**az=-4** **el=90**		✓
AZ-1-25	**GETTRAJ**	**az=-40** **el=90**		✓
AZ-1-26	**GETPOS**	**az=-40** **el=90**		✓
AZ-1-27	**SETTRAJ** **azPos=0** **elPos=90**	**az=0** **el=90**		✓
AZ-1-28	**GETTRAJ**	**az=0** **el=90**		✓
AZ-1-29	**GETPOS**	**az=0** **el=90**		✓
AZ-1-30	**INIT**	**OK**		✓
AZ-1-31	**STATUS**	**LOADED/** **Idle**		✓
AZ-1-32	**ONLINE**	**OK**		✓
AZ-1-33	**STATUS**	**ONLINE**		✓
AZ-1-34	**SETTRAJ** **azPos=25** **elPos=90**	**az=25** **el=90**		✓
AZ-1-35	**GETTRAJ**	**az=25** **el=90**		✓
AZ-1-36	**GETPOS**	**az=25** **el=90**		✓

Check elevation positioning (EL-1)

Test Id	Command	Expected Result/Status	Comments	Result
EL-1-1	STOP	OK		✓
EL-1-2	OFF	OK		✓
EL-1-3	STATUS	LOADED/ Idle		✓
EL-1-4	INIT	OK		✓
EL-1-5	STATUS	LOADED/ idle		✓
EL-1-6	ONLINE	OK		✓
EL-1-7	STATUS	ONLINE		✓

EL-1-8	SETTRAJ	az=0
		el=90
				✓

EL-1-9	GETTRAJ	az=0
		el=90
				✓

EL-1-10	GETPOS	az=0
		el=90
				✓

EL-1-11	SETTRAJ
	azPos=0
	elPos=45
		az=0
		el=45
				✓

EL-1-12	GETTRAJ	az=0
		el=45
				✓

EL-1-13	GETPOS	az=0
		el=45
				✓

EL-1-14	SETTRAJ
	azPos=0
	elPos=15
		az=0
		el=15
			just before limit	✓

EL-1-15	GETTRAJ	az=0
		el=15
				✓

EL-1-16	GETPOS	az=0
		el=15
				✓

EL-1-17	SETTRAJ
	azPos=0
	elPos=67
		az=0
		el=67
				✓

EL-1-19	GETTRAJ	az=0
		el=67
				✓

EL-1-20	GETPOS	az=0
		el=67
				✓

EL-1-21	SETTRAJ	az=0		✓

	azPos=0 **elPos=25**	**el=25**		
EL-1-22	**GETTRAJ**	**az=0** **el=25**		✓
EL-1-23	**GETPOS**	**az=0** **el=25**		✓
EL-1-24	**SETTRAJ** **azPos=0** **elPos=88**	**az=0** **el=88**		✓
EL-1-25	**GETTRAJ**	**az=0** **el=88**		✓
EL-1-26	**GETPOS**	**az=0** **el=88**		✓
EL-1-27	**INIT**	**OK**		✓
EL-1-28	**STATUS**	**LOADED/** **Idle**		✓
EL-1-29	**ONLINE**	**OK**		✓
EL-1-30	**STATUS**	**ONLINE**		✓
EL-1-31	**SETTRAJ** **azPos=0** **elPos=90**	**az=0** **el=90**		✓
EL-1-32	**GETTRAJ**	**az=0** **el=90**		✓

Check elevation/azimuth positioning (AA-1)

Test Id	Command	Expected Result/Status	Comments	Result
AA-1-1	**STOP**	**OK**		✓
AA-1-2	**OFF**	**OK**		✓
AA-1-3	**STATUS**	**LOADED/** **Idle**		✓
AA-1-4	**INIT**	**OK**		✓
AA-1-5	**STATUS**	**LOADED/** **Idle**		✓
AA-1-6	**ONLINE**	**OK**		✓
AA-1-7	**STATUS**	**Online/idle**		✓
AA-1-8	**SETTRAJ**	**AzPos=-240**		✓

	azPos=-240 **elPos=45,**	**ElPos=45**		
AA-1-9	**GETTRAJ**	**AzPos=-240** **elPos=45**		✓
AA-1-10	**SETTRAJ** **azPos=66** **elPos=75,**	**AzPos=66** **elPos=75**		✓
AA-1-11	**GETTRAJ**	**AzPos=66** **elPos=75**		✓
AA-1-12	**SETTRAJ** **azPos=-180** **elPos=20,**	**AzPos=-180** **elPos=20**		✓
AA-1-13	**GETTRAJ**	**AzPos=-180** **elPos=20**		✓
AA-1-14	**SETTRAJ** **azPos=34** **elPos=66**	**AzPos=34** **elPos=66**		✓

At this point we added tests for elevation limit positions. No problems were observed

Check azimuth tracking (AZ-2)

In this section test using the SETTRAJ command with both a target azimuth position ***p*** in degrees and a velocity ***v*** in deg/s.

The SETRAJ command maintains the given velocity for one minute and then stops. For example, if at the time the command is sent antenna was already at position ***p***, after one minute it will stop at position

$$p' = p + v * 60 \text{ deg}$$

In the test executed in 1999, the SETTRAJ command was sent without taking into account the initial position of the antenna. In this case, the antenna tries at any time t to reach the position

$$p' = p + v * t \text{ deg}$$

at the given speed. This test is not realistic, since in normal usage the antenna will be sent to a target position at maximum speed. A specific speed is specified only during sidereal tracking with given speed to follow the apparent motion of the target object.

We replaced the test steps of 1999 with a new set that gives as target position always the position reached by the antenna in the previous command.

The final position can be verified using the GETPOS command, but is it easier to check directly in the TCS control panels or using the ccseiDb online database browser.

Test Id	Command	Expected Result/Status	Comments	Result
AZ-2-1	**STOP**	**OK**		✓

AZ-2-2	**OFF**	**OK**		✓
AZ-2-3	**STATUS**	**LOADED/ uninitialised**		✓
AZ-2-4	**INIT**	**OK**		✓
AZ-2-5	**STATUS**	**LOADED/ initialised**		✓
AZ-2-6	**ONLINE**	**OK**		✓
AZ-2-7	**STATUS**	**Online/idle**		✓
AZ-2-8	**SETTRAJ azPos=-90, azVel=0.0**	**AzPos=-90**		✓
AZ-2-8b	**SETTRAJ azPos=-90, azVel=0.0**	**az moves with 0.1deg/s**	**After 60 sec: AzPos=-84**	✓
AZ-2-9	**GETTRAJ**	**azPos=-90, azVel=0.1**		✓
AZ-2-10	**SETTRAJ azPos=0, azVel=1**	**az moves with 1deg**	**AzPos=60, azVel=1**	**Replace d**
AZ-2-11	**GETTRAJ**	**azPos=0, azVel=1**	**Stops after 60secs**	**Replace d**
AZ-2-12	**SETTRAJ AzPos=-90, azVel=2**	**azPos=30, azVel=2**		**Replace d**
AZ-2-13	**GETTRAJ**	**azPos=-90, azVel=2**		**Replace d**
AZ-2-14	**SETTRAJ azPos=30, azVel=2**	**azPos=-199**	**Flips at +40 and unwraps**	**Replace d**
AZ-2-15	**GETTRAJ**	**azPos=30, azVel=2**		**Replace d**
AZ-2-16	**SETTRAJ azPos=-90, azVel=0.0**	**AzPos=-90**		✓
AZ-2-17	**SETTRAJ azPos=-90, azVel=0.5**	**azPos= azVel=**	**After 60 sec. Stops at az=-60**	✓
AZ-2-18	**SETTRAJ azPos=-90, azVel=0.0**	**azPos=-90**		✓
AZ-2-19	**SETTRAJ azPos=-90, azVel=0.5**		**After 60 sec. Stops at az=-60**	✓
AZ-2-20	**SETTRAJ azPos=-90, azVel=0**	**azPos=-90**		✓
AZ-2-21	**SETTRAJ azPos=-90, azVel=0.25**		**After 60 sec. Stops at az=-75**	✓
AZ-2-22	**SETTRAJ**	**azPos=-90**		✓

	azPos=-90, azVel=0.0			
AZ-2-23	SETTRAJ azPos=-90, azVel=0.025		After 60 sec. Stops at az=88.5	✓
AZ-2-24	SETTRAJ azPos=-90, azVel=0.0	azPos=-90		✓
AZ-2-25	SETTRAJ azPos=-90, azVel=0.006		After 60 sec. Stops at az=89.64	✓

At this point we tried negative velocities that were not included in the KP2000 test, but we were not been able to set them.
The CDT of the SETTRAJ command was wrong and did not allow negative values for the velocity.
We have fixed the CDT, but we have not been able to reload it properly (it seems there is a problem with CCSLite Linux). We have decided to verify this later, after the opportunity of a complete restart of the system. This was done one day later with few verification tests, not logged in the checklist.

Check elevation tracking (EL-2)

Test Id	Command	Expected Result/Status	Comments	Result
EL-2-1	STOP	OK		✓
EL-2-2	OFF	OK		✓
EL-2-3	STATUS	LOADED/ Uninitialised		✓
EL-2-4	INIT	OK		✓
EL-2-5	STATUS	LOADED/ Initialised		✓
EL-2-6	ONLINE	OK		✓
EL-2-7	STATUS	Online/idle		✓
EL-2-8	SETTRAJ elPos=45, elVel=0.0		el=45	✓
EL-2-9	SETTRAJ elPos=45, elVel=0.1	el moves with 0.1deg	el=51	✓
EL-2-10	GETTRAJ	ElPos=45, elVel=0.1		✓
EL-2-11	SETTRAJ elPos=75, elVel=0.0		el=75	✓

EL-2-12	**SETTRAJ** **elPos=75, elVel=0.2**	**el moves with 0.2deg**	**el=87**	✓
EL-2-13	**GETTRAJ**	**ElPos=75, elVel=0.2**		✓
EL-2-14	**SETTRAJ** **elPos=25, elVel=0.0**		**el=25**	✓
EL-2-15	**SETTRAJ** **elPos=25, elVel=0.5**		**el=55** **Here we made a typo in the speed value and we went into limits. In this situation the command never completes and we cannot send another command, since in the current implementation a new command is not handled properly is a previous one is still under execution. We had to stop and restart the system.**	✓
EL-2-16	**GETTRAJ**	**ElPos=25, elVel=0.5**		✓

Check other commands

Test Id	Command	Expected Result/Status	Comments	Result
AM-1-7	**GAZLIM**	**66.797,-293.203**		✓
AM-1-8	**GELLIM**	**90,14.8**		✓

At this point (21:30) we felt rather confident in the status of the system that seemed to perform as in 1999.
We have then decided to suspend the tests based on the checklist and to use the nighttime to point some objects in the sky, using the last set pointing model coefficients determined in 1999.

We then started the ACS TCS following the procedure described in section 8.4, performed basic tests and looked for some bright objects following the procedure described in section 0.

We had no problems in pointing first to the moon, then to all visible planets and some selected bright stars. In all cases the object was within the field of the CCD camera (20 arc-minutes aperture), although normally not on the reference point.

8.4 Test Tracking Software with ECS Software

Purpose: Verify the basic positioning and tracking at LCU level, the basic start/stop functionality, and the general system performance of the combined TCS package using the software configuration of the test performed in 1999.

Allocated time: 12h

Test execution:

Start of test: 2000/12/04 09:00

Actual test duration: 8h

Comments:

Commands are sent directly to the **kpamvServer** and to the **kptrkServer** on the LCU lal01 using the ESO Common Software tools:

- msgSend or ccsei engineering user interface
- ccseiDb Database Browser to browse the values stored in the LCU Real Time Database.

The user interfaces developed for the KP 1999 test can also be used to monitor and control the antenna during the test.

Notice that the commands sent to kptrkServer follow VLT standard in the order of arguments. For example, the OBJFIX command has Elevation as first argument and Azimuth as second argument. The same apply to offset commands.

Positioning LCU level (TRK-1)

Test Id	Command	Expected Result/Status	Comments	Result
TRK1-1	**kpamvServer: STOP**	**OK**		✓
TRK1-2	**OFF**	**OK**		✓
TRK1-3	**STATUS**	**loaded. not initialised**		✓
TRK1-4	**ONLINE**			✓
TRK1-5	**STATUS**	**Online. initialised**		✓
TRK1-6	**kptrkServer: STOP**	**OK**		✓
TRK1-7	**INIT**	**OK**		✓
TRK1-8	**ONLINE**	**OK**		✓
TRK1-9	**STATUS**	**ONLINE/IDLE**		✓
TRK1-10	**OBJFIX 60,0**	**el axis moves to 60deg** **az=0**		✓
TRK1-11	**OBJFIX 80,0**	**80**		✓
TRK1-12	**OBJFIX 60,0**	**60**		✓
TRK1-13	**OBJFIX 30,0**	**30**		✓
TRK1-14	**OBJFIX 10,0**	**rejected**	**CCSLite for Linux is not able to handle properly error replies from the LCU. Instead of**	✓

			getting a correct error reply, we seem to get an OK reply, but on the logMonitor the following error is logged: "error stack too big"	
TRK1-15	OBJFIX 60,0	60		✓
TRK1-16	OBJFIX 15,0	15	just before limit	✓
TRK1-17	OBJFIX 89.999,0	89.999	just before limit	✓
TRK1-17a	OBJFIX 20,0	20.0		✓
TRK1-18	OFFSAA 1800,0	20.50		✓
TRK1-19	OFFSAA –1800,0	20.0		✓
TRK1-20	OBBJFIX 60,0	60		✓
TRK1-21	OFFSAA –3600,0	DON'T WAIT FOR REPLY		✓
TRK1-22	OFFSAA 1200,0 not waiting for reply.	Reply saying previous command interrupted	Due to the above mentioned CCSLite problem we seem to get an OK reply instead of a command interrupted error reply	✓
TRK1-23	OFFSAA 1200,0 not waiting for reply	Reply saying previous command interrupted	As above	✓
TRK1-24	OFFSAA 1200,0 not waiting for reply	Reply saying previous command interrupted	As above	✓
TRK1-25	OBJFIX 45,-100	el=90, az=-100		✓
TRK1-26	OBJFIX 45,-180	-180		✓
TRK1-27	OBJFIX 45,-200	-200		✓
TRK1-28	OBJFIX 45,30	30		✓
TRK1-29	OBJFIX 45,60	60		✓
TRK1-30	OFFSAA 0,1800	60.5		✓
TRK1-31	OFFSAA 0,–1800	60		✓
TRK1-33	OFFSAA 0,–3600			✓
TRK1-34	OFFSAA 0,1200 not waiting for reply.			✓
TRK1-35	OFFSAA 0,1200 not			✓

	waiting for reply			
TRK1-36	**OFFSAA 0,1200 not waiting for reply**			✓
TRK1-38	**OBJNAME ZENITH**	**el=90**		✓
TRK1-39	**OBJNAME PARK**	**el=90 az = 0**		✓
TRK1-40	**OBJFIX 90,66**	**el=90, az=66**		✓
TRK1-41	**OBJFIX 80,-293**	**el=80, az=-293**		✓
TRK1-42	**OBJFIX 15,100**	**el=15, az=100**	**Out of range**	✓
TRK1-43	**OBJFIX 45,0**	**el=45, az=0**		✓
TRK1-44	**OBJFIX 80,-10**	**el=80, az=-10**		✓
TRK1-45	**OBJFIX 80,10**	**el=80, az=10**		✓

Tracking (TRK-2 to TRK-4)

At this point we felt confident in the stability of the system and in particular of the LCU.

Instead of sending commands to the LCU processes, we have decided to start-up the complete workstation ECS TCS and to send the equivalent TCS commands using the TCS user interfaces and having the commands follow the complete command and data flow chain.

To start-up the ECS TCS issue the following command:

➢ kptoolsStartTcs lal01 -vlt

where lal01 is the environment name for the LCU

To stop the ECS TCS issue the following command:

➢ kptoolsStopTcs -vlt

Once the ECS TCS is started, the TCS main control panels can be started:

➢ kpguiTCS

➢ kpguiStatus

For more details on the operation of the VLT TCS, refer to the VLT User Manual, UT Book, v.3.1, J. Spyromilio - VLT-MAN-ESO-10200-16340

8.4.1.1 Basic tracking commands (TRK-2)

Test Id	Command	Expected Result/Status	Comments	Result
TRK2-1	**OBJSTAR** **<HA=-1h>,300000**	**el=77** **az=80**	**Get ST with GETST then set RA~ST -> HA~0**	✓
TRK2-2	**STATUS**	**ONLINE/TRACKING**		✓
TRK2-3	**wait 10’**			✓
TRK2-4	**OFFSAD 0,1800**	**OK**		✓

		el ~ +0.1deg **az very small change**		
TRK2-5	**OFFSAD 0,-1800**	**OK** **el ~ -0.1deg** **az very small change**		✓
TRK2-6	**OFFSAD 0,-3600**			✓
TRK2-7	**OFFSAD 0,-3600 not waiting for reply**			✓
TRK2-8	**OFFSAD 0,-3600 not waiting for reply**	**net motion el ~-0.3deg.**		✓
TRK2-9	**OBJSTAR** **<HA ~ -15'>, 200000**	**OK** **el=77** **az=164**	**max velocity**	✓
TRK2-10	**track ~30'**			✓
TRK2-11	**OBJSTAR <HA~1h>, 100000**	**OK** **el=63** **az=217**		✓
TRK2-12	**SETAV 0.5,0**	**OK**		✓
TRK2-13	**GETAV**			✓
TRK2-14	**check that RA is increasing by 0.5"/sec, i.e. 1sec per 30secs**	**no extra motion in el (or VERY small)**		✓
TRK2-15	**SETAV –0.50,**	**OK**		✓
TRK2-16	**GETAV**			✓
TRK2-17	**check that RA is decreasing by 0.5"/sec, i.e. 1 sec per 30 secs.**			✓
TRK2-18	**SETAV 0,0**	**stops additional movement**		✓
TRK2-19	**wait 1h**	**nice, quiet tracking**		✓
TRK2-20	**STOP**	**OK**		✓
TRK2-21	**OBJNAME zenith**	**el moves to 90**		✓
TRK2-22	**STANBDY**	**OK**		✓
TRK2-23	**KpamvServer: STOP**	**OK**		✓
TRK2-24	**OFF**	**OK**		✓
TRK2-25	**OBJSTAR** **<HA ~ -4h>, -100000**	**OK** **el=15.5** **az=112**	**close to el limit**	✓
TRK2-26	**OBJSTAR**	**OK**	**close to az**	✓

	<HA ~ -4.8h>, -330000	el=30
		az=67
			limit	

TRK2-27	OBJSTAR
	<HA ~ -4h>, 360000
		OK
		el=40
		az=-293
			close to az limit	✓

TRK2-28	OBJSTAR
	<HA ~ +10'>, 320000
		OK
		el=89
		az=10
			close to el limit	✓

TRK2-29	OBJSTAR
	<HA ~ +10'>, 560000
		OK
		el=65
		az=0
			go through az=0	✓

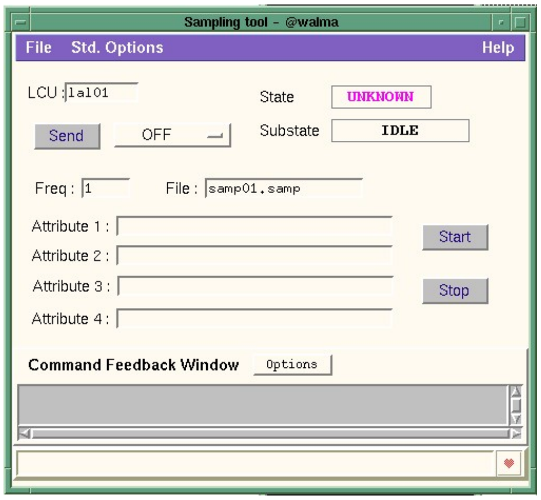
				✓

8.4.1.2 Quality position control – Dome closed (TRK-3)

In this section we verify tracking performances with the dome closed.

We set the antenna tracking in sidereal coordinates and we sample at 10Hz the azimuth and elevation position errors. Sampling is done using the `kpsampServer` process running on the LCU and reading the values directly from the online database.

Sampling is configured and started using the kpsampgui user interface, shown in the next figure.



LCU : la101
State UNKNOWN
OFF
Substate IDLE
Freq : 1
File : samp01.samp
Attribute 1 :
Attribute 2 :
Attribute 3 :
Attribute 4 :
Command Feedback Window

Figure 13 - Sampling Configuration User Interface (kpsampgui)

The samples are packed and stored in a file for later analysis. You can find the data files in the **kpdoc** module

Test Id	Command	Expected Result/Status	Comments	Result

TRK3-1	OBJSTAR
	<HA=~-10.0'>,300000
		el=87.106549
		az=131.810600
		We track through the meridian at high elevation. Final
			Get ST with GETST then set RA=ST+10 -> HA~-10	✓

		position is ST+10’		
TRK3-2	Sample data for 20’			✓
TRK3-3	OBJSTAR <HA=-3.5h>,300000	el=45.392568 az=78.064719		✓
TRK3-4	Sample data for 200’			✓
TRK3-5	OBJSTAR <HA=~-10.0’>,-100000	El=87.106549 az=131.810600 We track through the meridian at low elevation. Final position is ST+10’	Get ST with GETST then set RA=ST+10 -> HA~-10	✓
TRK3-6	Sample data for 20’			✓

8.4.1.3 Check database commands (TRK-4)

Test Id	Command	Expected Result/Status	Comments	Result
TRK-4-1	OBJSTAR <HA=~-10.0’>,300000	el=87.1 az=131.8	Get ST with GETST then set RA=ST+10 -> HA~-10	✓
TRK-4-2	GETPOS			✓
TRK-4-3	GETPOS			✓
TRK-4-4	GETPOS			✓

8.5 Test Tracking Software with ACS

Purpose: Verify the basic positioning and tracking at the LCU level, the basic start/stop functionality, and the general system performance of the combined TCS package.

Allocated time: 18h

<u>Test execution:</u>
Start of test: 2000/12/04 19:00
Actual test duration: 8h
Comments:

The KP 2000 system based on ACS must be started, and the time on all LCUs and workstations must be verified before starting the test.

The LCU should be rebooted before switching from ECS to ACS control. If this is not done, it is at least necessary to put the `kptrkServer` process in `STANDBY`, by sending the following command:

```
➢ msgSend -n lal01 kptrkServer STANDBY ""
```

When the LCU is rebooted, only ECS TCS processes are automatically loaded, started and set in STANDBY state.

ACS TCS processes are automatically loaded but the control task is not started, since the AMS device does not provide the equivalent of the `STANDBY` mode, and the control loops are always closed. This means that it is possible to dynamically switch from ECS to ACS TCS, but not the other way around.

To start the LCU processes[1]:

```
➢ rlogin lal01
➢ sp ActivateStart, "Activator"
➢ logout
```

To start the ACS TCS (workstation processes) issue the command:

```
➢ kptoolsStartTcs  lal01
```

where lal01 is the environment name for the LCU

To stop the ACS TCS issue the following command:

```
➢ kptoolsStopTcs
```

The new KP Java user interfaces are used to send commands and to monitor the antenna. To start it:

```
➢ cd ~/TCS_SOURCE/TCS/amsclient
➢ ./amsgui
```

This last command is a script that builds on the fly the Java classpath and executes the application.

ESO Common Software `ccseiDb` database browser is used to monitor properties that are not displayed in the panels.

Positioning LCU level (TRK-5)

Test Id	Command	Expected Result/Status	Comments	Result
TRK-5-10	**OBJFIX 0,60**	**alt axis moves to az =0 and El=60deg**	**At start time, the mount used to move away.** **Problem solved.**	✓
TRK-5-11	**OBJFIX 0,80**	**80**		✓
TRK-5-12	**OBJFIX 0,60**	**60**		✓
TRK-5-13	**OBJFIX 0,30**	**30**		✓
TRK-5-14	**OBJFIX 0,10**	**rejected**	**In the current version it is no more rejected.** **It goes to limit = 14.8deg**	✓
TRK-5-15	**OBJFIX 0,60**	**60**		✓

[1] When using the AMS LCU Software in a simulation installation, without hardware boards, it is necessary first to set the kpam API in simulation mode, by sending the following commands:

```
➢ msgSend lal01 kpamvServer SIMULAT
➢ msgSend lal01 kpamvServer INIT
➢
```

TRK-5-16	**OBJFIX 0,15**	**15**	**just before limit**	✓
TRK-5-17	**OBJFIX 0,89.999**	**89.999**	**just before limit**	✓
TRK-5-17a	**OBJFIX 0,20**	**20.0**		✓
TRK-5-18	**OFFSAA 0,1800**	**20.5**	**Works with Java GUI.** **Does not work with client1.tcl.** **Problem solved.**	✓
TRK-5-19	**OFFSAA 0,–1800**	**20.0**		✓
TRK-5-20	**OBJFIX 0,60**	**60**		✓
TRK-5-21	**OFFSAA 0,–3600** **3 times**	**Don't wait for reply between 1st, 2nd and 3rd commands**	**The mount goes to el=57deg**	✓
TRK-5-22	**OFFSAA 0,3600** **3 times**	**Don't wait for reply between 1st, 2nd and 3rd commands**	**The mount returns to el=60deg**	✓
TRK-5-25	**OBJFIX –100,45**	**az=-100, el=45**		✓
TRK-5-26	**OBJFIX –180,45**	**az=-180**		✓
TRK-5-27	**OBJFIX –200,45**	**az=-200**		✓
TRK-5-28	**OBJFIX 30,45**	**az=30**		✓
TRK-5-29	**OBJFIX 60,45**	**az=60**		✓
TRK-5-30	**OFFSAA 1800,0**	**az=60.5**		✓
TRK-5-31	**OFFSAA –1800,0**	**az=60**		✓
TRK-5-32	**OBJFIX 100,90**	**el=90**	**The mounts wraps to az=-260deg**	✓
TRK-5-33	**OFFSAA –3600,0** **3 times**	**Don't wait for reply between 1st, 2nd and 3rd commands**	**The mount goes to az=-263deg**	✓
TRK-5-34	**OFFSAA 3600,0** **3 times**	**Don't wait for reply between 1st, 2nd and 3rd commands**	**The mount returns to** **az=-260deg**	✓
TRK-5-37	**OBJFIX 45,45**	**az=45, el=45**		✓
TRK-5-40	**OBJFIX 66,90**	**az=66, el=90**		✓
TRK-5-41	**OBJFIX –293,80**	**az=-293, el=80**		✓
TRK-5-42	**OBJFIX 100,15**	**az=100, el=15**		✓
TRK-5-43	**OBJFIX 0,45**	**az=0, el=45**		✓
TRK-5-44	**OBJFIX –10,80**	**az=-10, el=80**		✓
TRK-5-45	**OBJFIX 10,80**	**az=10, el=80**		✓

Tracking (TRK-6, TRK-7)

8.5.1.1 Basic tracking commands (TRK-6)

Test Id	Command	Expected Result/Status	Comments	Result
TRK-6-1	**OBJSTAR <HA=-1hrs>,300000 RA=ST-HT, As ST=13000 then OBJSTAR 23000,300000**	**az=95 el=77**		✓
TRK-6-2	**Check on disp STATUS**	**ON TARGET**		✓
TRK-6-3	**wait! 10 minutes**			✓
TRK-6-4	**OFFSAD 1800,0**			✓
TRK-6-5	**OFFSAD -1800,0**		**The mount returns to ra=2.5h, dec=30deg**	✓
TRK-6-6	**OFFSAD -3600,0**			✓
TRK-6-7	**OFFSAD –3600,0 not waiting for reply**			✓
TRK-6-8	**OFFSAD -3600,0 not waiting for reply**		**The mount goes to ra=2.3h**	✓
TRK-6-9	**OBJSTAR <HA ~ -15min>, 20:00:00**		**max velocity**	✓
TRK-6-10	**track ~30min**			✓
TRK-6-11	**OBJSTAR <HA~1hrs>, 100000 (dec=10deg)**	**OK**		✓
TRK-6-12	**SETAV 0.5,0**	**OK wait for some minutes**		✓
TRK-6-14	**check that RA is increasing by 0.5”/sec, i.e. 1sec per 30secs on the sky**		**ActualRa increases of 0.000564H each mn of time**	✓
TRK-6-15	**SETAV –0.50,**	**OK**		✓
TRK-6-16	**Monitor AV**			✓
TRK-6-17	**check that RA is decreasing by 0.5”/sec, i.e. 1 sec**			✓

	per 30secs on the sky.			
TRK-6-18	**SETAV 0,0**	**stops additional movement**		✓
TRK-6-19	**wait 1hour**	**nice, quiet tracking**		✓
TRK-6-20	**STOP**	**OK**		✓

8.5.1.2 Quality position control – Dome closed (TRK-7)

In this section we verify tracking performances with the dome closed.
We set the antenna tracking in sidereal coordinates and we sample at 10Hz the azimuth and elevation position errors.

Sampling is done using the `kpsampServer` process running on the LCU and reading directly the values from the online database. Sampling is configured and started using the kpsampgui user interface shown in Figure 12.

The samples are packed and stored in a file for later analysis. The data files can be found in the **kpdoc** module where the ACS and ECS measures described in section 8.4.1.2 can be compared.

Test Id	Command	Expected Result/Status	Comments	Result
TRK7-1	**OBJSTAR** **<HA=~-10.0'>,300000**	**el=87.106549 az=131.810600** **We track through the meridian at high elevation. Final position is ST+10'**	**Get ST with GETST then set RA=ST+10 -> HA~-10**	✓
TRK7-2	**Sample data for 20'**			✓
TRK7-3	**OBJSTAR** **<HA=-3.5h>,300000**	**el=45.392568 az=78.064719**		✓
TRK7-4	**Sample data for 200'**			✓
TRK7-5	**OBJSTAR** **<HA=~-10.0'>,-100000**	**El=87.106549 az=131.810600** **We track through the meridian at low elevation. Final position is ST+10'**	**Get ST with GETST then set RA=ST+10 -> HA~-10**	✓
TRK7-6	**Sample data for 20'**			✓

8.6 Create Pointing Model

Purpose: Build optical pointing model using pomgui panels and tpoint

Allocated time: 12h

Test execution:

Find first objects (0), with ECS TCS:

- ***Start of test:*** 2000/12/03 21:30
- ***Actual test duration:*** 3h

Find first objects (0), with ACS TCS:

- ***Start of test:*** 2000/12/04 22:30
- ***Actual test duration:*** 4h

Build pointing model (0), with ACS TCS:

- ***Start of test:*** 2000/12/05 19:30
- ***Actual test duration:*** 8h + 8h of analysis at daytime on 2000/12/06
- ***Start of test:*** 2000/12/06 21:30
- ***Actual test duration:*** 3h

Comments:

This test was actually split in four sessions.

In the first session we obtained "first light" and tested if the pointing model of 1999 was still performing with the old ECS TCS.

The following night we again obtained "first light" this time using the new AMS and the same pointing model of 1999. We spent a lot of time on this test, because at first we thought the performance was much worse than that with the ECS TCS. But in the end, we found that we were not looking at the same sky area and the behavior was actually the same.

On the following night we built a new pointing model, always starting from the 1999 values, but we did not obtain good results. The following day, we spent most of the time analyzing the results and comparing line by line the code of `ams` and `kptrk`. In the end, it was found that kppom was receiving corrected and not raw (ra,dec) coordinates for the object position from ams. This was due to an error in the configuration of the kpscanAms process.

The last night we generated another pointing model.

In any case, we never reached the pointing performance obtained in 1999, but not because of differences in the system behavior. The reason being that we did not put the same attention to carefully positioning the pointing targets on the CCD, and we did not acquire as many targets.

The purpose of our test was not get the best performances out of the 12m, but to verify the feasibility of AMS and ACS.

There are a few terms that should always be present, i.e. the pure geometrical terms **IA, IE, CA, AN, AW, TF** and **TX**.

Also **NPAE** and **ECES** should also be used. They should give good results, provided enough measurements are taken.

When these basic terms are introduced, it is necessary to look at residuals; plot them and try to identify any systematic behavior, any easily identifiable dependency of el and/or az.

As initial Pointing Model configuration we have used the final values of the 1999 test:

Term	Value	Sigma
IE	**+57.42**	**2.334**
IA	**+227.14**	**3.534**

CA	**+31.72**	**2.299**
AN	**+21.64**	**0.756**
AW	**-11.08**	**0.664**
TF	**+131.13**	**3.630**

Find first objects (POM-1)

Test Id	Command	Expected Result/Status	Comments	Result
POM1-3	**Start up Antenna**	**ONLINE/IDLE**		✓
POM1-4	**Update environmental constants from site monitor:** **Set ASM data in workstation database of tcsmon using ccseiDb** **temperature (C), pressure (MBAR), humidity (PCT)** **:Appl_data:TCS:tcsmon:control:environment.humidity** **:Appl_data:TCS:tcsmon:control:environment.pressure** **:Appl_data:TCS:tcsmon:control:environment.temperature**	**OK**		✓
POM1-5	**Verify ASM data on LCU** **GETASM/kptrkServer**	**set as above, updated every 3 min.**		✓
POM1-6	**Disable all pointing terms** **DISTERM/kptrkServer** **Verify with GETTERM**			✓
POM1-7	**Wait for moon to rise**			✓
POM1-8	**Preset to Apparent Coordinates of** **moon**	**Moon on CCD**		✓
POM1-9	**OFFSAD 3600 -> North**			✓
POM1-10	**OFFSAD 3600 -> South**			✓
POM1-11	**OFFSAD 3600 -> East**			✓
POM1-12	**OFFSAD 3600 -> West**			✓
POM1-13	**Select 5 very bright stars from Almanac**			✓

	evenly distributed over the sky and verify positioning of the antenna			
				✓

Build pointing model (POM-2)

The process to construct a pointing model is very interactive and iterative.

The following check list describes with as much details as possible the steps to be performed, but cannot cover all possibilities. It is not possible to actually follow it blindly step by step.

The construction of the pointing model consist of 3 phases:

- **Pointing error measure**
  In this phase the antenna is sent in sequence to a number of targets that cover as much as possible the visible sky. For each target, the pointing error is measured. Ideally the pointing error is measured automatically with software analyzing the CCD image. However, this is not possible on the 12m, and the antenna must be moved manually with offset steps until the target is on the reference point on the CCD display.
- **Pointing error fit**
  In this phase TPOINT is used to fit the data acquired and to generate the pointing model coefficients. This is a very iterative procedure and requires a good knowledge of pointing model theory and some hands on experience with the tools.
- **Pointing Model Installation**
  In this last phase, the pointing coefficients are installed in the real time system to be applied. The result is then verified by pointing the antenna first to stars already used to measure the pointing error, and then to new targets.

Test Id	Command	Expected Result/Status	Comments	Result
POM2-4	**Update environmental constants from site monitor:** **SETASM/trkwsContro l** **temperature (C), pressure (MBAR), humidity (PCT)**	**OK**		✓
POM2-5	**Verify ASM data on LCU** **GETASM/kptrkServer**	**set as above**		✓
POM2-5a	**Disable all pointing terms** **DISTERM/kptrkServer** **Verify with GETTERM**			✓
POM2-6	**run pomguiStart**	**This script starts al POM panels and processes**		✓
POM2-7	**load catalog hip-**			✓

	fk5.tcs			
POM2-8	**load template pomTemplate5.tpl**			✓
POM2-9	**Disable in Options: AO Corrections, Center Object**			✓
POM2-9a	**Set DUT in lower left field of panel** **or send SETDUT to trkwsControl**	**get it from maia.usno.navy.mil** **DUT1 = UT1-UTC**		✓
POM2-10	**Start Manual Control**			✓
POM2-11	**Preset to star**			✓
POM2-12	**Offset antenna to set star at reference position on CCD**			✓
POM2-13	**Store result**			✓
POM2-14	**Do that for all stars**			✓
POM2-15	**Select TPOINT terms**			✓
POM2-16	**Do fitting for current error file**			✓
POM2-17	**Install model**			✓
POM2-19	**Check that terms have been properly installed**			✓
POM2-20	**Preset to stars from catalogue**			✓
POM2-21	**Do pointing model again**			✓
POM2-22	**Save pointing model in Log**			✓
POM2-23	**Collect statistical data of pointing model and save it (e.g. rms)**			✓
POM2-24	**Select various stars from catalogue hip-fk5.tcs** **and measure offset from reference point.**			✓
POM2-25	**Select various Major planets and clusters to verify pointing model**			✓
POM2-26	**do pointing model with pomTemplate20.tpl**			✓

POM2-27	**do pointing model with pomTemplate50.tpl**			✓

8.7 Tracking Tests

Purpose: Collect data on tracking performance for optical tracking.

Allocated time: 12h

<u>Test execution:</u>

Start of test: **NOT PERFORMED**

Actual test duration:

Comments: We did **not** perform this test because we preferred to investigate in more detail the Pointing Modeling procedures and to test fixes to the Java user interfaces.

We performed some long time tracking tests, without following in detail the procedure and without measuring the offsets. Tracking proved to work reliably over long periods with little drift in target position. We were not interested in the absolute performance of the tracking system, but only in its qualitative behavior.

Test Id	Command	Expected Result/Status	Comments	Result
TRKW-1	**Preset to star from catalogue**			
	Let CCD expose			
	Move star to reference point			
	Let track for 30'			
	Move star to reference point			
	Record Offset ra/dec in arcsec			
TRKW-2	**Preset to star from catalogue**			
	Let CCD expose			
	Move star to reference point			
	Let track for 30'			
	Move star to reference point			
	Record Offset ra/dec in arcsec			
TRKW-3	**Preset to star from catalogue**			
	Let CCD expose			
	Move star to reference			

	point			
	Let track for 30'			
	Move star to reference point			
	Record Offset ra/dec in arcsec			
TRKW-4	**Preset to star from catalogue**			
	Let CCD expose			
	Move star to reference point			
	Let track for 30'			
	Move star to reference point			
	Record Offset ra/dec in arcsec			
TRKW-5	**Preset to star from catalogue**			
	Let CCD expose			
	Move star to reference point			
	Let track for 30'			
	Move star to reference point			
	Record Offset ra/dec in arcsec			

8.8 Problems and Solutions

Hardware

Several problems were discovered and corrected. The most major hardware problem was the bent pins on the back plane of the VME chassis. This required almost two days to find and correct.

ACS Core Libraries

- **PROBLEM: Devices are very sensitive to problems in clients - MUST BE FIXED WITH TOP PRIORITY**

If a client connected to a device has problems, if it is killed, core dumps, or if the network connection is lost or congested, the device behaves badly. Typically, there are huge memory leaks and high CPU consumption. This must be localized in monitor's implementation and lower BACI levels.

In no case shall devices/servants be affected by client or network problems.

STATUS: under investigation.

- **PROBLEM: Cannot stop activator - MUST BE FIXED WITH TOP PRIORITY**

The Activator process cannot be stopped in a clean way. There is already a Bugzilla SPR concerning this. There are major problems with CORBA memory management and they have to be deeply investigated, solved and, possibly, hidden from ACS users. It is really terrible to have to kill processes by hand because otherwise they core dump or lock. Looking at the code, now most delete for dynamically allocated objects have been commented out. This is OK for a prototype, but not for a real system.

STATUS: under investigation

- **PROBLEM: Memory leak with ESO::Completion return values. - MUST BE FIXED WITH TOP PRIORITY**

Whenever an ESO::Completion object is returned in a local CORBA call, we have a memory leak. The ROdouble::set_sync method() was called in the foreground loop to set the value of properties (logical properties need set method, used only by the device servant and not publicly available via the IDL). This was causing huge memory leaks. As a workaround we are now just directly writing in the database. The same behavior we have also with every local call that returns an ESO::Completion.

STATUS: under investigation

- **PROBLEM: Logging macros should be renamed according to Logging specifications.**

Now we are always using the COS_ERROR macro, also for LOGs and TRACEs. This is confusing.

STATUS: to be implemented for ACS 1.0

- **PROBLEM: Error handling**

Error handling is poor. We had planned to implement error handling using the error stack concept, but we had no time. This is really something missing and has to be implemented with high priority and retrofitted in all existing code.

STATUS: to be implemented for ACS 1.0

- **PROBLEM: Printfs and log messages**

All messages logged by maci and baci should go through our logging functions and do not use printf. The LOG type will send them to stdout as well. This is also important to have timestamp added.

STATUS: to be implemented for ACS 1.0

ACS User Interface libraries (Java and Tcl/Tk)

- **PROBLEM: Memory leaks with Java panels - MUST BE FIXED WITH TOP PRIORITY**

The AMS control panel uses 34Mb when it starts up, but after being connected to an AMS device, its footprint grows at the rate of 1Mb per minute. When it reaches 250Mb, the Linux workstation becomes to slow to continue working and we have to stop and restart the system.

In the panel there is no user code that could explain the memory leak, since the panel is fully visually generated.

The problem must the be either in:

- Core Java implementation
- Orbacus libraries
- Abeans code or Abeans ESO plug implementation

STATUS: Under inverstigation.

- **PROBLEM: Reconfiguring managerCORBALoc for panels requires editing file - MUST BE FIXED WITH TOP PRIORITY**

During the test period, are working all in parallel on the same machines. We have a pool of Linux and LCUs and we make tests in parallel.

We run several Manager/Activator/AMS devices on our various workstations and LCUs to be completely independent one from the other.

The problem is that when we start Java user interfaces based on ABeans, we have to modify by hand the DefaultAbean.data (or the app specific) file. On C++ clients, we just specify this on the command line (or with an ENV variable).

STATUS: A modified version of Abeans was implemented and tested. This allows to specify managerCORBALoc on the command line or a startup on a popup panel. We had anyway problems using it outside Visual Age, probably because of erroneusly generated .jar files.

- **PROBLEM: If a device is restarted, panels cannot re-connect. - MUST BE FIXED WITH TOP PRIORITY**

Actually we have only one AMS device instance per Manager. This means that the BeanSelector has only one entry. If we connect to the device and then the device is stopped and restarted we have no way to re-connect. We can only close the panel and restart it.

If the list contains more devices, we just switch to another one and back, but in our case it does not work, since we have nothing to switch to. The re-connect should be automatic. What was implemented by Matej (re-connect initiated by servant) does not work and is commented out. We would anyway prefer that the re-connect is initiated by the clients, not by the servants, for example polling at (low) configurable frequency to see if the servant is back alive.

REASON: The problem with the BeanSelector list seems duw to the version of Swing we are using. The newer Abeans do not show this problem.

STATUS: A modified version of Abeans was implemented and tested. With this new version it is possible to click on the "select" item in the list and then again on the device to select it. We had anyway problems using it outside Visual Age, probably because of erroneusly generated .jar files. Automatic re-connection is under investigation.

- **PROBLEM: If a panel cannot attach to the Manager or looses connection cannot exit - MUST BE FIXED WITH TOP PRIORITY**

If a panel is started with a non-existing manager in the managerCORBALoc or if the device to which it is attached exits or dies, the exit button does not work any more. The only way out is kill.

STATUS: A modified version of Abeans was implemented and tested. With this new version the first time the exit button is pressed, the panel tries to exit cleanly. If it does not manage, pressing exit a second time forces a hard exit, but in this case CORBA is not cleaned up. More investigations are needed.

- **PROBLEM: TCL and VxWorks connection problems**

TCL applications (Combat + MICO) work fine when talking to device servants on Linux workstations. When we try to talk to device servants on an LCU, no callback is delivered and even monitors do not work. Calls from TCL to VxWorks are OK. Just callbacks do not work. The servants do not produce and error diagnostic, so we cannot say what is going on. Seems a problem of communication VxWorks/TAO -> MICO.

STATUS: As a workaround, TCL panels actually poll the LCU instead of using monitor. The solution is to call the TCL client with the -ORBNoResolve option:

```
> client.tcl -ORBNoResolve
```

- **PROBLEM: When we call methods of devices, we do not get any feedback displayed on panels.**

We would like to have:

- A message on the GUI text pane when a method is executed, with method and parameters and the time in ISO format, like with logs). We have implemented this easily in the sending button ActionPerformed code, but should be handled automatically in the ABeans
- All working and done replies logged. Nothing done.

At best, it would be nice to have the possibility or changing the color of the command button in the time between sending of the command and getting the done reply.

STATUS: under investigation

AMS Device and User Interfaces

- **PROBLEM:** Memory leak with Java monitors on AMS.commandRa

When we have A JAVA PANEL attaching monitors (for example attaching to a dial widget) to the property

> AmsMount::commandRa

we notice on VxWorks (where it is easy to measure) that we loose about 6k per minute. The leak occurs with and only with commandRa. We have it from all panels, also the most simple panel just displaying that single property. With any other property we have tried (for example commandDec), we have no leak. With a C++ client attaching a monitor to the same property, we have no leak. We suspect a typo or a stupid error in the java classes for the AMS device of for the corresponding ABean, but we have not been able to find any. Other device classes (like SMCALC, sun and moon calculator) do not show any leak with monitors.

STATUS: We have no idea! Under investigation

- **PROBLEM: AMS on the LCU is not able to send replies to commands, i.e. cannot call callbacks.**

We call from clients methods of the AMS device and we wait for replies, i.e. callbacks. Everything goes fine for a while, but at a certain point we stop getting the callback called. The code to send the callback is executed and we do not seem to have any error, but the callback is not delivered. Restarting the client does not help. We have to investigate more to have a more precise description of the phenomenon.

STATUS: under investigation

- PROBLEM: Dartboard: Moon and sun icons do not disappear from the screen when the astronomical object disappear below horizon.

Instead they remain on the dartboard.

STATUS: Minor problem. Nothing to be done for the prototype

ESO CCSLite for Linux

- **PROBLEM: Linux CCSLite: Activator core dumps on Linux. - MUST BE FIXED WITH TOP PRIORITY**

When the Linux host is subject to significant load, the Activator very often crashes. The error "fatal flex scanner internal error - end of buffer missed" is displayed. In the CCS logMonitor appear DB read fails for attributes that are fully correct. This seems a Linux CCSLite problem and has to be investigated.

STATUS: to be investigated. The problem could be that ACS is using the CCSLite database from multithreading processes and access is not rigorously guarded with semaphores to allow better performance. Tests have to be done warranting that only one thread at the time can call database access functions or CCSLite must be verified for safe multithreading support.

- **PROBLEM: Error messages from an LCU handled incorrectly**

When a Linux application receives an error reply is received from an LCU application, the message buffer is corrupted. The client seems to receive a normal (non error) reply and the following message is logged by the message system: "*message buffer too big*".

STATUS: to be investigated. This could be caused by an incompatibility between the Linux and the VxWorks message systems, that are not of the same versions (Linux implementation is quite old, since it correspond to a NOV2000 pre-release built in June 2000) .

- **PROBLEM: cmdSetup command seems not to be working. We are not able to unload/reload a CDT.**

STATUS: to be investigated

Development tools

- **PROBLEM: It is not possible to use GDB on Linux with ACE/TAO maci/baci. - MUST BE FIXED WITH TOP PRIORITY**

ACE throws an unknown interrupt and we cannot attach to the proper thread. It is also not possible to use the Tornado debugger (CrossWind) when running on the LCU (CrossWind is based on GDB). THIS IS A MAJOR PROBLEM, since nowadays it is not reasonable to debug with printfs, in particular when you have 20Hz control loops.

STATUS: GDB 5.0 supports debugging of multi threading applications under Linux

- **PROBLEM: The Tornado tools (browser, debugger, host shell) work intermittently or sometimes not at all.**

These tools make development on the LCU much easier. It would also be nice to use Tornado Project in some integrated way. With Tornado 3 (just released) it may impossible to do development without using Project.

STATUS: to be investigated

8.9 Final Remarks

The following section presents final remarks and comments on the Kitt Peak Test from all the people directly involved in it.

G. Chiozzi

This test project has reached its objectives, since we have been able during the test period in Kitt Peak to perform successfully all the tests planned.

We can then reaffirm that the ACS prototype and the AMS software built on top of it have demonstrated that the technology chosen and the ACS architecture are adequate for the foreseen role in the ALMA project.

We had anyway a significant number of (non-stopping) problems and some of them have to be urgently solved for the first official release of ACS. This is to remind us that ACS 0.0 is a prototype and a lot of work is still necessary to get a real system out of it.

In this respect, the underlying presence of the ESO Common Software has helped in working around the deficiencies of ACS, for example providing stable and tested tools to browse the configuration database, the real time value of properties and to sample LCU data.

I am personally fully satisfied by CORBA. In particular, interoperability between different ORBs was fully demonstrated.

Linux works reasonably well and I do not expect any major problem in that area, is we can keep under control the operating system versions and configurations supported by ACS.

I have some concerns regarding Java. The language in itself and the libraries available are very nice and I have no complaints on that. On the other hand, the user interfaces produced are nice, but they use a huge amount of memory, are notably slow and we have experienced unexpected and unreasonable memory leaks. Visual Age has very nice visual programming concepts, but the Linux implementation is poor and does not support JDK 1.2. It is necessary to switch to Visual Age on Windows and upgrade to JDK 1.2. This should mitigate the above-mentioned problems.

I have found the software engineering practices used for this project perfectly adequate and reliable, but I am used to that and as a consequence my opinion is of relative value.

The most important remark is that ACS (like any other system of this kind) is complex bringing with it some of the complexity of CORBA. Alain and Ron have done a great job in implementing AMS in 4 months of work, and AMS has proven itself stable and reliable. This has required a lot of dedication from them, and a lot of support from the ACS team.

The lack of documentation is reasonable for a prototype. For ACS 1.0 it will be necessary to provide good documentation, able to cover both tutorial and reference purposes. In my opinion the quality of the documentation will be a key factor in the success of ACS, much more than the features provided with the implementation of release 1.0.

Mentoring, tutoring and online support will be also a key factor and the ACS team must ready to plan major quotas of their time for these activities.

Due to lack of time, we had to scrap from the ACS prototype two important areas that were in the original planning presented in December 1999:

- Error handling
- Modular testing

It is very important to cover these areas for ACS 1.0.

Another major aspect of this project was the positive effect on the interrelations between the groups involved in ALMA. Working all together, from different sites, on a project with near and rigid deadlines has tightened the relations and built team spirit. This alone is in my opinion enough to justify the project and to suggest a repeat experience with similarly structured projects in the future.

B. Gustafsson

I think Gianluca already said most what has to be said.

I like to add also that with the help of ACS and Corba the code is really portable.

To port ACS from Linux/Unix to VxWorks was really simple. We had to make very few changes to the code. The main problem with VxWorks is that the

present compiler does not support name spaces. Also we had problems with native exceptions, but by using ACE exceptions this problem could be circumvented.

Another area we should investigate more is the use of 64 bit integer for time representation. This did not work properly for VxWorks.

One problem we had in particular on VxWorks was memory leaks. These problems were also present under Linux, but due to less memory on the VxWorks targets and a flat memory structure this caused more problems.

Another problem with VxWorks is that you have to reboot the system when restarting the CORBA server. This is due to usage of global data in TAO.

The most positive with the test was the cooperation with NRAO and IRAM.

I think this gave the feeling that we are one team working on the Alma project and not different groups competing with their contribution.

R. Heald

I think Gianluca and Alain have, for the most part, have covered my thoughts and conclusions of the test. I will add only a couple of items.

One thing I believe is clear is the ACS asynchronous call method (using callbacks) is too complex, and further more, it is unnecessary. I would guess that the purpose of nearly a third of the AMS code is to handle the callbacks. This is too much for the results obtained. There are just not many cases where callers need to know when a command has completed (i.e. asynchronous calls) are needed. When there is a case multiple processes or the select() function can be used instead.

I would also add that there were many intangible results of the tests. As Gianluca said, the environment greatly improved relations among the team members, and a certain team spirit was established. I believe it provided a future model for cooperation between members of the software team. This should definitely be included any decision on future tests at the 12m.

R. Karban

The results of the ACS test on the 12m Kitt Peak telescope are very satisfying.

We have used a completely new tracking software based on the prototype ACS, that uses many new concepts and implementations, and combined it with former generation applications of the VLT and the former ESO common software.

Having done this, is in my opinion quite impressive.

Not only this: it also worked very well and did what was specified for the tests.

The tests have proven the various concepts of ACS and their applicability in a real world example.

If it was CORBA (and its various implementations used during the test) as middle-ware, Java and Java Beans as a GUI concept, that can be easily integrated with IDL interfaces of the ACS applications, or the very well integrated CCSlite real-time database as a configuration database, which has shown that it is still a state of the art concept and implementation for this particular purpose.

In general I can say that the test convinced me of the concepts used by ACS (e.g. properties, characteristics, monitors, Corba, Maci, Baci, Java, etc.) but everything apart of the CCSlite database has some implemention weaknesses.

Corba has the least weaknesses in terms of implementations, although I have to say that C++ applications using Corba are difficult to read and manage. This really cries for some easy to use framework on top of it. The main reason is the confusing memory management.

Java Corba applications give a much cleaner impression, because Java takes care of the memory management, at least when facing the user. Probably not in the Java internals. From Java I'm a bit disappointed in terms of implementation and in terms of tools. Java memory leaks and the Java environment (class paths etc.) are a nightmare. But I hope that this is only a problem of the Java version we used during the test. There is good hope that newer releases of Java will overcome this problems.

VisualAge as the Java IDE is conceptually really great, but too slow. Connecting visual components with ACS applications using the IDL is really a snap.

The core ACS (MACI, BACI, etc.) is very promising but needs some cleanup and improvements in the implementation.

Linux as an operating system looks very weak to me. In particular when several people use it for compiling and operating the system it turns out to be very very weak. Frozen screens during compilation are not state of the art. The only advantage of Linux is the cheap hardware and when used as a single user system it can handle the things quite well.

Anyway I would prefer to have only one OS. Now we use SUN for VxWorks development and Linux for the rest. I would scrap either Linux at all and do everything on SUN, or when VxWorks is available on Linux check carefully the feasibility of Linux and then use only Linux.

To conclude I think we are on the right track for ACS and should continue that way.

A. Perrigouard

The purpose of this test was to demonstrate in a clear way the possibility of using the new techniques proposed for ALMA to control a single dish. It was clear in the sense the optical camera showed the stars that were targeted and the pointing model coefficients were successfully determined.

But before this final demonstration we knew that this new technique of distributed objects was mature enough for the project.

The aim of this test was to check the first release of ACS based on CORBA and also to test some of the software engineering rules.

This first release was reliable and complete enough for this test but one can draw those remarks.

ACS is very complex and very large in term of disk spaces, and number of lines of code to understand and maintain. Today some IDL types defined in ACS are not yet functional.

But above all, good documentation with simple examples to complete the 2 or 3 existing applications is missing. Today these applications are used as good tutorials and the ACS architecture draft document is used as reference. Before implementing everything foreseen in the draft document, we need accurate documentation describing the current release. Some programmers will be included later in the project and will not have the experiences of those involved in it from the beginning.

This documentation should consider the different programming languages mapped to CORBA and selected for ALMA, which include C++, Java and maybe TCL/TK.

Another point is the debugging. CORBA and ACS will be easier to implement with an adequate debugger.

No specific tool was used to develop a TCL/TK based GUI. The Java based GUI was developed with the toolkit Visual Age for Java from IBM. For all GUIs it would be more efficient to use the appropriate toolkit with a complete tutorial adapted to our environment including ACS/CORBA. Birger made a good presentation for Visual Age that can be adapted to be an excellent readable document on paper.

Considering software engineering after a long rest period following the test, I think it is worthy to use and to test cmm for archiving the code development. There were other alternatives but no real volunteers to promote and implement them. So let's start with cmm.

The working environment with many different directories and the complex makefiles are hard to understood from the beginning for non-ESO people. It has the advantage of already existing, but time should be spent to explain (written document) its reason and efficient use.

9 ACKNOWLEDGMENTS

The following picture, taken at the 12m shows the team that actually performed the test, but many other people have been involved and deserve a big THANKS from all of us.

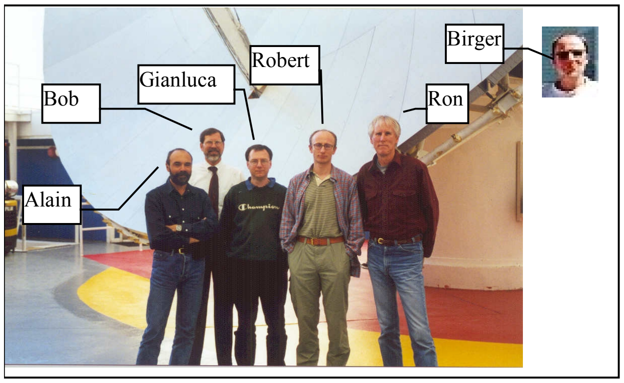


Figure 14 - The KP2000 test team

In particular we would like to thank B. Jeram, P. Sivera and G. Filippi from ESO, R. Lemke from the AIRUBMP Bochum and the whole "Kontrol Gruppe für Beschleuniger" (KGB) team of the IJS Institute in Ljubljiana. They all gave an essential contribution to the development of the ACS 0.0 prototype.

10 APPENDICES

10.1 Software Installation Procedure

This section describes how to build the ACS/AMS software system used for the Kitt Peak test.

ACS build, based on pkgin

This section describes how to install ACS 0.0, starting from the official distribution CD, the ACS Installation Manual, Rev 0.0, G. Chiozzi, P. Sivera, G. Filippi, and applying the patches that are necessary for the Kitt Peak configuration.

1) **Linux:** install FULL Red Hat 6.1 or 6.2 (for 6.2 check notes on ACS Installation Manual par 3.1).
   Sun: it is assumed that Sun Solaris 2.6 and ESO Common Software NOV2000 have been already installed. Go directly at step 3.
2) Install VLTROOT following instructions on the ACS Installation manual[2]:
   - 3.2
   - 3.3
   - 3.4
3) Install the module `pecsalmasw 1.9`[3], logout and login.

[2] **NOTE: Do not login in a GNOME environment** when you perform steps 2 and 3, but on a console, on KDE or on anything else. At this point there is a naming clash with the GNOME panel executable and the panel application of the ESO Common Software. The installation procedure fixes this problem and you will be able again to use GNOME later.

4) repeat step 3.4 of the installation procedure

The following steps replace for the time being section 3.5 if the installation procedure:

5) **Linux**: install pkgin 1.55, cfitsio 1.2, seq 2.51.1.1, vcc 2.46.1.1 in VLTROOT
Sun: install vcc 2.46.1.1 in VLTROOT (for mv2700 support)

6) logout and login

7) `> cd ~/`

8) `> mkdir ACSSW`

9) `> cd ACSSW`

10) `> cmmCopy acssw 1.19`

11) `> export ACSROOT=<new ACSROOT>   i.e. /alma/OCT2000/ACSSW`

`> export INTROOT=$ACSROOT`

12) `pkginBuild acssw`

You can now continue from section 4 of the installation procedure.

Kitt Peak TCS and AMS installation procedure

This section describes how to install the Kitt Peak TCS (as from 1999 test) and AMS software.

The complete installation and configuration of the KP TCS system is done using the integration module kptcsBUILD. This module is used to build the simulation setup in Socorro and Garching as well as the real mode setup at Kitt Peak.

The selection of which system is built is controlled by the Makefile variable TCSID. It is TCSID=1 for the 12m. This environment variable must be specified in the environment of the operations account. The operations and installation account is **almamgr**.

If a test installation is done, with no control electronics or even with no LCUs, TCSID=0 is a good value to use. For a test installation without LCUs is also not necessary to have a sun cross-development machine. In this case, install only the Linux part and skip all steps related to sun installation and configuration.

1) setup environment in `~/.pecs/apps-all.env adding the definition for the following environment variables:`
```
export TCSID=x (use Socorro for an installation without LCU)
   x=0 Socorro
   x=1 Kitt Peak
   x=2 Tucson
   x=3 Charlottesville
 export TARGET_TCS=KP_TCS
 export TZ=UTC
 export ACC_HOST=<the hostname of the sun>
```

[3] **NOTE:** "Installing" a patch for a module means to retrieve it from the cmm archive, building and installing:

```
prompt> cd MODULES
```

(or any directory where you want to put the copy from the archive)

```
prompt> cmmCopy <module name> <version>

prompt> cd <module name>/src; make clean all install
```

In a test installation, ACC_HOST is typically the same Linux workstation.
Add in accData.sql: lalxt (test setup, simulation), lalxc (control,all hardware available), where x is TCSIS. This is not necessary on an installation without LCUs
on Linux: export RTAPENV=walma
on SUN: create default environment, w<sun>,
export RTAPENV=w<sun>
> cmmModify kptcsBUILD
edit kptcsBUILD/ENVIRONMENTS/lalxt/userScript.lalxt.replace to add w<sun> in lqs table
edit kptcsBUILD/ENVIRONMENTS/lalxc/userScript.lalxc.replace to add w<sun> in lqs table

2) `> cd ~/`

3) `> mkdir TCS_SOURCE`

4) `> cd TCS_SOURCE`

5) `> cmmCopy kptcsBUILD 1.35`

6) `> pkginVersions kptcsBUILD`
Check if some modules need updates.
If so `cmmModify kptcsBUILD` and update accordingly

7) **Linux:**
```
 > export INTROOT=~/introot
 > pkginBuild kptcsBUILD -tostep BUILD_MOD
 > pkginBuild kptcsBUILD -step BUILD_ENV -env walma
```
Sun:
```
 > export INTROOT=~/introot.sun[4]
 > pkginBuild kptcsBUILD -tostep BUILD_MOD
 > pkginBuild kptcsBUILD -step BUILD_ENV -env lalxt
 >pkginBuild kptcsBUILD -step BUILD_ENV -env lalxc
```

8) start workstation environments on Linux and sun (is used) with vccEnvStart

9) boot LCU(s), if used

10.2 Configuration data in Database

Site constants (for Kitt Peak)

Attribute	Description	Value	Notes
longitude		**1.9480466993**	**Unit: radian** **astronomical convention west-positive**
latitude		**0.5576908252**	**Unit: radian**

[4] **NOTE:** Watch how INTROOT is defined. The ACS installation procedure, drives you to define an INTROOT named ~/introot. If you are installing ACS for both Linux and Sun, you have to be careful, since the two INTROOTS are different and cannot refer to the same physical directory. In this case, you can call ~/introot.sun the INTROOT for the SUN machine, but you have to do it consistently all over the installation steps on Sun. What you have to actually do depends finally on your disks and NFS configuration.

height	**height above sea level in meter**	**1914**	**Unit: meter**

Assumption: latitude = -111 36' 53.475" (East), longitude = +31 57' 11.99" (North)

Convention used by slalib: South is negative, East is positive

Environment assumptions for test period

Attribute	Description	Value	Notes
temperature	**environment temperature**	**10.0**	**Unit: Celsius**
pressure	**environment pressure**	**750**	**Unit: Pascal**
humidity	**relative humidity**	**0.12**	**Unit: %**

Configuration of axes

Attribute	Description	Value	Notes
posLims:az	**0: outer lower limit in rad**	**0**	**not used**
	1: low vicinity limit in rad	**-6.28**	
	2: high vicinity limit in rad	**6.28**	
	3: outer upper limit in rad	**0**	**no used**
posLims:el	**0: outer lower limit in rad**		**not used**
	1: low vicinity limit in rad	**0.35**	
	2: high vicinity limit in rad	**1.5708**	
	3: outer upper limit in rad		**not used**
TrkRadius	**maximum position error in rad to switch from tracking to Presetting**	**4.848136e-05**	

10.3 kptcsBUILD configuration file

```
#*****************************************************************************
# ALMA project
#
# "@(#) $Id: kptcsBUILDINSTALL.cfg,v 1.35 2001/02/05 11:59:37 vltsccm Exp $"
#
# who       when      what
# --------  --------  ----------------------------------------------
# rkarban 2000-12-04 created
#

PAF.HDR.START;                  # Start of PAF Header
PAF.TYPE         "Configuration"; # Type of PAF
PAF.ID           "        ";   # ID for PAF
PAF.NAME         "        ";   # Name of PAF
PAF.DESC         "        ";   # Short description of PAF
PAF.CRTE.NAME    "        ";   # Name of creator
PAF.LCHG.DAYTIM  "        ";   # Timestamp of last change
PAF.CHCK.NAME    "        ";   # Name of appl. checking
PAF.HDR.END;                    # End of PAF Header
```

```
INSTALL.PACKAGE.NAME         "KPTCS"
INSTALL.LOGINNAME.USER       "$USER"
INSTALL.LOGINNAME.USERMGR    "$USER"
INSTALL.MODULE.CPU           "PPC604"

######################################################
# Modules
######################################################


# put VXWORKS to avoid building on Linux
INSTALL.MODULE13.NAME         "lcc3"
INSTALL.MODULE13.VERSION      "4.36.1.1 FORCED"
INSTALL.MODULE13.OPTIONS      "VXWORKS IGNORE_WARNINGS CLEAN"
INSTALL.MODULE13.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE13.SUBPKG       "VLTSW_new"

#
##############################################################
#
INSTALL.MODULE30.NAME         "kpacm"
INSTALL.MODULE30.VERSION      "1.2"
INSTALL.MODULE30.OPTIONS      "VXWORKS CLEAN"
INSTALL.MODULE30.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE30.SUBPKG       "TCS"

INSTALL.MODULE31.NAME         "vlb"
INSTALL.MODULE31.VERSION      "1.0"
INSTALL.MODULE31.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE31.OPTIONS      "VXWORKS IGNORE_WARNINGS CLEAN"
INSTALL.MODULE31.SUBPKG       "TCS"

INSTALL.MODULE32.NAME         "kpam"
INSTALL.MODULE32.VERSION      "1.16"
INSTALL.MODULE31.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE32.OPTIONS      "VXWORKS_LCU IGNORE_WARNINGS CLEAN"
INSTALL.MODULE32.SUBPKG       "TCS"

INSTALL.MODULE33.NAME         "ams"
INSTALL.MODULE33.VERSION      "1.21"
INSTALL.MODULE33.OPTIONS      "VXWORKS_LCU IGNORE_WARNINGS CLEAN"
INSTALL.MODULE33.SUBPKG       "TCS"


INSTALL.MODULE35.NAME         "kptcs"
INSTALL.MODULE35.VERSION      "1.8"
INSTALL.MODULE35.OPTIONS      "VXWORKS_LCU IGNORE_WARNINGS CLEAN"
INSTALL.MODULE35.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE35.SUBPKG       "TCS"

INSTALL.MODULE36.NAME         "kpmsw"
INSTALL.MODULE36.VERSION      "1.3"
INSTALL.MODULE36.OPTIONS      "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE36.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE36.SUBPKG       "TCS"

INSTALL.MODULE37.NAME         "kptif"
INSTALL.MODULE37.VERSION      "1.3"
INSTALL.MODULE37.OPTIONS      "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE37.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE37.SUBPKG       "TCS"

INSTALL.MODULE38.NAME         "kptrkws"
INSTALL.MODULE38.VERSION      "1.10"
INSTALL.MODULE38.OPTIONS      "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE38.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE38.SUBPKG       "TCS"

INSTALL.MODULE39.NAME         "kpprs"
INSTALL.MODULE39.VERSION      "1.3"
INSTALL.MODULE39.OPTIONS      "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE39.MAKE         "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE39.SUBPKG       "TCS"
INSTALL.MODULE40.NAME        "kppom"
INSTALL.MODULE40.VERSION     "1.7"
INSTALL.MODULE40.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE40.MAKE        "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE40.SUBPKG      "TCS"

INSTALL.MODULE41.NAME        "kptcsmon"
INSTALL.MODULE41.VERSION     "1.2"
INSTALL.MODULE41.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE41.MAKE        "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE41.SUBPKG      "TCS"

INSTALL.MODULE42.NAME        "kpamv"
INSTALL.MODULE42.VERSION     "1.11"
INSTALL.MODULE42.MAKE        "-k ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE42.OPTIONS     "VXWORKS_LCU IGNORE_WARNINGS CLEAN"
INSTALL.MODULE42.SUBPKG      "TCS"

INSTALL.MODULE43.NAME        "kpgui"
INSTALL.MODULE43.VERSION     "1.14"
INSTALL.MODULE43.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE43.SUBPKG      "TCS"

INSTALL.MODULE44.NAME        "kposf"
INSTALL.MODULE44.VERSION     "1.3"
INSTALL.MODULE44.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE44.SUBPKG      "TCS"

INSTALL.MODULE45.NAME        "tcscam"
INSTALL.MODULE45.VERSION     "2.7 FORCED"
INSTALL.MODULE45.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE45.SUBPKG      "TCS"

INSTALL.MODULE46.NAME        "kpsamp"
INSTALL.MODULE46.VERSION     "1.3"
INSTALL.MODULE46.OPTIONS     "VXWORKS_LCU CLEAN"
INSTALL.MODULE46.MAKE        "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE46.SUBPKG      "TCS"

INSTALL.MODULE47.NAME        "kpscan"
INSTALL.MODULE47.VERSION     "1.6"
INSTALL.MODULE47.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE47.MAKE        "ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE47.SUBPKG      "TCS"

INSTALL.MODULE48.NAME        "kptools"
INSTALL.MODULE48.VERSION     "1.3"
INSTALL.MODULE48.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE48.SUBPKG      "TCS"

INSTALL.MODULE49.NAME        "kptrk"
INSTALL.MODULE49.VERSION     "1.9"
INSTALL.MODULE49.OPTIONS     "VXWORKS IGNORE_WARNINGS CLEAN"
INSTALL.MODULE49.MAKE        "-k ACE_ROOT_DIR= ACE_ROOT= clean all install"
INSTALL.MODULE49.SUBPKG      "TCS"

INSTALL.MODULE50.NAME        "smcalc"
INSTALL.MODULE50.VERSION     "1.7"
INSTALL.MODULE50.OPTIONS     "VXWORKS_LCU IGNORE_WARNINGS CLEAN"
INSTALL.MODULE50.SUBPKG      "TCS"

INSTALL.MODULE51.NAME        "kpcmdtrans"
INSTALL.MODULE51.VERSION     "1.4"
INSTALL.MODULE51.OPTIONS     "IGNORE_WARNINGS CLEAN"
INSTALL.MODULE51.SUBPKG      "TCS"

INSTALL.MODULE52.NAME        "amsclient"
INSTALL.MODULE52.VERSION     "1.4"
INSTALL.MODULE52.SUBPKG      "TCS"

#####################################################
# Environments
#####################################################

INSTALL.LCUENV1.NAME         "lal$(TCSID)c"
INSTALL.LCUENV1.TPLSRC       "$(INTROOT)/ENVIRONMENTS/lalxc"
INSTALL.LCUENV2.NAME         "lal$(TCSID)t"
INSTALL.LCUENV2.TPLSRC       "$(INTROOT)/ENVIRONMENTS/lalxt"

INSTALL.RTAPENV1.NAME        "walma"
INSTALL.RTAPENV1.TPLSRC      "$(INTROOT)/ENVIRONMENTS/walma"

#######################################################
# scan links
#######################################################
```

10.4 AMS IDL

```
#ifndef _AMS_IDL_
#define _AMS_IDL_

#include "baci.idl"

/**
 * An IDL interface to the Antenna Mount System (AMS).
 * This CORBA IDL interface exists between two sections of the AMS.  The
 * high-level section executes at the array central control area in the Array
 * Control Computer (ACC), while the low-level section executes at each
 * antenna in the Antenna Bus Master (ABM) computer.
 *
 * @authors R. Heald, Modifed by A. Perrigouard
 * @version %I%, %G%
 */
module AMS {

  /**
   * Defines the interface for controlling and monitoring movement of the
   * antenna.
   *
   * Callbacks can return two completion types, either successful or aborted.
   */
  interface AmsMount : ESO::Device
  {

    enum coordType { Mean, Apparent };

    /**
     * (Pre)sets a new equatorial source for the antenna to track.
     * The source position is given in J2000 equatorial coordinate
     * system.  The antenna goes to Moving status (if not already
     * there), moves to the current source position, and begins tracking
     * the source.  The pointing model is always applied.
     * <p>
     * A callback is used to inform the caller when the antenna has
     * reached the source position (within the pointing tolerance).  This
     * includes any offset that may have been set using <code>offsad</code>
     * or <code>offsaa</code>.  If the method is called again before the
     * source given in the previous call is reached, then the previous
     * callback immediately receives an "aborted" completion.
     *
     * @param ra        source right ascension (hour)
     * @param dec       source declination (degree)
     * @param epoch     source Julian epoch (years)
     * @param pmRa      source sky proper motion in right ascension
     *                  (arc-sec/year)
     * @param pmDec     source sky proper motion in declination
     *                  (arc-sec/year)
     * @param radVel    source radial velocity (kilometer/sec)
     * @param par       source parallax correction (arc-sec)
     * @param type      Mean or Apparent
     * @param callBack  callback when position is reached
     * @param CBdesc    contains timeout and callback tag
     */
    void objstar (
         in double ra,
         in double dec,
         in double epoch,
         in double pmRa,
         in double pmDec,
         in double radVel,
         in double par,
         in coordType type,
         in ESO::CBvoid callBack,
         in ESO::CBDescIn CBdesc
         );

    /**
     * (Pre)sets a new non-moving position for the antenna.
     * The position coordinates are given in azimuth and elevation.  The
     * antenna goes to Moving status (if not already there), moves to
     * the given position, and goes to Stopped status.
     * <p>
     * Notice the pointing model is NOT applied.
     * <p>
     * A callback is used to inform the caller when the antenna has reached
     * the new position.  If the method is called before the position
     * given in the previous call is reached, then the previous callback
     * immediately receives "aborted".
     *
     * @param az        position azimuth (degree)
     * @param elev      position elevation (degree)
     * @param callBack  callback when position is reached
     * @param CBdesc    contains timeout and callback tag
     */
    void objfix (
        in double az,
        in double elev,
        in ESO::CBvoid callBack,
        in ESO::CBDescIn CBdesc
        );

    /**
     * Adds an equatorial coordinate offset.
     * The status must be Moving, otherwise an error is returned.
     * Offsets are accumulated and the total offsets are added to the
     * apparent antenna position.  The total offsets are cleared whenever
     * the <code>objstar</code> or <code>objfix</code> method is called.
     * <p>
     * A callback is used to inform the caller when the correct position is
     * reached.  If the method is called before the position given in the
     * previous call is reached, then the previous callback immediately
     * receives "completed".
     *
     * @param ra        offset right ascension (arc-sec)
     * @param dec       offset declination (arc-sec)
     * @param callBack  callback when position is reached
     * @param CBdesc    contains timeout and callback tag
     */
    void offsad (
        in double ra,
        in double dec,
        in ESO::CBvoid callBack,
        in ESO::CBDescIn CBdesc
        );

    /**
     * Adds a horizon coordinate offset.
     * Offsets are accumulated and the total offsets are added to the
     * antenna position.  The total offsets are cleared whenever the
     * <code>objstar</code> or <code>objfix</code> method is called.
     * <p>
     * A callback is used to inform the caller when the correct position is
     * reached.  If the method is called before the position given in the
     * previous call is reached, then the previous callback immediately
     * receives "completed".
     *
     * @param az        offset azimuth (arc-sec)
     * @param elev      offset elevation (arc-sec)
     * @param callBack  callback when position is reached
     * @param CBdesc    contains timeout and callback tag
     */
    void offsaa (
        in double az,
        in double elev,
        in ESO::CBvoid callBack,
        in ESO::CBDescIn CBdesc
        );

    /**
     * Sets antenna additional velocities.
     * This method allows the setting of velocities that are added to the
     * current tracking.  These are normally used when tracking a moving
     * object.  NOTICE THERE ARE BETTER METHODS FOR TRACKING MOVING OBJECTS
     * AND THE IMPLEMENTATION OF THIS METHOD IS LOW PRIORITY.
     *
     * @param ra        right ascension additional velocity (arc-sec/sec)
     * @param dec       declination additional velocity (arc-sec/sec)
     * @param callBack  callback when the command is accepted
     * @param CBdesc    contains timeout and callback tag
     */
    void setav (
       in double ra,
       in double dec,
       in ESO::CBvoid callBack,
       in ESO::CBDescIn CBdesc
       );

    /**
     * Returns time-tagged actual antenna position.
     * This method supplies inputs for the reverse calculation of the
     * actual right ascension and declination.  These are used for display
     * only; as a comparison with the commanded right ascension and
     * declination.  THE IMPLEMENTATION OF THIS METHOD IS LOW PRIORITY.
     *
     * @param az        returned actual azimuth, approximate range
     *                    +270 to -270 (degree)
     * @param elev      returned actual elevation, approximate range
     *                    0 to 125 (degree)
     * @param azCor     returned actual azimuth with pointing model and
     *                    refraction corrections removed, approximate
     *                    range +270 to -270 (degree)
     * @param elevCor   returned actual elevation with pointing model and
     *                    refraction corrections removed, approximate
     *                    range 0 to 125 (degree)
     * @param time      returned time tag (MJD)
     */
    void getpos (
        out double az,
        out double elev,
        out double azCor,
        out double elevCor,
        out double time
        );

    /**
     * Possible status conditions.
     * <ul>
     * <li>Off     - antenna stopped, brakes engaged<br>
     * <li>Stopped - antenna stopped, brakes released<br>
     * <li>Moving  - antenna is moving<br>
     * <li>Error   - error condition exists<br>
     * </ul>
     */
    enum status { Off, Stopped, Moving, Error };

    /**
     * Stops any antenna movement.
     * Antenna movement is started using <code>objstar</code> or <code>
     * objfix</code> methods.
     *
     * @param flag      Off or Stopped
     * @param callBack  callback when the command is accepted
     * @param CBdesc    contains timeout and callback tag
     */
    void stopMotion (
            in long stoped,
            in ESO::CBvoid callBack,
            in ESO::CBDescIn CBdesc
            );

    /**
     * Antenna control status.
     * Maybe any member of enum status.
     */
    readonly attribute ESO::ROlong antennaStatus;

    /**
     * ON/OFF source position tolerance (arc-sec).
     * "ON" source is declared whenever the vector sum of the azimuth and
     * elevation errors is within this tolerance.  Otherwise "OFF" source
     * is declared.
     */
    readonly attribute ESO::RWdouble tolerance;

    /**
     * Possible on target flag value.
     * <ul>
     * <li>No     - Off target<br>
     * <li>Yes    - On target<br>
     * </ul>
     */
    enum onTargetflag { No, Yes };

    /**
     * On target flag
     */
    readonly attribute ESO::ROlong onTarget;

    /**
     * Current commanded azimuth (degree)
     * Approximate range +270 to -270.
     */
    readonly attribute ESO::ROdouble commandAz;

    /**
     * Current commanded elevation (degree).
     * Approximate range 0 to 125 (2.18 rad).
     */
    readonly attribute ESO::ROdouble commandEl;

    /**
     * Current actual azimuth (degree).
     * Approximate range +270 to -270.
     */
    readonly attribute ESO::ROdouble actualAz;

    /**
     * Current actual elevation (degree).
     * Approximate range 0 to 125 (2.18 rad).
     */
    readonly attribute ESO::ROdouble actualEl;

    /**
     * Current actual azimuth velocity (degree/s).
     * Approximate range +4 to -4.
     */
    readonly attribute ESO::ROdouble actualVelocityAz;

    /**
     * Current actual elevation velocity (degree/s).
     * Approximate range +4 to -4.
     */
    readonly attribute ESO::ROdouble actualVelocityEl;

    /**
     * Current commanded right ascension (hour)
     */
    readonly attribute ESO::ROdouble commandRa;

    /**
     * Current commanded declination (degree)
     */
    readonly attribute ESO::ROdouble commandDec;

    /**
     * Current source Julian epoch
     * (year)
     */
    readonly attribute ESO::ROdouble epoch;

    /**
     * Current commanded proper motion in right ascension
     * (arc-sec/year)
     */
    readonly attribute ESO::ROdouble pmRa;

    /**
     * Current commanded proper motion in declination
     * (arc-sec/year)
     */
    readonly attribute ESO::ROdouble pmDec;

    /**
     * Current commanded radial velocity (kilometer/sec)
     */
    readonly attribute ESO::ROdouble radVel;

    /**
     * Current commanded parallax correction (arc-sec)
     */
    readonly attribute ESO::ROdouble parallax;

    /**
     * Current commanded coordinate type (Mean or Apparent)
     */
    readonly attribute ESO::ROlong coordinateType;

    /**
     * Current actual right ascension (hour)
     * deduced from encoder without refraction
     * Approximate range 0 to 24.
     */
    readonly attribute ESO::ROdouble encoderRa;

    /**
     * Current actual declination (degree)
     * deduced from encoder without refraction
     * Approximate range -90 to 90.
     */
    readonly attribute ESO::ROdouble encoderDec;

    /**
     * Current actual right ascension (hour).
     * deduced from encoder, without pointing model and in J2000
     * Approximate range 0 to 24.
     */
    readonly attribute ESO::ROdouble actualRa;

    /**
     * Current actual declination (degree).
     * deduced from encoder, without pointing model and in J2000
     * Approximate range -90 to 90.
     */
    readonly attribute ESO::ROdouble actualDec;

    /**
     * Current actual right ascension deviation (arc-sec)
     * actual minus commanded positions times cosine declination
     */
    readonly attribute ESO::ROdouble deviationRa;

    /**
     * Current actual declination deviation (arc-sec)
     * actual minus commanded positions
     */
    readonly attribute ESO::ROdouble deviationDec;

    /**
     * Current offset azimuth (arc-sec)
     */
    readonly attribute ESO::ROdouble offsetAz;

    /**
     * Current offset elevation (arc-sec)
     */
    readonly attribute ESO::ROdouble offsetElev;

    /**
     * Current offset right ascension (arc-sec)
     */
    readonly attribute ESO::ROdouble offsetRa;

    /**
     * Current offset declination (arc-sec)
     */
    readonly attribute ESO::ROdouble offsetDec;

    /**
     * Remaining track time for source (hour).
     * The source will be at the elevation lower limit after this period
     * has elapsed.
     */
    readonly attribute ESO::ROdouble remainingTime;

    /**
     * UT1-UTC, range -0.9 to +0.9 (sec).
     * This value is normally is obtained from the IERS bulletin.
     */
    readonly attribute ESO::RWdouble dUT1;

    /**
     * Current sidereal time (hour).
     */
    readonly attribute ESO::ROdouble siderealTime;

    /**
     * Current UTC time (MJD).
     */
    readonly attribute ESO::ROdouble utc;

    /**
     * Polar motion X coordinate (radian).
     * The polar motion X,Y are normally obtained from the IERS bulletin.
     * There value is zero for almost all work.
     */
    readonly attribute ESO::RWdouble polarMotionX;

    /**
     * Polar motion Y coordinate (radian).
     * See the notes with polarMotionX.
     */
    readonly attribute ESO::RWdouble polarMotionY;

    /**
     * Defines values giving the maximum antenna axis travel.
     */

    /**
     * Azimuth clockwise (viewed from above) limit.
     */
    readonly attribute ESO::RWdouble azCW;

    /**
     * Azimuth counter-clockwise (viewed from above) limit.
     */
    readonly attribute ESO::RWdouble azCCW;

    /**
     * Elevation low limit.
     */
    readonly attribute ESO::RWdouble elLow;

    /**
     * Elevation high limit.
     */
    readonly attribute ESO::RWdouble elHigh;

    /**
     * Defines values giving the antenna location.
     * These must be set before the antenna can be pointed.
     */

    /**
     * Antenna site mean longitude (degree).
     */
    readonly attribute ESO::RWdouble longitude;

    /**
     * Antenna site mean geodetic latitude (degree).
     */
    readonly attribute ESO::RWdouble latitude;

    /**
     * Antenna site height above sea level (meters).
     */
    readonly attribute ESO::RWdouble height;

    /**
     * Defines values giving inputs for the refraction correction.
     * These include the observation frequency and local meteorological
     * measurements.  They must be set before the antenna can be pointed.
     */

    /**
     * Observation sky frequency (Hz).
     */
    readonly attribute ESO::RWdouble frequency;

    /**
     * Local ambient temperature (degree Celsius).
     */
    readonly attribute ESO::RWdouble temperature;

    /**
     * Local atmospheric pressure (millibar).
     */
    readonly attribute ESO::RWdouble pressure;

    /**
     * Local relative humidity (percent).
     */
    readonly attribute ESO::RWdouble humidity;

    /**
     * Temperature lapse rate in the troposphere (degree K/meter)
     */
    readonly attribute ESO::RWdouble lapseRate;

    /**
     * Defines the interface for controlling and monitoring the antenna
     * pointing model.  Application of the pointing model removes errors
     * inherent to the antenna mechanical structure, i.e. geometrical
     * misalignments, flexures, etc.
     */

    /**
     * Sets the coefficient of an individual pointing model term.
     *
     * @param name      term name
     * @param coeff     coefficient
     * @param callBack  callback when the command is accepted
     * @param CBdesc    contains timeout and callback tag
     in ESO::stringSeq name,
    */
    void setCoeff (
          in string name,
          in double coeff,
          in ESO::CBvoid callBack,
          in ESO::CBDescIn CBdesc
          );

    /**
     * Enables (or Disables) an individual pointing model term.
     * When a term is Enabled its contribution is added to the total
     * pointing model correction.  When a term is Disabled it doesn't
     * make a contribution to the pointing model correction.
     *
     * @param name      term name
     * @param flag      TRUE=> enable, FALSE=> disable
     * @param callBack  callback when the command is accepted
     * @param CBdesc    contains timeout and callback tag
     in ESO::stringSeq name,
    */
    void enableTerm (
            in string name,
            in boolean enabled,
            in ESO::CBvoid callBack,
            in ESO::CBDescIn CBdesc
            );

    /**
     * Returns flags and list of pointing model Properties.
     * This method is useful for user interfaces wanting to display the
     * list of pointing model terms.  The name, description, and value
     * (coefficient) of each term can be obtained from the Property list,
     * and the corresponding state (enabled or disabled) can be obtained
     * from the flag bit pattern.
     *
     * @param termFlags each bit indicates state of corresponding term,
     *                  1-> enabled, 0- disabled
     * @param termList  list of term Properties in same order as termFlags
     */
    void getTermList (
             out ESO::pattern termFlags,
             out ESO::PropertyDescSeq termList
             );

    /**
     * Enables/Disables the individual pointing model terms.
     *
     * EnablePM has a single bit for each term, 1-enable, 0-disable
     * <p>
     *    bit   term<br>
     *    ---   ----<br>
     *    b0    ACEC<br>
     *    b1    ACES<br>
     *    b2    AN<br>
     *    b3    AW<br>
     *    b4    CA<br>
     *    b5    ECEC<br>
     *    b6    ECES<br>
     *    b7    IA<br>
     *    b8    IE<br>
     *    b9    NPAE<br>
     *    b10   TF<br>
     *    b11   TX<br>
     readonly attribute ESO::ROpattern EnablePM;
    */
    readonly attribute ESO::ROlong EnablePM;

    /**
     * Coefficient of azimuth centering error (cosine component).
     * This the cosine component of the once-per-revolution cyclic errors
     * produced by a mis-centered azimuth setting-circle.
     */
    readonly attribute ESO::ROdouble ACEC;

    /**
     * Coefficient of azimuth centering error (sine component).
     * This the cosine component of the once-per-revolution cyclic errors
     * produced by a mis-centered azimuth setting-circle.
     */
    readonly attribute ESO::ROdouble ACES;

    /**
     * Coefficient of north-south misalignment of azimuth axis.
     * In an altazimuth mount, misalignment of the azimuth axis
     * north-south, i.e. rotation about a horizontal east-west axis equal
     * to the coefficient.
     */
    readonly attribute ESO::ROdouble AN;
    /**
     * Coefficient of east-west misalignment of azimuth axis.
     * In an altazimuth mount, misalignment of the azimuth axis east-west,
     * i.e. rotation about a hirizontal north-south axis equal to the
     * coefficient.
     */
    readonly attribute ESO::ROdouble AW;

    /**
     * Coefficient of left-right collimation error.
     * In an altazimuth mount, the collimation error is the
     * non-perpendicularity between the nominated pointing direction and
     * the elevation axis.  It produces a left-right shift on the sky that
     * is constant for all elevations.
     */
    readonly attribute ESO::ROdouble CA;

    /**
     * Coefficient of elevation centering error (cosine component).
     * This is the cosine component of the once-per-revolution cyclic
     * errors produced by a mis-centered elevation setting-circle.
     */
    readonly attribute ESO::ROdouble ECEC;

    /**
     * Coefficient of elevation centering error (sine component).
     * This is the sine component of the once-per-revolution cyclic
     * errors produced by a mis-centered elevation setting-circle.
     */
    readonly attribute ESO::ROdouble ECES;

    /**
     * Coefficient of index error in azimuth.
     * This is the azimuth index error in an altazimuth mount, i.e. the
     * zero-point error in azimuth.
     */
    readonly attribute ESO::ROdouble IA;

    /**
     * Coefficient of index error in elevation.
     * This is the elevation index error in an altazimuth mount, i.e. the
     * zero-point error in elevation.
     */
    readonly attribute ESO::ROdouble IE;

    /**
     * Coefficient of non-perpendicularity of azimuth/elevation error.
     * In an altazimuth mount, if the azimuth axis and elevation axis are
     * not exactly at right angles, horizontal shifts of the image occur
     * that are proportional to sine of the elevation.
     */
    readonly attribute ESO::ROdouble NPAE;

    /**
     * Coefficient of tube flexure error (sin component).
     * This is classical tube flexure, i.e. the change in zenith distance
     * proportional to the sine of the zenith distance.
     */
    readonly attribute ESO::ROdouble TF;

    /**
     * Coefficient of tube flexure error (tangent component).
     * This is empirical tube flexure, i.e. the change in zenith distance
     * proportional to the tangent of the zenith distance.
     */
    readonly attribute ESO::ROdouble TX;
  };
};

#endif /* _AMS_IDL_ */
```