LBT-ADOPT
TECHNICAL REPORT

Doc.No : 481f301c
Version : 1.2
Date : 28 Mar 2011

AOS
The Complete Guide

Luca Fini, Alfio Puglisi, Lorenzo Busoni

CAN: 481f301c

INAF - Osservatorio di Arcetri
http://adopt.arcetri.astro.it/

ABSTRACT

This report is a complete guide to the AOS both from the functional point of view and for many implementation aspects. It is intended to be useful in order to understand AOS functionalities, so that instrument software programmers can better exploit AOS capabilities for their purposes, system programmers can be helped in troubleshooting or maintenance activities and telescope operators can better know how to use the AdOpt subsystem.
This report also includes all the information that was previously covered in document CAN 481f300 [1]

Revision history

Version	Date	CAN Number	Description
1.0	June 2009	481s301a	First draft
1.1	December 2010	481s301b	Pre commissioning version. Includes modifications and upgrades resulting from the tests in the Solar Tower Lab.
1.2	March 2011	481s301c	End of commissioning version. Includes modifications and upgrades resulting from the commissioning activity at the telescope.

Contents

Glossary of terms and acronyms

AdSec. Short for Adaptive Secondary Mirror. In this context usually refers to the group of *AO Supervisor* components controlling the hardware devices related to the secondary mirror.

ADAM. Ethernet controlled digital output. Used to enable various devices within the AdSec.

AdSec Server. The server running the *AO Supervisor* components which control the Adaptive Secondary hardware.

AdSec Arbitrator. The Adaptive Secondary Arbitrator. A component of the *AO Supervisor* which executes commands related to the *AdSec* coordinating the operations of the hardware devices in the Adaptive Secondary. Commands to *AdSec-Arb* may come either from a specific GUI or from *AO-Arb*.

AdSecArb. Short for AdSec Arbitrator.

AO System. The hardware and software components of the LBT first light Adaptive Optics System. Includes the Wavefront Sensor, the Adaptive Secondary Mirror, the *AO Servers* and some auxiliary devices (such as networking hardware) and includes the *AO Supervisor* and the real-time software.

AO Arbitrator. A component of the *AO Supervisor* which manages the execution of high-level commands, coordinating the operations of *WFS-Arb* and the *AdSec-Arb*. Commands to *AO-Arb* may come either from a specific interface or from *AOS*.

AO Servers. The servers running the *AO Supervisor* related processes.

AO Console. The operator console of either the *AdSec Server* or the *WFS Server*.

AO Supervisor. The software system which manages all the components of the *AO System*

AOArb. Short for AO Arbitrator.

AOSup Short for AO Supervisor.

BCU. Basic Control Unit. Electronics board used as basic building block for most of the electronics in the AO System (see [2]).

BCU 47. The BCU used as frame grabber for the CCD 47.

C-BCU. Crate BCU. The six BCUs which controls the AdSec.

CCD 39. The CCD used for the Wavefront sensor (also: *WFS CCD*).

CCD 47. The CCD used for the *TV*.

Copley. Motor driver for the Bayside stages.

Fastlink. The real-time data communication link between the WFS and the AdSec.

FLAO. First Light AO system for the LBT (or, if you like it better: FLorence AO System for LBT). The whole AO system for the LBT, including both hardware and software components.

Flowerpot. Auxiliary unit to control the calibration source and the related optics (cube beam splitter).

Hexapod. Mechanical support of the secondary mirror which allows to control the global mirror position with six degrees of freedom.

IIF. Instrument Interface, the TCS subsystem which provides a standard interface for instrument software.

MsgD. Message Dispatcher, the *AO Supervisor* message dispatching daemon.

OSS. Optical subsystem, the TCS subsystem which manages many devices in the telescope optical path. Most notably, from the point of view of the AO System, operates the hexapod.

RTDB. AO Real Time Database, the *AO Supervisor* own variable repository. Its functionalities are supported by `MsgD`.

S-BCU. Switch BCU. The BCU operating as input source switch for the AdSec.

TO. The Telescope Operator.

TTM. Tip-Tilt Mirror. A small mirror used to modulate the pupil image un top of the WFS pyramid.

TV. Technical Viewer. An auxiliary CCD camera used by the Wavefront Sensor to acquire the reference star.

WFS. The Wavefront Sensor. In this context usually refers to the software subsystem controlling the hardware devices related to the wavefront sensor.

WFS CCD. The CCD used in the *WFS* to measure the light wavefront deformation (also: *CCD 39*).

WFS Server. The server running the *AO Supervisor* components which control the Wavefront Sensor hardware.

WFS Arbitrator A component of the *AO Supervisor* which executes commands related to the *WFS* coordinating the operation of the hardware devices of the WFS. Commands to the WFS Arbitrator may come either from a specific GUI or from the *AOArb*.

WfsArb. Short for WFS Arbitrator.

Foreword

This document is divided into two parts. Part I describes AOS architecture and code structure, and includes details about the interaction between AOS and TCS, on one side and the AO Supervisor, on the other. Part II, describes the operational functions implemented by AOS to support the Adaptive Optics System. This second part is directly derived from a previous document (CAN 481f300 [1]) which is now obsolete.

This description of the AOS is related to AOS Version 6.x, corresponding to the version released at the end of the commissioning of the FLAO system.

Part I

AOS/AO Supervisor Software Architecture

1 Introduction

1.1 The AOS

The Adaptive Optics Subsystem (AOS) is a standard software component of the LBT Telescope Control System (TCS) whose purpose is to interface the TCS to the AO Supervisor[1]. It provides all the functionalities needed for the interaction between the LBT Adaptive Optics system and the rest of the telescope, including instruments.

The AOS is essentially an interface layer between the AOSup and the TCS as a whole. It defines a set of commands which can be used by other TCS subsystems[2] to operate the AO System during an observation and supports the continuous mirroring of a small set of AOS related variables from the TCS Data Dictionary to the AOSup RTDB, and vice-versa. Although the main flux of control is originated from the TCS and the AOSup operates mainly as a slave, AOS also allows the AOSup to issue a few commands[3] to the TCS.

1.2 The AO Supervisor

1.2.1 AO Supervisor Architecture

A detailed description of the software architecture of the AO Supervisor is provided elsewhere [3, 4], anyway, a brief general description is included here in order to provide information required to understand the operation of AOS.

Figure 1 shows the overall architecture of the AOSup, and its relationships with the TCS.

The AOSup is based on a distributed architecture where several independent processes cooperate in order to properly manage the underlying hardware, while ensuring safe operations of any device. The organization of processes in the AOSup reflects the splitting of FLAO system in two subsystems: the Adaptive Secondary (AdSec) and the Wavefront Sensor (WFS). The two software subsystems are currently hosted by two independent servers[4] indicated in the figure by the two large boxes named, respectively, `adsecdx` and `wfsdx`[5]

[1] Please note that in the following pages we will use the term **AOS** to indicate the Adaptive Optics Subsystem, i.e.: the software component of the TCS which communicates which the AO Supervisor, while we will use the term **AO System** to indicate the Adaptive Optics System as a whole. The term **AO Supervisor** or AOSup, indicates the software controlling the AO System.

[2] The TCS subsystems which use the AOS commands are currently the IIF and the AOS GUI.

[3] This feature is used essentially to support the "Offload modes" function and can be used to allow the control of the AdSec hexapod from AO Supervisor procedures. See sections 8 and 9.1.

[4] This hardware architecture was selected in order both to provide enough computing resources and to have a clear separation between the two components of the AO System, but it is not a requirement: the software design, in fact, allows different organization of processes, e.g.: the whole AOSup could run on a single server.

[5] The names refer to the right side AO System. For the left side the AO Servers are named `adsecsx` and `wfssx`.

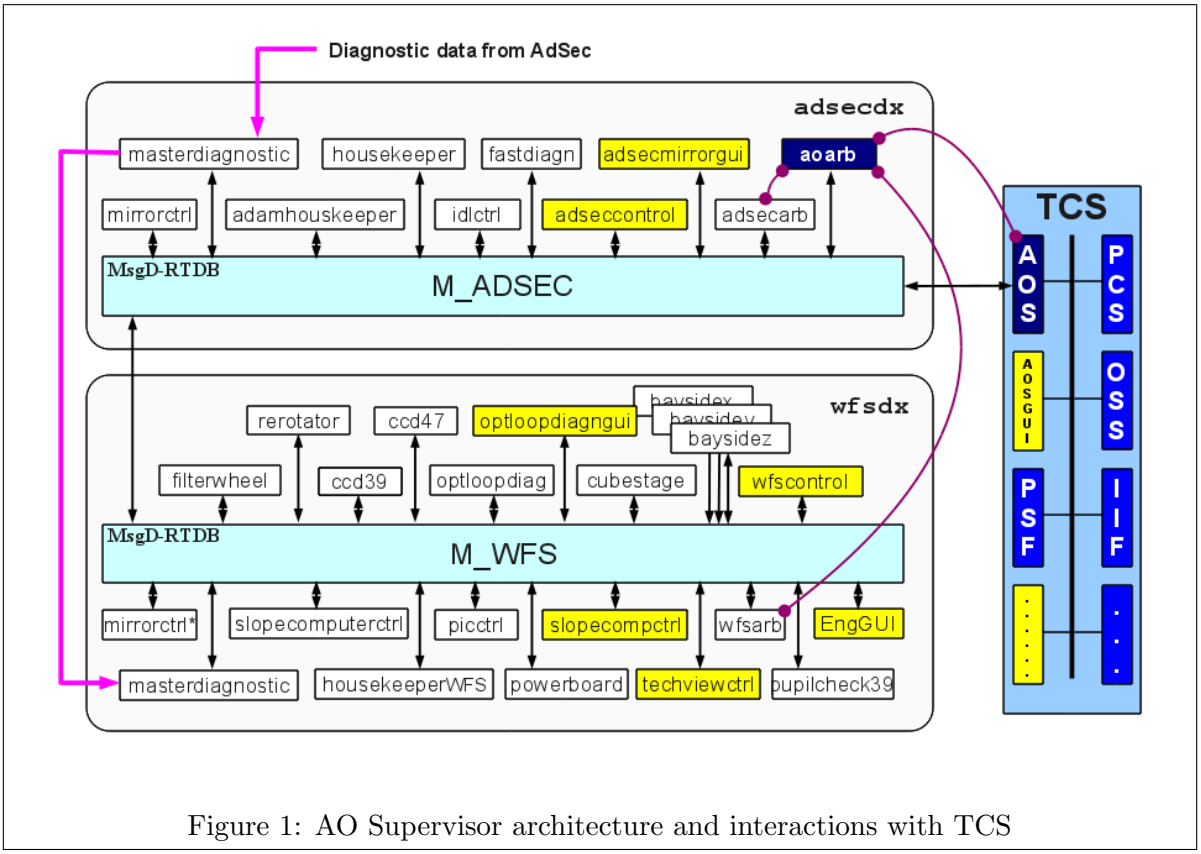

Figure 1: AO Supervisor architecture and interactions with TCS

All interactions between AOSup processes are supported by a central process named MsgD-RTDB[6] which allows the exchange of messages between any processes, supports a centralized repository of variables, and acts as a centralized logging facility. As shown in the picture several MsgD-RTDB processes can communicate among them to allow interaction between processes belonging to different subsystems. In our case the structure includes two interconnected MsgD-RTDB processes (named M_ADSEC and M_WFS, respectively), one for each subsystem[7].

From the AOSup side, most of the interactions with TCS are managed by a dedicated process the "AO Arbitrator" (`aoarb`), whose main function is to receive commands from the AOS and coordinate the execution of requested operations by sending subcommands to the arbitrators controlling the WFS and the AdSec: `wfsarb` and `adsecarb`, respectively. The AO Arbitrator includes a finite state machine which defines the legitimate sequences of commands which can be issued. The main sequence of control is represented in figure 1 by the violet lines.

1.2.2 AO System operating modes

When supporting an observation, the AO System has four different operating modes:

[6] Message Dispatcher and Real-Time Database. See a more detailed description in [].

[7] The architecture shown in figure 1 is to be extended to support the operations of LBTI by adding an identical WFS section for the LBTI Wavefront Sensor.

- **FIX-AO**, when operating in "seeing limited" mode. In other words, this is not an "adaptive" mode in that the shape of the adaptive mirror is fixed, but is anyway supported by the AO System. When in FIX-AO mode the AOSup can operate without the support of the WFS subsystem.
- **TTM-AO**, when operating in adaptive mode, but correcting only the tip-tilt components of the atmospheric aberration.
- **ACE-AO**, when operating in full adaptive mode with an automatic selection of AO loop parameters.
- **ICE-AO**, when operating in full adaptive mode and providing means to allow the observer to adjust AO loop parameters[8].

The AOSup can be operated according to the above operating modes by means of a set of commands issued by the AOS and received and executed by the AO Arbitrator.

1.2.3 AO Arbitrator Commands

The set of commands which AOS can use to control the AO System is shown in table 1. By using these commands properly (i.e.: in the proper sequence) the AOS can control all the operations that are needed to successfully perform an observation.

Note that AOS defines its own set of commands which are provided to other TCS subsystems to operate the AO System, but the AOS command set is not exactly matching the AO Arbitrator set: in order to adapt the internal logic of the AO System, to the operational scheme of TCS some AOS commands are implemented as short procedures mixing together commands issued to the AO Arbitrator and internal TCS commands. More details related to AOS commands and their implementation can be found in section 7 and extensively in the second part of this report.

1.2.4 AO Arbitrator Command Sequences

In order to properly operate the AO System, a specific sequence of commands must be issued from the AOS to the AOSup. The AO Arbitrator is driven by a Finite State Machine (whose diagram is shown in figure 2) which ensure that commands are provided in legal sequences.

The main sequence of commands to perform an adaptive mode observation, starting from *Operational* status, consists of the following steps:

1. `PresetAO`: provides the AOSup with values for all parameters which are needed to prepare for AO; moved to *ReadyToAcquire* status.
2. `AcquireRefAO`: starts the acquisition of the reference star. When finished moves to *RefAcquired* status.
3. `StartAO`: closes the AO loop.

[8] ICE-AO operating mode has not yet been implemented.

Table 1: Commands accepted by the AO Arbitrator

Command	Description
`PresetFlat`	Preset AO System for seeing limited operation. The AO System at startup has already a default seeing limited shape. This command may be used to select different shapes, e.g.: optimized for specific instruments and/or operations.
`PresetAO`	Preset AO System for adaptive operation
`AcquireRefAO`	Acquire the reference star and become ready for closing the AO loop
`CheckRefAO`	Returns offset values between reference star actual and nominal position
`RefineAO`	Perform optimization of AO loop parameters[a]
`ModifyAO`	Modify some AO loop parameter[a]
`StartAO`	Start the AO mode (i.e.: close the AO loop)
`OffsetXY`	Offset AO pointing[b]
`OffsetZ`	Offset AO focus[b]
`CorrectModes`	Apply mirror shape correction while in closed loop
`PauseAO`	Temporarily suspend the adaptive correction loop
`ResumeAO`	Resume a suspended adaptive correction loop
`Stop`	Stop current operation
`SetZernikes`	Apply mirror shape correction while in seeing limited mode

[a] This command has not yet been implemented.

[b] When this command is issued in closed loop, the offset value is limited to about half an arcsec, due to the limited speed of the offload modes mechanism.

As shown in figure 2 in each operational status the AO Arbitrator may accept some commands other than the one defined in the main sequence. As an example the `OffsetXY` command is legal both in *LoopClosed* and in *LoopSuspended* status. This is needed to support target offsetting of any length: when the desired offset is less than one half of an arcsec, the offset command can be issued maintaining the loop closed. For larger offsets, instead, the following sequence of commands must be issued:

1. `Pause`
2. `OffsetXY`, while an opposite offset is requested to the telescope pointing system
3. `Resume`

The task of implementing this sequence is to the subsystem controlling the AOS (typically the IIF).

1.2.5 Safety issues

During the operation of the Adaptive Mirror a number of safety issues must be taken into account, among those a few require a special support from the AOS:

1. **Telescope elevation**: the mirror must be set in safe position if the telescope moves below 20 degrees of elevation.

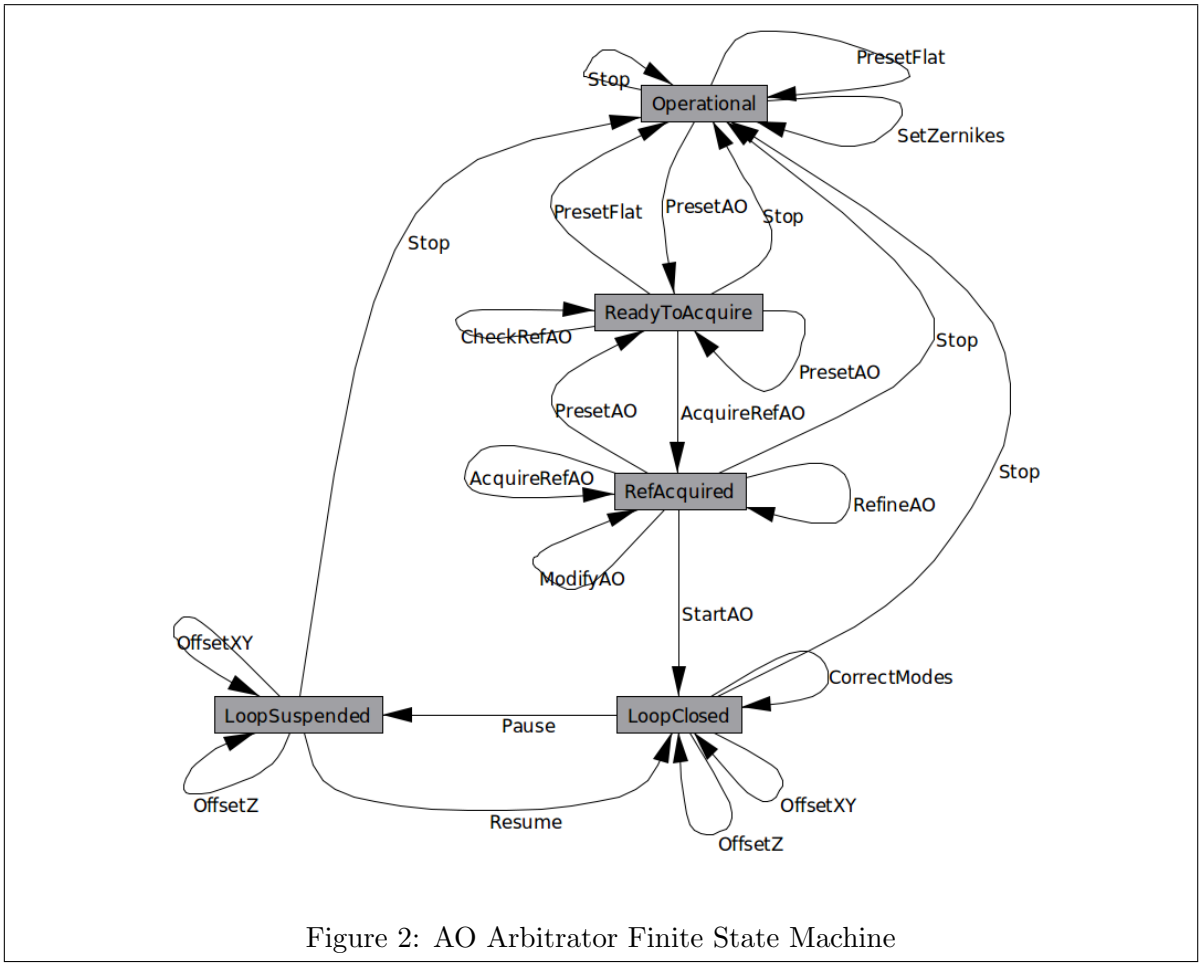

Figure 2: AO Arbitrator Finite State Machine

2. **Swing arm position**: the mirror must not be operated when the swing arm holding it is not in the optical path position.

3. **Wind**: a special status (TSS) must be enabled if the wind speed in the vicinity of the thin shell raises over 8 m/s.

The above conditions must be securely detected to take the proper actions.

AOS provides support for the checks by maintaining proper variables in the RTDB and implementing an expiration mechanism for those variables (see section 4). This implies that the AOSup cannot be operated if the AOS (and at least a subset of other TCS subsystems) is not up and running[9].

2 AOS Architecture

The AOS is a standard subsystem of TCS. As such it is globally structured like any other TCS subsystems. As detailed in the following sections, AOS is divided essentially into three modules:

[9]To allow to operate the AO System when AOS is not operating, e.g.: for lab tests and "close dome" maintenance activity, the AOSup can be set in *lab mode*, where the checks are disabled. This feature is not available through the AOS or any other procedure available to telescope operators.

Main, AOSSubsystem, and AOSClient.

2.1 Main

The Main module[10] performs the following steps:

1. Gets configuration data; e.g.: the IP number of the machine where the AOSup MsgD is running, the initial debug level, etc. (for a list of all configuration parameters see section 10).
2. Initializes the telemetry collection system.
3. Starts the AOSSubsystem's thread.
4. Initializes the AOS commands.
5. Waits for AOSSubsystem to terminate.

2.2 AOSSubsystem

AOSSubsystem[11] is structured as any other TCS subsystem and like other subsystems is executed in its own thread and is provided with an execute() method which is where part of the job is done.

In AOS, execute() is essentially a loop which controls the communication rendez-vous with the MsgD.

The loop performs five simple steps:

1. Checks for a terminate request
2. Instantiates an AOSAoApp object.
3. Starts AOSAoApp by executing its Exec() method.
4. Destroys the AOSAoApp object.
5. Sleep 1 second

The AOSAoApp object is derived from the AOApp class provided by the AOSup software. When started it tries to connect as a client to MsgD waiting indefinitely for a reply. When the communication is established, the AOSAoApp finishes up its initialization and begins its operations by calling the Exec() method.

The Exec() method returns only when the communication with MsgD terminates (e.g.: because the AOSup is stopped). In this case the loop described above is iterated and AOS returns to wait for communication with MsgD.

Because most of the functionality of the AOS is embedded into the class AOSAoApp, the latter is described in deeper details in section 3.

[10] Main module code is in files Main.cpp, Main.hpp.
[11] AOSSubsystem code is in files: AOS.cpp, AOS.hpp.

2.3 AOSClient

`AOSClient`[12] implements the class to be used by clients (on the TCS side) to send commands to the AOS. This follows closely the TCS standard for client communication and its purpose is simply to receive from other TCS subsystems commands to be translated into commands to be delivered to the AOSup (see sect. 7).

3 AOSAoApp

`AOSAoApp`[13] is the class at the heart of the AOS; it is derived from the standard `AOApp` class used to implement clients for the AOSup and provides all the functionalities needed for the communication between AOSup components. It implements a number of base functionalities such as message queue management, automatic replies to several housekeeping messages, error logging, and more.

Any application based on `AOApp` is intrinsically multi-threaded, although this is hidden into the implementation and multi-threading is not directly managed by the application programmer.

After being instantiated `AOSAoApp` executes its `Exec()` method which performs a standard initialization procedure as follows:

1. Installs handlers for asynchronous messages coming from the AOSup.
2. Waits for `MsgD` to accept a connection.
3. Executes the `Run()` method.

The Run() method is where the functionalities of `AOSAoApp` are implemented and is essentially a loop which terminates when the communication with `MsgD` is interrupted for any reason or when receiving from some Supervisor client an explicit termination command[14]. When the `Run()` method terminates, the `Exec()` method as a consequence is also terminated and the control returns to `AOSSubsystem`, as described in section 2.2.

While connected as an `MsgD` client the AOS registers itself either with the name `AOS.L` (left side) or `AOS.R` (right side).

3.1 Handlers

Handlers are special methods[15] of `AOSAoApp` which manage asynchronous messages coming from the AOSup. They manage messages related to the following functions:

- `arbalert_hndl`: Receives “alert” notifications from the AO Arbitrator

[12] The source code is in files: `client/AOSClient.cpp` and `client/AOSClient.hpp`.
[13] The related source code is in files: `AOSAoApp.cpp`, `AOSAoApp.hpp`.
[14] This is a standard feature of all Supervisor clients and is used to properly shutdown all the client processes when the AOSup is terminated
[15] The related source code is in files: `AOSHandlers.cpp` and `AOSHandlers.hpp`.

- `varnotify_hndl`: Receives notifications of relevant RTDB variables change.
- `hexapod_hndl`: Receives and executes hexapod related commands.
- `housekeep_hndl`: Receives and executes housekeeping commands used for debugging purposes.
- `offload_hndl`: Receives and reissues "offload" requests (see section 8).
- `default_hndl`: Receives all other messages directed towards the AOS.

Handlers are installed as part of `AOSAoApp` initialization procedure. Each handler runs in its own thread and receives a predefined subset of the messages sent by `MsgD` to be processed.

Here follows a brief description of each handler.

3.1.1 `arbalert_hndl`

Alerts are messages sent by arbitrators to notify error conditions which are not the consequence of some error in command execution, but are dependent on external conditions such as hardware failures and the like.

The purpose of the corresponding handler is currently to decode the error message and properly display it on the operator GUI and log it in the TCS log system. If the need arises alert messages could also be used to perform some specific operation, e.g.: to force a stop of the current operation.

3.1.2 `varnotify_hndl`

The variable notification feature of the `RTDB` is used to mirror relevant variables from the `RTDB` into the TCS `DD`. As part of its initialization the `AOSAoApp` registers with the `MsgD` to be notified of updates of a specified set of variables.

Whenever any AOSup client writes to one of the variables this handler receives a notification (which includes the variable value). Upon notification, the handler writes the variable value into the corresponding variable in TCS `DD` and, possibly, performs some variable specific action.

A list of notified variables can be found in section 4.

3.1.3 `hexapod_hndl`

This handler manages hexapod related commands[16] sent from the AOSup.

When an hexapod command is received, the handler checks whether the command can be accepted and, if positive, converts it into a command to OSS to be applied to the hexapod. An acknowledge message is sent back to the sender of the command.

Details on hexapod commands are covered in section 7.

[16] In AOS design the capability to directly control the position of the adaptive mirror hexapod was included to support some specific operations needed for engineering or calibration procedures.

3.1.4 housekeep_hndl

This handler manages a set of housekeeping commands, mostly used for debugging and troubleshooting operations. For a list of defined housekeeping commands see section 7.

3.1.5 offload_hndl

This handler receives messages containing requests to offload errors accumulated in the adaptive mirror onto some other telescope device (e.g.: the pointing system, the hexapod or the primary mirror). A detailed description of the offload mechanism can be found in section 8.

3.1.6 default_hndl

The default handler will receive all messages not trapped by other handlers. It is used to manage the TERMINATE command[17].

All other commands are unexpected and thus will only generate warnings.

3.2 AOSAoApp::Run()

After a successful instantiation and AOSAoApp and its registration as a client of MsgD, the Run() method is executed.

The Run() method is the main loop of the AOS, despite the fact that most of the work is performed elsewhere, i.e.: by handlers for commands originating from the AOSup, and by the TCS Command Interface code for commands originating from other TCS subsystems.

When started the Run() method performs the last steps of AOSAoApp initialization and enters the main loop where, at a convenient rate[18], the following tasks are performed:

1. Checks the current status of the AOSup and sets the AOS status accordingly. This is done by the synchronize() method.
2. Performs the mirroring of variables from the TCS DD to the RTDB by polling the variables from the DD and writing their values into the RTDB. This is done by the updateTCSVariables() method.

In order to derive the current status of communication, the synchronize() method makes a number of tests: communication with the MsgD, presence of the AOARB arbitrator, value of the "Connection status" variable. Possible results of the checks are the following:

- NO CONNECTION: no communication with MsgD has been established.

[17] All AO Supervisor clients are required to properly manage the TERMINATE command. It is not intended for normal use but only in debugging or troubleshooting operations.
[18] Currently the main loop rate is about 1s.

- NO ARBITRATOR: communication with `MsgD` has been established, but the AO Arbitrator is not running.
- STANDALONE: both `MsgD` and the AO Arbitrator are up and running, but the AO Subsystems will not accept commands from AOS.
- OPERATING: the AOSup is up and running and willing to receive commands.
- SIMULATION: the AOS was started in simulation mode, no connection to `MsgD` is attempted.

At the same time the responsiveness of the WFS Arbitrator is checked to allow proper status display on the AOS GUI.

As a by-product of the above checks, if the `MsgD` does not respond, the communication cycle of the `AOSAoApp` is terminated as detailed in section 3.

4 Variables

AOS, alike all other TCS subsystems, uses a dedicated section in the TCS Data Dictionary to hold variables for various purposes. In the following tables the most relevant[19] variables are listed and described.

Table 2: AOS housekeeping variables

aos.side[s].connected	bit	Indicates when the AOS is connected to `MsgD`
aos.side[s].conntime	string	Time of last connection to `MsgD`
aos.side[s].msgdident	string	`MsgD` identification string
aos.side[s].msgdip	string	IP address of machine running the `MsgD`
aos.side[s].running	bool	Indicates when the AOS is connected to `MsgD` and communicating with the AOArb
aos.side[s].servstat	string	Current AOSup communication status
aos.side[s].starttime	string	Time of AOS start
aos.side[s].tel_enabled	bit	Indicates when the telemetry collection is enabled

AOS variables can be divided into four groups:

- **Housekeeping variables**, used for various housekeeping and information tasks, listed in table 2.
- **TCS variables**, i.e.: variables defined into TCS `DD` which are mirrored into AOSup `RTDB`. They are listed in tables 3, 4, 5. For these variables, values are updated by periodically polling the TCS Data Dictionary; the polling mechanism is described in section 3.2.

  Note 1: most variables have direct correspondence between the `DD` and the `RTDB`. A few are functions of more than one `DD` variable (e.g.: `ISTRACKING` is a boolean value derived from `mcs.azDrive.trackingMode`, `mcs.elDrive.trackingMode` and `mcs.onSource`).

[19] A few variables, used exclusively for internal uses of the AOS code, are omitted.

Table 3: TCS variables mirrored to `MsgD`: Environment

AOS.AMB.WINDSPEED `env.anemometer.r_average_10s`	real(**T**)	Wind speed as measured from the AdSec anemometer
AOS.DIMM.SEEING `iif.DIMM.seeing`	real	Seeing value from the DIMM
AOS.EXTERN.WINDDIRECTION `env.weather.lbt.windDirection`	real	External wind direction
AOS.EXTERN.WINDSPEED `env.weather.lbt.windSpeed`	real	Wind speed as measured from the AdSec anemometer

Table 4: TCS variables mirrored to `MsgD`: Positions

AOS.TEL.AZ `mcs.azDrive.position`	real(**T**)	Current telescope azimuth
AOS.TEL.DEC `pcs.pointingStatus.achieved.achieved_DEC.Radians`	real	Current telescope declination
AOS.TEL.EL `mcs.elDrive.position`	real(**T**)	Current telescope elevation
AOS.TEL.RA `pcs.pointingStatus.achieved.achieved_RA.Radians`	real	Current telescope right ascension
AOS.x.GUIDECAM.CENTROID.X `gcs.side[s].GuideCam.centroid_FWHM_X`	real	Guide camera centroid X
AOS.x.GUIDECAM.CENTROID.Y `gcs.side[s].GuideCam.centroid_FWHM_Y`	real	Guide camera centroid Y
AOS.x.HEXAPOD.ABS_POS `oss.side[0].adsc.abs_pos`	real[6]	Hexapod absolute position
AOS.x.HEXAPOD.STATUS `hexapod_status()`	int funct	Hexapod status.
AOS.x.ROTATOR.ANGLE `mcs.rotatorSide[s].rotators[2].actualPositionASec`	real	Rotator angle
AOS.x.TERTIARY.ABS_POS `oss.side[s].terc.abs_pos`	real[4]	Hexapod absolute position

Note 2: for a small subset of the variables it is important to asses that it is actually a "live" variable, i.e.: it is constantly updated by the owner process. Such variables (indicated with a "T"), have an associated timestamp variable; as party of the polling process, the timestamp variable is checked and if it becomes "older" than a preset expiration time the corresponding variable it is set to an undefined value. This allows the AOSup to take actions based on variable expiration[20].

- **RTDB variables**, variables defined into the AOSup `RTDB` which are mirrored into TCS `DD`. These variables are divided into subgroups and are listed in the following tables: table 6, for

[20] This mechanism is used, e.g.: for safety related variables as described in section 1.2.5.

Table 5: TCS variables mirrored to `MsgD`: Miscellaneous status items

AOS.TEL.HBS_ON `hbs_on()`	int funct	Set to 1 when HBS is on
AOS.TEL.ISGUIDING `tel_guiding()`	int funct	Set to 1 when telescope is guiding
AOS.TEL.ISTRACKING `tel_tracking()`	int funct	Set to 1 when telescope is tracking
AOS.x.SWA.DEPLOYED `swa_deployed()`	int funct	(**T**) Set to 1 when the swing arm is deployed
AOS.x.TEL.VENT_ON `vent_on()`	int funct	(**T**) Set to 1 when the ventilation system is running

Table 6: AO System variables used by AOS

AOARB.x.CORRECTEDMODES `aos.side[s].ao.correctedmodes`	int	Number of corrected modes
AOARB.x.FSM_STATE `aos.side[s].ao.status`	string	Adaptive optics system status
AOARB.x.IDL_STAT `aos.side[s].idlstat`	string	When 0 indicates a failure in the IDL subsystem
AOARB.x.LAB_MODE `aos.side[s].labmode`	long	When 1 indicates that the AOSup is operating in "Lab Mode" (see section 1.2.5)
AOARB.x.LOOPON `aos.side[s].ao.loopon`	long	Set to 1 when the adaptive loop is closed
AOARB.x.MODE `aos.side[s].ao.mode`	string	Current observation mode (`FIX_AO`, `TT_AO`, `ACE_AO`, `ICE_AO`.)
AOARB.x.MSG `aos.side[s].ao.msg`	string	Generic message string. Currently not used.
AOARB.x.OFL_ENABLED `aos.side[s].ao.ofl_enabled`	long	Set to 1 when the offload mechanism is enabled (see sect. 8
AOARB.x.STREHL `aos.side[s].ao.strehl`	real	Estimated Sthrel ratio
AOARB.x.WFS_SOURCE `aos.side[s].ao.wfs_source`	real	Wavefront sensor currently in use. Default is "FLAO", the WFS for first light AO. The other currently defined value is "LBTI"

variable related to the AO system as a whole, table 7, for variables related to the AdSec, table 8, for variables related to the WFS.

Values for these variables are updated as needed by means of the notification mechanism provided by `MsgD`.

Table 7: Adaptive Secondary variables used by AOS

AOARB.x.ADSEC.COIL_STATUS `aos.side[s].adsec.coil_status`	long	Set to 1 when adsec coils are enabled
AOARB.x.ADSEC.FSM_STATE `aos.side[s].adsec.status`	string	Status of the AdSec FSM
AOARB.x.ADSEC.HEALTH `aos.side[s].adsec.health`	long	Indicates that all processes relevant for the AdSec are up and running
AOARB.x.ADSEC.LED `aos.side[s].adsec.led`	long	Indicates some status of the thin shell (0=off/1=safe/2=set)
AOARB.x.ADSEC.MSG `aos.side[s].adsec.msg`	string	Generic message related to AdSec. Currently unused
AOARB.x.ADSEC.PWR_STATUS `aos.side[s].adsec.main_pwr_status`	long	Set to 1 when adsec main power is on
AOARB.x.ADSEC.SHAPE `aos.side[s].adsec.shape`	string	Currently selected mirror shape
AOARB.x.ADSEC.TSS_STATUS `aos.side[s].adsec.tss_status`	long	Set to 1 when mirror TSS system is enabled

Notes on variable tables:

1. In the tables variable types are indicated as string, int, real. They are currently implemented in the AOS with corresponding types defined for the `DD`: string, long, double
2. In variable names related to TCS `DD` the character "`s`" is used to indicate the telescope side (current corresponding values are 0 for left side and 1 for right side).
3. In variable names related to AOSup `RTDB` the character "`x`" also indicates the side, where needed. Its value may be either `L` or `R` for left and right side, respectively.
4. The source code relevant for the management of variables can be found in files: `VarUtils.cpp` and `VarUtils.hpp`.

5 Events and logging

`AOSAoApp` is provided with 5 levels of log messages[21], in decreasing order of verbosity: `trace`, `debug`, `info`, `warning` and `error`.

The first two levels (most verbose) correspond only to lines logged via the *Syslog* mechanism and can be disabled either by a configuration parameter (see section 10) or via run-time messages from the AOSup.

[21] The related source code is in files: `AOSEvent.hpp`, `AOSEvent.cpp`.

Table 8: Wavefront Sensor variables used by AOS

wfsarb.x.CCDBIN@M_WFS `aos.side[s].wfs1.ccdbin`	int	WFS CCD binning
wfsarb.x.CCDFREQ_LIMITS@M_WFS `aos.side[s].wfs1.ccdfreq_limits`	real[2]	WFS CCD frequency limits
wfsarb.x.CCDFREQ@M_WFS `aos.side[s].wfs1.ccdfreq`	real	WFS CCD current frequency value
wfsarb.x.COUNTS@M_WFS `aos.side[s].wfs1.counts`	int	WFS CCD average counts per frame
wfsarb.x.FILTER1@M_WFS `aos.side[s].wfs1.filter1`	string	WFS filter 1 position
wfsarb.x.FSM_STATE@M_WFS `aos.side[s].wfs1.mod_ampl`	real	Status of the WFS FSM
wfsarb.x.HEALTH@M_WFS `aos.side[s].wfs1.health`	long	Indicates that all processes relevant for the WFS are up and running
wfsarb.x.LED@M_WFS `aos.side[s].wfs1.led`	long	Indicates status of the WFS hardware (off/set)
wfsarb.x.MOD_AMPL@M_WFS `aos.side[s].wfs1.mod_ampl`	real	WFS TT-mirror modulation amplitude
wfsarb.x.MSG@M_WFS `aos.side[s].wfs1.msg`	string	WFS related generic message
wfsarb.x.NO_SUBAPS@M_WFS `aos.side[s].wfs1.no_subaps`	int	Number of subapertures in current WFS configuration
wfsarb.x.PYRAMID_POS@M_WFS `aos.side[s].wfs1.pyramid_pos`	int[2]	Position of WFS pyramid with respect to TV CCD
wfsarb.x.TV_BINNING@M_WFS `aos.side[s].wfs1.tv_binning`	int	TV CCD current binning
wfsarb.x.TV_EXPTIME@M_WFS `aos.side[s].wfs1.tv_exptime`	real	TV CCD current exposure time
wfsarb.x.TV_FILTER2@M_WFS `aos.side[s].wfs1.tv_filter2`	string	TV CCD current fiter 2 selection

The following three levels correspond both to *Syslog* messages and to TCS Events and thus are also logged to the LSS system. `Warning` and `error` events are obviously related to error conditions of increasing severity, while `info` events are generated to log command execution and offload requests.

6 TV image frames

The AOS GUI (see section 11) includes the capability to display images from the Wavefront Sensor Technical Viewer. Such images are of two types: a) frames continuously generated by the TV CCD during its operation, and b) the specific frame when the reference source has been acquired as a

consequence of an `AcquireRefAO` command. The former are transmitted asynchronously by means of the variable notification mechanism supported by the `MsgD`; the latter are sent in the set of parameters returned by the `AcquireRefAO` command. The two types of images are displayed independently.

Technical Viewer images are formatted as specified in the following structure:

```
struct TVIMAGE {
     int rows;       // Number of rows
     int cols;       // Number of cols
     unsigned char pix[];
}
```

Pixels are represented as gray levels in the range 0-255. The structure is packed into a byte stream of the proper length.

Due to implementation details of the TCS middleware, it is not practical to use either the parameter passing mechanism of TCS commands or the `DD` to transmit even moderately large data sets such as the 256x256 image from the TV.

For this reason an indirect mechanism is used to send TV frames to be displayed on the AOS GUI: frames from the TV are written to a temporary file in a NFS shared directory as soon as they are received from AOSup. AOS GUI polls for changes in file access date and displays the file content whenever the date changes.

The paths for TV image support files are specified in a configuration parameter.

7 Commands

The AOS provides two command based interfaces one is built over the AOSup command mechanism is used to receive commands from the AO Supervisor the other is based on the standard TCS command mechanism and is used to provide AO services to the TCS.

7.1 Commands from the AO Supervisor

Commands received from the AOSup are used to support the “Offload modes” function (see section 8), to allow the AO Supervisor to directly operate the hexapod during some calibrations procedures (see section 9.1) or to perform housekeeping tasks not directly connected with the operation of the AO System (see section 9.2).

7.2 Commands from other TCS subsystems

AOS defines a set of commands to be used by other TCS subsystems to operate the AO System. These commands are described in full details n the second part of this report (see sections 12 through 14).

8 Offload Modes

Mode offload is needed during Adaptive Optics closed loop when the AdSec gets near to some physical limit, in order to offload the error on the first modes accumulated in the deformable mirror to some other telescope device.

A typical example is when the telescope accumulates tracking errors: if the adaptive loop is on, those errors are compensated by the adaptive secondary by an increasing static tip-tilt component. As errors increase the AdSec would finally get close to intrinsic limits in the maximum amount of tip-til it can compensate. The error must thus be lowered, e.g.: by adjusting the telescope pointing or by moving the hexapod.

The mode offload mechanism is based on a specific command sent by AOSup and received by the related handler (see page 12). The command argument consists of an array of 22 zernike terms which is managed by a specific method of the `AOSAoApp` object[22].

Each array received is managed as follows:

1. The array is divided into two parts: the lower order part (elements 2 to 4: the first element is never considered) related to tip/tilt and focus terms, and the higher order part, including elements from 5 to 22. The two parts are independently compared against an array of threshold values.

   The `tip-tilt-focus` offload action is scheduled to be executed if the absolute value of any of the low order offload components is greater than the corresponding threshold. Analogously the `highorder` offload action is scheduled to be executed if the absolute value of any of the high order component is greater than the corresponding threshold.

2. The command queue is checked to verify that no other commands are waiting. If more commands are waiting the current one is discarded[23] with a warning event.

3. If any offload action is to be executed, the offload vector is rescaled by multiplying it by a vector of gain coefficients[24] to adjust for differences in units used for zernike values in the AOS with respect to the AO Supervisor.

4. If a `tip-tilt-focus` offload action is scheduled, tip tilt and focus values are computed from the low order zernike terms and the correction is sent as a PSF command to be applied to the Hexapod.

5. If a `highorder` offload action is scheduled, the offload vector (with the first 4 components set to 0) is sent to the PSF to be applied to the primary support system.

[22] The related code is in file `AOSAoApp.cpp` (method: `offload()`)

[23] The AOSup is configured to send offload messages at a rate not greater than one Hertz. Despite this, the actual offload execution by the TCS may be slower than that. Discarding waiting offload request except the most recent one, ensures that the TCS does not receive more offload commands than it can manage.

[24] Threshold and rescaling values are defined in a configuration file which is read at start time (see section 10).

9 Engineering and housekeeping commands received from AOSup

9.1 Engineering commands

Table 9: Hexapod commands issued by AOSup

Command	Description
HXPINIT	Initialize hexapod
HXPMOVETO	Move hexapod to absolute position
HXPMOVEBY	Move hexapod relative tom current position
HXPMOVSPH	Move hexapod on a sphere
HXPNEWREF	Set new reference point
HXPGETPOS	Get hexapod current position
HXPGETABS	Get hexapod absolute position
HXPOPENBRAKE	Open brake
HXPCLOSEBRAKE	Close brake
HXPISINITIALIZED	Test if hexapod has been initialized
HXPISMOVING	Test if hexapod is moving

This set of commands are used by the AOSup to request the TCS to execute specific actions. They are never used during an observation, but may be needed to perform engineering tasks, e.g.: for calibrations or maintenance operations.

Currently only commands to operate the hexapod are supported. They are listed in table 9.

9.2 Housekeeping commands

This set of commands are used for various housekeeping tasks such as modifying the logging level, forcing the logging of specific items into TCS logging system, enabling or disabling optional functions.

Housekeeping commands are listed in table 10.

Table 10: Housekeeping commands

Command	Description
LogItem	Requests to log some piece of information into the TCS Logging System.
LogOn	Activate mirroring to `MsgD` of AOS generated log Messages.
LogOff	Deactivate mirroring to `MsgD` of AOS generated log Messages.
DbgLevel	Set debug level of AOS

9.2.1 LogItem

Requests to log some piece of information into the TCS Event System.

AOSup will have it's own logging system to be used for troubleshooting and engineering tasks. A selected subset of the log data will be stored in the TCS log system by means of `LogItem` commands.

9.2.2 LogOn/LogOff

Activate/deactivate the mirroring of AOS generated log messages to the `MsgD`.

After activation all the log messages[25] generated by the AOS other than being logged via the standard TCS logging mechanism, will also be echoed to the `MsgD` in order to be merged with AOSup specific logs. This will help in the maintenance of the system by allowing to easily correlate the relevant log messages.

9.2.3 DbgLevel

Activate/deactivate the AOS log messages. This command may be used ad debugging tool to increase/decrease the verbosity of messages generated by AOS. Under normal conditions debug logging should be off (i.e.: `DbgLevel=0`).

Note that if a LogOn command has also been sent, all logs will be mirrored in the `MsgD` log file.

10 Configuration items

Table 11: AOS Configuration Parameters

Item	Type	Description
AOSLMsgDIp AOSRMsgDIp	string	Message daemon IP number. The IP address of the workstation running `MsgD` in the numerical dotted form.
AOSLtv_file AOSRtv_file	string	Basename of temporary file for the TV snapshot image. The procedure will create and update files named `<name>.dat` and `<name>.tmp`
AOSLfl_file AOSRfl_file	string	Basename of temporary file for the specification of the list of "flat" shapes available[26].
AOSLdbgLevel AOSRdbgLevel	int	AOS debug level. For debugging purposes AOS is provided of an internal log mechanism writing onto syslog. In normal use a debug level of 0 must be used; values 1 and 2 will provide increasingly verbose log.
AOSLsimulation AOSRsimulation	bool	Enable simulation mode when true. Simulation mode is used to perform a limited number of tests on AOS without the need to activate the AOSup.
AOSLlogdir AOSRlogdir	bool	AOS own log directory (currently unused)
AOSLtelEnable AOSRtelEnable	bool	Enable telemetry collection

AOS has a dedicated section in the general configuration file (`lbt.conf`). A list of AOS related parameters is shown in table 11. The naming follows a simple convention: they are all prefixed by either `AOSL` or `AOSR`, respectively, for left and right side.

[25] Except for messages explicitly logged to TCS Syslog by the `LogItem` command.

AOS also uses data stored in two configuration files (for each side). They are stored in the AOS specific configuration area[27] and contain data (thresholds, and conversion coefficients) related to the "Offload Modes" mechanism (see section 8) and for the `SetZernikes` command (see section 14.14).

Table 12: `DXOffloadModes.dat` and `SXOffloadModes.dat`

```
# Gains and thresholds for Zernike values sent for Offlaod Modes commands
#
# 1. Thresholds.
# If no value exceeds the threshold, no correction is applied. If some do
# exceed the threshold, the correction is applied. Thresholds are applied
# before multiplying by the gains. As a consequence, thresholds values
# are in AO Supervisor units (meters).
#
#  z1    z2    z3    z4   z5   z6   z7   z8   z9  z10  z11  z12  z13  z14  z15  z16  z17  z18  z19  z20  z21  z22
  1.0  1.e-7 1.e-7 1.e-7 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1 5e-1

#
# 2. Gains.
# Zernike components are managed differently. Z1 is ignored,  Z2,Z3 must
# result in tip/tilt values in arcsec. Z4 must result in the focus value, in
# mm. All other values must result in standard zernike values in nanometers.
#
#  z1     z2       z3       z4     z5    z6   z7   z8   z9   z10  z11  z12  z13  z14  z15  z16  z17  z18  z19  z20  z21  z22
  0.0     -0.3e6   0.3e6    2e4    0     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

Examples of the content of the mentioned files can be found in tables 12 and 13.

11 AOS GUI

Although AOS is essentially a thin interface layer to allow TCS subsystems to operate the AO system, and thus it should be totally controlled by the instrument software through the IIF, it is anyway provided with a GUI which is intended to be up and running at the operator console during observations. Its design is based on a set of requirements provided by potential users [5]. A previous document describing the AOS GUI [6] is now obsolete.

The main task of the AOS GUI is informative: it provides access to a set of parameters which allow the operator to verify the good health of the AO system and the correctness of the operations.

Other than that, a small number of buttons allows the TO to perform the main housekeeping operation on the AO System: switch on/off the WFS, switch on/off the Adaptive Secondary and so on.

The AOS GUI includes also a command panel (usually hidden) from which commands usually received from TCS subsystems can be manually sent to the AOSup.

[27] Usually: `/lbt/.....`

Table 13: `DXZernikes.dat` and `SXZernikes.dat`

```
# Gains for Zernike values sent by SetZernikes command
#
#  z1    z2    z3    z4   z5    z6    z7    z8    z9    z10   z11   z12   z13   z14   z15   z16   z17   z18   z19   z20   z21   z22
  0.0   0.0   0.0   0.0  1e-9  1e-9 -1e-9 -1e-9 -1e-9 -1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9  1e-9
```

11.1 Information panel

In figure 3 a snapshot of the main panel of AOS GUI is .

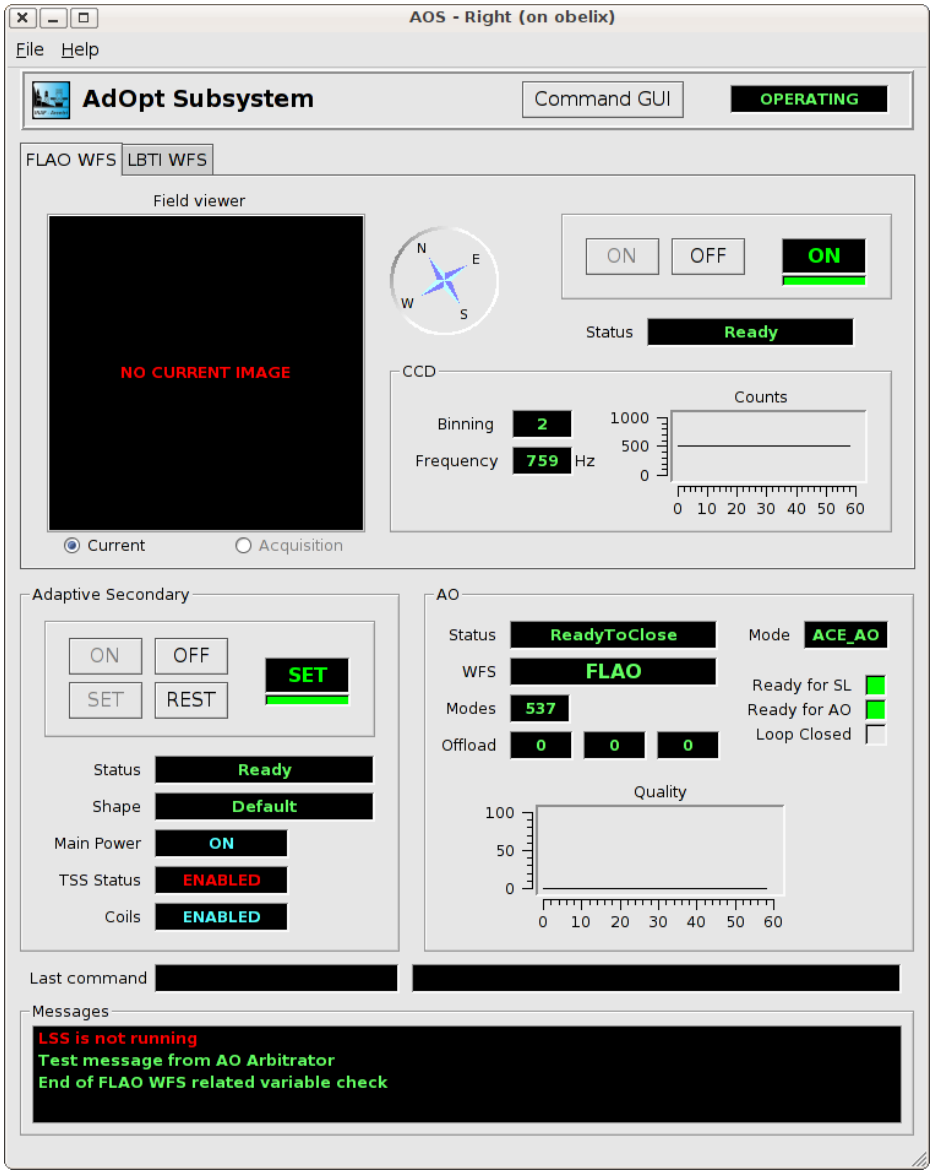

Figure 3: AOS GUI information panel

Starting from top we may identify 4 main sections:

1. **Wavefront sensor section**[28], displaying data related to the Wavefront sensor. It includes a snapshot of the technical viewer[29], a section for the control of the WFS hardware with a few

[28] As you may notice in the figure, AOS GUI is already provided with a panel for the control of the LBTI Wavefront Sensor. This part is not yet operating.

[29] Due to operating requirements of the WFS subsystem, the technical viewer will receive enough light to generate a useful image only before the AO loop is closed. Thus, the TV image will not be available during the observation.

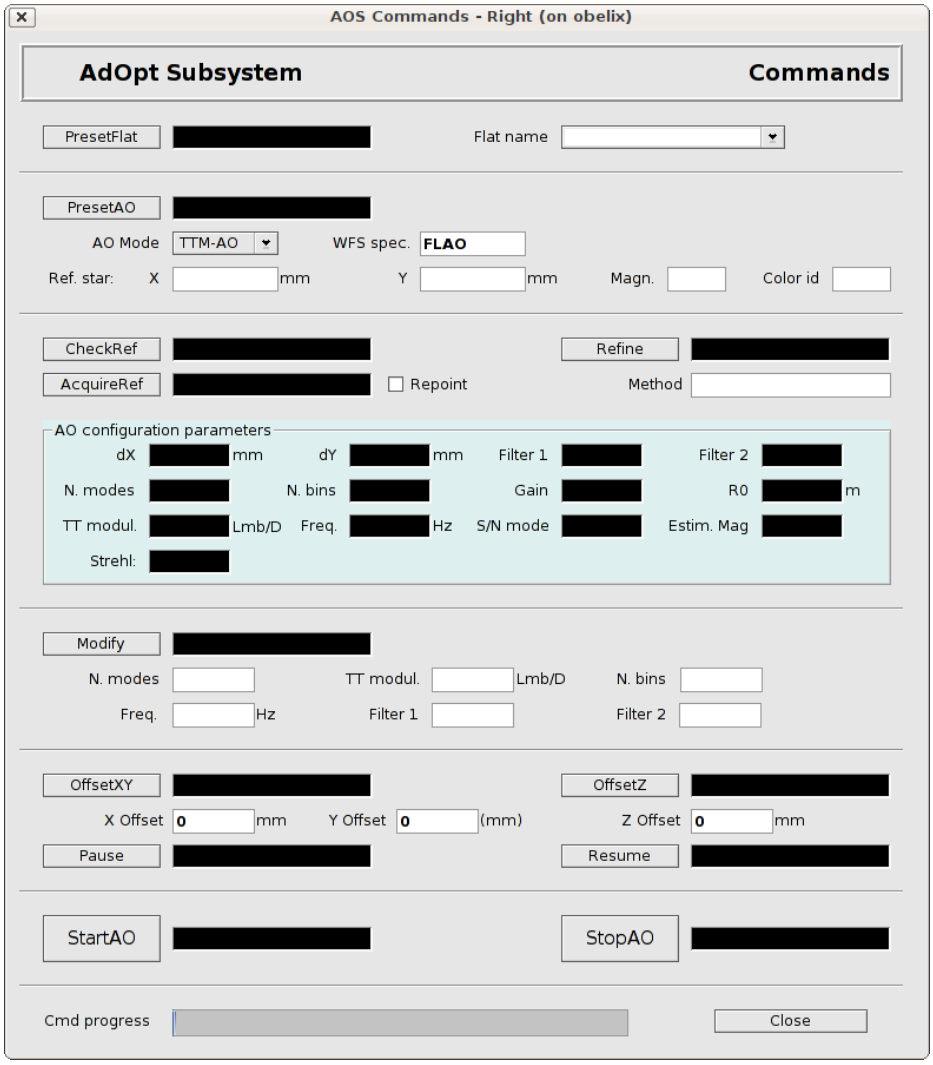

Figure 4: AOS GUI command panel

elements showing the current status, and a CCD subsection showing parameters related to the WFS-CCD.

2. **Adaptive secondary section**, containing a section for the control of the AdSec hardware, and a section with a number of status parameters.

3. **AO system section**, to the right of the AdSec section, displaying data and parameters related to the whole Adaptive Optics System. It includes an operating mode indicator, the number of corrected modes, three values showing the amount of "offload" currently required[30]. It also include a temporal graph of an image quality indicator (Strehl ratio). Three more on/off indicators provide some global status information.

4. **Log & Ctrl section**, containing items related to the management of the AOS subsystem. It will display status information, such as the current command in execution, messages and errors.

[30]The two tip-tilt components and the coma.

11.2 Command panel

Figure 4 shows a snapshot of the command panel available in the AOS GUI.

The purpose of the command panel is twofold.

It provides the capability to test the AOS and the interactions with the AOSup independently from the rest of TCS and, which is more interesting during the operative life of the system, allows an operator a manual intervention when some procedure fails for unexpected reasons.

The command panel provides a set of buttons which correspond to most of the operating commands defined for the AOS described in sections 12 through 14.

A few of the commands have associated arguments which may be modified before sending the command. Because the panel is intended either for debugging or for unexpected conditions, there are hardly preliminary checks applied to commands issued from the panel or to values sent as arguments. All the security checks are performed by the AOSup and any illegal or potentially dangerous command is simply rejected with an error message.

Part II

AOS Operational Commands

12 Introduction

In the following sections we will describe in full details the set of commands defined by AOS to allow the TCS to operate the AO System[31].

The main task of AOS is to translate each command received from TCS subsystems into a proper command (or sequence of commands) to be sent to the AO System for execution. For doing that AOS will obviously use the set of commands defined by the AO Arbitrator and already briefly described in section 1.2.

You may note that in most cases the translation between AOS commands and the corresponding AOArb commands is essentially a 1-to-1 mapping: the AOS command is directly implemented as a call to the corresponding AO Arbitrator command, sometimes after some adjustment of the arguments.

In a few cases the implementation of the AOS command is slightly more complex and may require intermediate operations before the AO Supervisor command is actually called.

Anyway, most AOS command have a corresponding AO Arbitrator command with the same name. In the following pages whenever a command name may be ambiguous, we use the prefix *AOArb:* to clearly identify commands to be sent to the AO Arbitrator.

All AO Arbitrator commands are executed in a synchronous way: after the call they will request the AO Supervisor to execute the related operations and will return after completion with either a success or failure status. Some command may also provide additional return values.

AOS Commands are defined as standard TCS commands by a specific interface definition[32].

The set of commands can be divided in two groups:

1. **Housekeeping commands**: which are provided to allow the TO to operate the AO System devices (switch on, switch off, etc.). These commands are listed in table 14.
2. **Observation commands**: which are provided to support the AO operation during an observation. These commands are listed in table 15.

AOS commands are described with many details in the following sections.

[31]The definition of the set of commands is the result of discussions with various people both from Arcetri and from Tucson: C. Biddick, N. Cushing, M. De La Pena, S. Esposito, R. Green, J. Kraus, D. Miller, A. Puglisi, A. Riccardi, P. Salinari, M. Wagner, with a special mention to J. Hill.

[32]See code in source file `.../aos/commands/AOSCommands.cpp`.

Table 14: AOS Housekeeping Commands

TCS comm.	AOArb comm.	Description
AdsecOn	AdsecOn	Set AdSec on
AdsecOff	AdsecOff	Set AdSec off
AdsecSet	AdsecSet	Set AdSec to status "set" (operating and with default shape)
AdsecRest	AdsecRest	Set AdSec to status "rest" (operating in safe position)
WfsOn	WfsOn	Set WFS on
WfsOff	WfsOff	Set WFS off

Table 15: AOS Observation Commands

TCS comm.	AOArb comm.	Description
PresetFlat	PresetFlat	Preset AO System for seeing limited operation
PresetAO PresetAOg	PresetAO	Preset AO System for adaptive operation
AcquireRefAO	AcquireRefAO CheckRefAO	Acquire the reference star and become ready for closing the AO loop
CheckRefAO	CheckRefAO	Returns offset values between reference star actual and nominal position
RefineAO	RefineAO	Perform optimization of AO loop parameters
ModifyAO	ModifyAO	Modify some AO loop parameter
StartAO	StartAO	Start the AO mode (i.e.: close the AO loop)
OffsetXY OffsetXYg	OffsetXY	Offset AO pointing
OffsetZ	OffsetZ	Offset AO focus
CorrectModes	CorrectModes	Apply mirror shape correction
SetZernikes	SetZernikes	Apply mirror shape correction
Stop	Stop	Stop current operation
Pause	Pause	Temporarily suspend current operation
Resume	Resume	Resume suspended operation
setNewInstrument		Preset parameters related to an instrument

13 Housekeeping Commands

Housekeeping commands are intended to be used by the Telescope Operator to perform general operations on the AO System such as switching on and off the AO subsystems, and the like.

They are generally made available to the TO as "buttons" on the AOS GUI. Although it is possible, there is no reason for the IIF to provide access to these commands directly.

13.1 AdSecOn

The AdSecOn command directly invokes the *AOArb:*`AdSecOn` command to switch on the AdSec Subsystem. After receiving this command the AO System will switch on and initialize the Adaptive Secondary and put it in "safe" status; i.e.: ready for subsequent operations but with the thin shell in "safe" position[33].

The command has neither input arguments nor return values.

13.2 AdSecOff

The AdSecOff command directly invokes the *AOArb:*`AdSecOff` command to switch off the AdSec Subsystem. After receiving this command the AO System will put the thin shell in "safe" position and then switch off the hardware devices.

The command has neither input arguments nor return values.

13.3 AdSecSet

The AdSecSet command directly invokes the *AOArb:*`AdSecSet` command to put the thin shell in ready position[34]. After receiving this command the AO System will apply the default seeing limited shape to the shell.

The command has neither input arguments nor return values.

13.4 AdSecRest

The AdSecRest command directly invokes the *AOArb:*`AdSecRest` command to put the thin shell in "safe" position.

The command has neither input arguments nor return values.

13.5 WfsOn

The WfsOn command directly invokes the *AOArb:*`WfsOn` command switch on the WFS hardware.

The command has neither input arguments nor return values.

13.6 WfsOff

The WfsOff command directly invokes the *AOArb:*`WfsOff` command switch off the WFS hardware.

The command has neither input arguments nor return values.

[33] Safe position for the thin shell of the Adaptive Secondary means that it is "sucked" against the permanent magnets. In this state the mirror shape is not ready for seeing limited operations.

[34] This means that the Adaptive Mirror is ready for seeing limited operations. A precalibrated "flat" shape is actively maintained.

14 Observation Commands

14.1 PresetFlat

This command is issued to request the AO System to prepare itself for the FIX-AO[35] mode of observation. This mode doesn't require a reference star, but it is possible to select among several different precalibrated "flat" shapes depending on the instrument in use.

Note also that the AO system will always startup automatically and preset itself with a default flat shape even before it is connected to the AOS. So the PresetFlat command is only useful if a specific shape different from the default one has to be set.

Table 16: *AOArb:*`PresetFlat` command input arguments

Name	Type	Units	Comment
`FlatSpec`	string		Specification of the desired "flat"

Table 17: *AOArb:*`PresetFlat` command return values

Name	Type	Units	Comment
`FlatSpec`	string		Specification of the selected "flat"

PresetFlat command input arguments are specified in table 16.

The AOS will directly issue an *AOArb:*`PresetFlat` to the AOArb, then will wait for completion.

The `PresetFlat` command, when successful, returns a single value: the name of the selected flat shape (see table 17).

14.2 PresetAO / PresetAOg

This command comes in two flavors. The `PresetAO` version is intended to be used by instrument software (through the IIF), the `PresetAOg` version is suited to be called by the AOS GUI. In both cases the command is issued to prepare the AO System for an observation in adaptive mode, i.e.: one of TTM-AO, ACE-AO, ICE-AO[35].

The main difference between the two flavors is the way to specify the reference star position:

- **PresetAO**: requires the specification of the reference star by means of a `Position` object which contains all the details required to define the star (coordinates, magnitude, etc.).
- **PresetAOg**: requires the specification of reference star details by means of individual arguments separately (which are usually entered in corresponding field of the command panel of AOS GUI[36]).

[35] See sect 1.2.2.
[36] See section 11.

From the implementation point of view when `PresetAO` is called the required values are extracted, or computed, from the `Position` object and then `PresetAOg` is called.

In `PresetAOg` the remaining arguments are added and then the function *`AOArb:`*`PresetAO` is called. Table 18 shows the input arguments of the function. Note that a number of input arguments are not mandatory and can have "null" values[37].

Table 18: *`AOArb:`*`PresetAO` command input arguments

Name	Type	Units		Comment
`AOMode`	string		Req.	Either "TTM-AO" or "ACE-AO" or "ICE-AO"
`Instr`	string		Req.	Specifies the instrument currently authorized (e.g.: `IRTC`)
`focStation`	string		Req.	Specifies the focal station currently authorized (e.g.: `bentGregorianFront`)
`WFS`	string		Req.	Specifies the source of WFS data(e.g.: FLAO)
`SOCoords`	real[2]	mm	Opt.	Position of the scientific object in focal plane coordinates
`ROCoords`	real[2]	mm	Req.	Position of the reference object in focal plane coordinates
`Elevation`	real	radians	Req.	Telescope elevation[38]
`RotAngle`	real	radians	Req.	Angular position of rotator[b]
`GravAngle`	real	radians	Req.	Angular position of rotator with respect to gravity[b]
`Mag`	real	TBD	Opt.	Magnitude of reference star
`Color`	real	TBD	Opt.	Color Index of reference star
`R0`	real	TBD	Opt.	Estimated value of R0
`SkyBrghtn`	real	TBD	Opt.	Sky brightness
`WindSp`	real	TBD	Opt.	Wind speed
`WindDir`	real	TBD	Opt.	Wind direction

Upon receiving the command the AO System will perform all set up operations needed to prepare for the acquisition of a reference star. These include all the operations which can be done without having the light of the reference star available. The remaining part of setup will be requested with the `AcquireRefAO` command[39].

When the `PresetAO` command has been completed and the telescope has reached the pointing position and is tracking and guiding on the target requested by the instrument, a command `AcquireRefAO` or `CheckRefAO` may follow.

14.3 AcquireRefAO

The `AcquireRefAO` command is usually issued after a `PresetAO` in order to request the AO System to proceed to reference object acquisition.

[37] The empty string is used as "null" value for string arguments and *NaN* is used as "null" value for numerical arguments.

[39] The sequence of low level operations needed to setup the AO System has been split in two steps (typically: `PresetAO` followed by `AcquireRefAO`) in order to allow the AO System to perform potentially time consuming adjustments (such as moving the mechanical assets of the WFS) while the telescope is slewing to point the source.

The acquisition of the reference star[40] can be performed in two ways: 1) by moving the WFS internal stages properly, or 2) by adjusting telescope pointing. The `AcquireRefAO` command is thus provided with an argument to select the desired acquisition method.

If the request is for "repointing" mode, the following sequence of operations is performed:

1. An *AOArb:*`CheckRefAO`[41] command is issued to evaluate the amount of XY offset needed.
2. A command to adjust pointing is issued to PCS.
3. An *AOArb:*`AcquireRefAO` command is issued to actually perform reference star acquisition.

Because of the repointing made at step 2, the following *AOArb:*`AcquireRefAO` command is not expected to require the movement of WFS internal stages.

If telescope repointing is not wanted only step 3 above is performed. In this case, the pointing errors are compensated in the AO System by moving the WFS internal stages.

In either case the AOSup internal procedure, after putting the reference star in the right spot, will compute parameters needed for optimization of the AO loop and set up all the needed optical devices.

After successful completion, the *AOArb:*`AcquireRefAO` function will send back the computed AO loop parameters as detailed in table 19.

Table 19: *AOArb:*`AcquireRefAO` command return values

Name	Type	Units	Comment
ΔX	real	mm	Pyramid X displacement with respect to nominal position
ΔY	real	mm	Pyramid Y displacement with respect to nominal position
`slNull`	string		Selected slope null
`NModes`	int		Number of corrected modes
`Itime`	real	s	CCD integration time
`Nbins`	int		CCD binning
`TTMod`	real	TBD	Tip-Tilt internal mirror modulation
`F1spec`	string		Selected position of filter wheel # 1
`F2spec`	string		Selected position of filter wheel # 2
`Strehl`	real		Measured Strehl ratio in J,H,K bands
`R0`	real	TBD	Measured R0
`MSNratio`	real[672]	TBD	Measured S/N per mode

After receiving the parameter block, the AOS will usually issue a `StartAO` command. In more complex cases the command might be followed also by a `RefineAO` or a `ModifyAO` command (see sections 14.5 and 14.6).

Note the ΔX and ΔY arguments. They return the amount of displacement of the WFS stages needed to put the source on top of the pyramid[42]: in this mode they can be used to evaluate the errors in the pointing model and, possibly, to correct it.

[40] This operations essentially consists in putting the reference star light on top of the WFS pyramid.
[41] The `AOArb:CheckRefAO` command can be used also independently and is described in section 14.4.
[42] Obviously in "repoint" mode the returned values should be zero.

14.4 CheckRefAO

This command can be used to measure the offset between the reference star nominal and actual positions.

Table 20: *AOArb:*`CheckRefAO` command return values

Name	Type	Units	Comment
ΔX	real	mm	Pyramid X displacement with respect to nominal position
ΔY	real	mm	Pyramid Y displacement with respect to nominal position

It can be used to perform calibrations or to evaluate collimation parameters and it is not probably useful during usual observations, although it may be called as part of the reference star acquisition procedure described in section 14.3.

This command will simply issue an *AOArb:*`CheckRefAO` command to the AO Arbitrator. This will instruct the AOSup to take an image with the technical viewer, identify the reference star, which is supposed to be within the TV field, compute the ΔX and ΔY values and return them back.

After successful completion, the command returns the reference star position as detailed in table 20.

14.5 RefineAO

The `RefineAO` command[43] is used to request the AO system to perform better estimation of the AO loop parameters. It may be issued after the `AcquireRefAO` and will start the AOSup procedure[44] to optimize the selection of parameters.

Table 21: *AOArb:*`RefineAO` command input arguments

Name	Type	Units	Comment
`Method`	string		Optimization method

The command has a single argument which may be used to select a specific optimization method (see table 21).

This command can be issued only in reply to the successful completion of a previous *AOArb:*`AcquireRefAO` command. It is thus assumed that the reference object selected in the previous *AOArb:*`PresetAO` command is still correctly positioned.

Upon receiving the command AOSup will perform the optimization procedure and reply with the same parameter block described for the *AOArb:*`AcquireRefAO` command (see table 19).

[43] The `RefineAO` command has been defined but the corresponding procedures have not yet been fully implemented in the AO Supervisor.

[44] The AO loop parameters optimization procedure evaluates the performances of the AO for a small interval of parameters values in order to determine the optimal set. It may require several minutes to complete, so it is offered as an optional function.

14.6 ModifyAO

The `ModifyAO` command[45] has been defined to support the ICE-AO operating mode. It may be used to request the AO System to modify the value of some AO loop parameter before closing the AO loop. The command must specify the set of AO loop parameters selected by the observer as detailed in table 22).

Table 22: *AOArb:*`ModifyAO` command input arguments

Name	Type	Units	Comment
`NModes`	int		Number of corrected modes
`Itime`	real	s	CCD integration time
`Nbins`	int		CCD binning
`TTMod`	real	TBD	Tip-Tilt internal mirror modulation
`F1spec`	string		Selected position of filter wheel # 1
`F2spec`	string		Selected position of filter wheel # 2

This command can be issued only in reply to the successful completion of a previous AcquireRefAO and/or RefineAO command. It is thus assumed that the reference object selected in the previous `PresetAO` command is still correctly positioned.

AOSup will perform a check of the validity of parameters[46], recompute the optimization values, and will reply with the same parameter block described for the `AcquireRefAO` command (see sect. 14.3).

The calling procedure can in principle issue an arbitrary number of ModifyAO commands before requesting the closing of the AO loop with a StartAO command. The limitation being the capability of the telescope to track the reference star for the time needed for the operation.

14.7 StartAO

The `StartAO` command is issued by AOS to request the closing of the AO loop. It doesn't require any parameter in that it is only provided to synchronize the AO operation with the scientific instrument operation and when it is issued the AO System must be fully ready to close the AO loop.

AOS will simply call the corresponding *AOArb:*`StartAO` command and wait for a reply.

14.8 OffsetXY / OffsetXYg

Also this command, as `PresetAO`, comes in two flavors. The `OffsetXY` version is intended to be used by instrument software (through the IIF), the `OffsetXYg` version is suited to be called by the AOS GUI.

The difference between the two flavors is in the way to specify the amount of offset:

[45] The `ModifyAO` command has been defined but the corresponding procedures have not yet been fully implemented in the AO Supervisor.

[46] Parameter checking will be particularly strict for this command in order to ensure safe operation of the AO system.

- **OffsetXY**: requires the specification of the focal plane coordinates of the new pointing position.
- **OffsetXYg**: requires the specification of the offset value in mm in the focal plane reference.

When `OffsetXY` is used, the coordinates of the reference star got from the previous `PresetAO` command are subtracted from the coordinates provided as arguments and then `OffsetXYg` is called.

The AOS command `OffsetrXYg` in turn will issue an AOArb:`OffestXY` command to request the offsetting of the AO System (i.e.: move the WFS stages).

The AO arbitrator command requires two delta position values as detailed in table 23.

Table 23: *AOArb:*`OffsetXY` command input arguments

Name	Type	Units	Comment
`DeltaX`	real	mm	Requested X position offset
`DeltaY`	real	mm	Requested Y position offset

The command has different effects depending on the current mode. When the AO loop is closed any offset applied to the WFS stages will be corrected by the AO system and this will result in an offset of the field on the scientific camera[47]. When the AO loop is not closed (e.g.: after a `Pause` command) it will simply move the WFS stages.

14.9 OffsetZ

This command is issued in order to offset the focus position of the AO System (i.e.: by moving the WFS stages).

The AOS will directly call the corresponding *AOArb:*`OffsetZ` command.

If the command is issued when the AO loop is closed, it will be compensated by the AO system, thus it will result in a focus offset in the focal plane of the scientific camera; if the command is issued when the AO loop is open, it will simply move the Z stage of WFS.

The command requires one delta value as specified in table 24.

Table 24: *AOArb:*`OffsetZ` command input arguments

Name	Type	Units	Comment
`DeltaZ`	real	mm	Requested focus offset

[47]Offsets in closed loop mode are compensated by AdSec with tip-tilt movements, which will need to be offloaded to telescope pointing correction by the mode offload mechanism (see section 8). Due to round-trip delays this means that only offsets of the order of about one half arcsec can be done in a single step. Bigger offsets must be either done in small steps with proper delay between them, or must be implemented by pausing the AO loop (see section 14.11) sending an OffsetXY and concurrently offsetting the telescope pointing, and then resuming the AO loop (see section 14.12).

14.10 CorrectModes

This command is used to apply a modal correction to mirror shape when the AO System is in Closed Loop (e.g.: to correct non common path aberrations).

A vector of Δ values must be specified (see table 25).

Table 25: *AOArb:*`CorrectModes` command input arguments

Name	Type	Units	Comment
`DeltaM`	real[672]	TBD	Modes correction vector

AOS will directly issue the corresponding AOArb command *AOArb:*`CorrectModes` and wait for completion.

14.11 Pause

This command temporarily suspend the AO loop; the AO System must remain ready to resume, i.e.: to close again the AO loop.

AOS will directly call the corresponding *AOArb:*`Pause` command and wait for completion.

AOSup will do the proper action (i.e.: open the AO loop) and then acknowledge the request. The command may be followed by either an `OffsetXY` or a `Resume` or a `Stop` command.

14.12 Resume

Resumes a suspended AO loop after a `Pause`. AOS will directly call the corresponding *AOArb:*`Resume` command and wait for completion.

The loop can be resumed successfully only if during the pause the telescope tracking (or guiding) system is able to maintain the correct pointing of the reference star.

14.13 Stop

This command is issued to stop the current operation. After this command any setting defined by a previous `PresetAO` command will be canceled.

AOS will directly issue the corresponding *AOArb:*`Stop` command.

Table 26: *AOArb:*`Stop` command input arguments

Name	Type	Units	Comment
`Msg`	string		Reason for stopping

As shown in table 26, the command requires a string parameter containing a description of the reason for stopping (the argument is intended to provide the AO Supervisor with info related to the reason for stopping, e.g.: for use in the logging files).

14.14 SetZernikes

This command is used to apply a modal correction to mirror shape when the AO System is in Seeing Limited mode.

AOS will directly call the corresponding *AOArb:*`SetZernikes` command. A vector of Δ values must be specified (see table 27).

Table 27: *AOArb:*`SetZernikes` command input arguments

Name	Type	Units	Comment
`DeltaM`	real[672]	TBD	Shape correction vector

This command is conceptually very similar to the `CorrectModes` command described in section 14.10, but is to be used when the AO System is operating in seeing limited mode to apply statical corrections to the mirror shape. It will actually result in very different operations at the AO System side[48].

14.15 setNewInstrument

This command has no AO Arbitrator counterpart. It is implemented because all TCS subsystem are required to be able to receive `setNewInstrument` commands.

Based on the specified instrument and focal station the AOS will preset internal variables which will be used to provide the required information when an *AOArb:*`PresetAO` command will be actually issued (see section 14.2).

The command will affect the following internal variables:

- `Instrument`: stores the authorized instrument identification string.
- `FocalStation`: stores the authorized focal station identification string.
- `WfsSpec`: based on specified instrument and focal station this variable is set to the correspondent WFS specification[49].

[48] In Seeing limited mode the corrections are applied to the mirror shape and involve only the AdSec. In Diffraction limited mode the correction are applied by modifying the "slope null" vector in the slope computer.

[49] This feature has been added to support different optical configurations, such as the one needed for LBTI operations.

A Source Files Identification

The main AOS related source files are listed and briefly described in table 28.

Table 28: AOS Source Files

`AOSAoApp.cpp` `AOSAoApp.hpp`	The core of the AOS. Together with code in `AOSHandlers` implements all the functionalities of AOS.
`AOSClear.cpp` `AOSClear.hpp`	Contains various function to clear or preset the Data Dictionary variables.
`AOSConfig.cpp`	Contains the code for methods to get data from configuration files.
`AOSEvent.cpp` `AOSEvent.hpp`	Contains classes derived from the TCS Event class to manage all events used in AOS.
`AOSTelemetry.cpp` `AOSTelemetry.hpp`	Contains the code to manage the telemetry recording.
`AOS.cpp` `AOS.hpp`	Implementation of the AOS subsystem framework. It essentially provides for the management of the thread. The actual functionalities are implemented in `AOSAoApp`.
`AOSHandlers.cpp`	Contains the code for handling messages sent by the AOSup.
`Main.cpp`	Just launches the subsystem task and cleans up at the end.
`VarUtils.cpp` `VarUtils.hpp`	Contain code for the implementation of variable mirroring back and forth.
`Version.hpp`	Defines the version number and maintains documentation of the history of modifications.

B Interface with AOSup

The code needed to interface AOS with the AOSup is contained in a directory subtree rooted in `../aos/aosupervisorlib`.

In other words the TCS source code tree contains all source files needed to properly compile and build the AOS executable.

Although modification of the interface is done with the greatest care, in order to avoid incompatibilities in the communication, it may be sometimes necessary to upgrade the interface code to align it with new features added to the AOSup code.

This is essentially a manual operation, because it might require changes in the organization of source files, anyway whenever modifications are limited to file content and no change in file organization has been made, a specific shell command (`update`) is included in the source code tree for updating the files. This anyway requires that the AOSup source tree is available and properly installed.

C Emulator and test procedures

In the AO Supervisor section of the source tree quoted in section B is contained a subdirectory (`../aos/aosupervisorlib/emulator`) which is not used for the building of AOS, but contains the source files needed to build two programs and some procedures which can be used for an offline test of the AOS. The purpose of the code contained there is to build an emulator and the required support process (`MsgD-RTDB`) which can take the part of the AO Supervisor to allow a preliminary test of the AOS when the actual AO Supervisor system is not available.

The building and usage of the AOSup emulator is described in a `README` file included in the set, which is copied here for your convenience:

```
This directory includes files needed to build an emulator of the AO Supervisor
to be used in software tests of AOS.

The Emulator is made up of three components:

msgdrtdb: a standalone version of the Message Daemon

thrdtest: a test program used to emulate the Ao Arbitrator

aostest.p: a command procedure to be fed to thrdtest to
emulate Ao Arbitrator operations.


BUILD PROCEDURE

The command 'make' issued on this same directory compiles and builds the two
executables: msgdrtdb and thrdtest.


RUNNING ENVIRONMENT

In order to properly communicate, the AOS must know the IP number of the
computer running msgdrtdb. This is specified in file 'lbt.conf' as parameter:
AOSLMsgDIp (left) and AOSRMsgDIp (right). The correct value must be set before
starting AOS.


HOW TO RUN THE TEST

1. Start TCS as usual


2. Start AOS and AOSGUI. The connection status indicator (top right on the GUI)
   shows: NO CONNECTION


3. start MsgD with:    source start_msgd.sh
```

```
   The connection status indicator on AOSGUI now shows: NO ARBITRATOR

   If the indicator remains in DISCONNECTED status, there is no communication
   between AOS and MsgD. You can verify the IP number actually used by AOS
   from the 'Help->About' panel in AOSGUI. The AO Supervisor IP number must
   be the one of the workstation where MsgD is running (usually: 127.0.0.1).


4. Start the test program with: ./thrdtest -aos right

   Use right of left as required.

   If the program starts correctly you should see the following:

AOARB.R: 6.6 [TH](Built: Oct 20 2010 12:13:51) - Dbg lev.:0, quiet, Line edit & history:No

Trying to connect to MsgD @ 127.0.0.1:9752

AOARB.R cmd:

   If you get a communication error, then MsgD is not running on this
   workstation.

   The prompt will show either AOARB.R or AOARB.L for right and left.


5. At the AOARB.x prompt start the command procedure with: exec aostest R

   Use "exec aostest L" for left side. The case is meaningful.

   You can now follow instructions and prompts from the procedure.

   NOTE: if you do not see the expected results during the execution of the test
   procedure, first verify you have specified the same side on all relevant
   commands.


NOTE: if the procedure is not executed to the end (e.g.: it has been interrupted
with CTRL-C) you must manually kill the message daemons. You can use the kill
command on the two processes named "msgdrtdb".
```

References

[1] Luca Fini, Lorenzo Busoni, and Alfio Puglisi. AOS Functional Description. Technical Report 481f300, INAF-Arcetri, May 2009.

[2] Roberto Biasi, Mario Andrighettoni, and Daniele Veronese. LBT672 Design Report: Electronics. Technical Report 641a006, Microgate, May 2008.

[3] Luca Fini, Alfio Puglisi, and Lorenzo Busoni. Integration of the AdOpt Software into TCS. Technical Report 486f004, INAF-Arcetri, Nov 2005.

[4] Lorenzo Busoni, Simone Esposito, Luca Fini, Alfio Puglisi, Armando Riccardi, and Marco Xompero. AO Supervisor - Functional Description. Technical Report 486f009, INAF-Arcetri, Mar 2009.

[5] Douglas Miller. Adaptive Optics Subsystem (AOS) GUI Requirements. Technical Report 481s302, LBTO, Dec 2009.

[6] Luca Fini and Francesco Tribioli. AOS GUI Description. Technical Report AdOptSW.010, INAF-Arcetri, Dec 2007.

Doc_info_start
Title:
Document Type: Specification
Source: Osservatorio di Arcetri
Issued by: Luca Fini
Date_of_Issue: 28 Mar 2011
Revised by:
Date_of_Revision:
Checked by:
Date_of_Check:
Accepted by:
Date_of_Acceptance:
Released by:
Date_of_Release:
File Type: PDF
Local Name:
Category: 400
Sub-Category: 480
Assembly: 481 Telescope Control Software
Sub-Assembly:
Part Name:
CAN Designation: 481f301
Revision: c
Doc_info_end