ARIES21, the control command line shell for the 40m radiotelescope

P. de Vicente

Informe Técnico IT-OAN 2008-9

Change Record

Version	Date	Author	Remarks
1.0	14-12-2007	P. de Vicente	First version
1.1	10-08-2009	P. de Vicente	Upgraded
1.2	23-09-2011	P. de Vicente	Upgraded

Índice

1. Introduction

The ARIES21 (Antena Radioastronómica Milimétrica para el siglo 21) 40 m antenna at the OAN facilities in Yebes is controled by an interactive shell based on iPython called "aries". This shell is a client application whose philosophy mimics "apecs" (APEX command control system). Some of the available commands have the same names as the "apecs" ones but are implemented in a different way.

As with APEX, the ARIES Control Software is based on the Alma Common Software Infrastructure. All the hardware and software elments are implemented as software modules whose interface is published and available to all the other modules using CORBA calls via the ACS as a transport layer.

"aries" is an interactive iPython shell with commands that allow to create high level typical astronomical observations and some of the most common commands to operate the antenna and set up some instruments.

2. Philosophy of the commands

The interactive shell uses iPython and as such inherits all its features. Within the aries command line interface, it is possible to use all Python packages installed in the computer. This allows to create powerfull scripts or macros for automatic and repetitive operations. In order to load a python module the python `import` instruction should be used.

2.1. General form of commands

All commands follow the Python syntax; a typical command is a name followed by an opening parenthesis, parameters and finished by a closing parenthesis. For example:

```
command(parameter1='option1', parameter2=double_value, parameter3=[Python list])
```

Parameters for commands may be of any valid Python type. It is possible to pass parameters without specifying the name of the parameter, For example:

```
command('option1', double_value, [Python list])
```

but in this case parameters should be sorted according to the expected order of the command. By specifying the parameter names it is possible to give parameters in any order, like here:

```
command(parameter2=double_value, parameter3=[Python list], parameter1='option1')
```

2.2. Help for the commands

It is possible to obtain help from the commands by typing:

```
help command
```

The help will show the help and the last parameter values used with this command.
All available commands for controling the antenna are obtained by typing:

```
help aries
```

They are organized in several subsets: general, catalogs, instruments, target, pattern, antenna, and engineer

2.3. History and auto completion

iPython keeps the history of the commands and therefore remembers them. It is possible to recover them by typing the first letters of the command and pressing the "up" arrow. The first letters should be enough to avoid ambiguity. IPython history is stored by default in file `history` located in folder `.ipython`.

The shell also has autocompletion and will show all commands that match the typed letters. To activate it, type the first letters, and press the "tab" keyword. When using autocompletion take into account that not only the aries commands are available but any command included in the imported python packages. Autocompletion will also show variables, both global and local ones created by the user, within the aries21 environment.

2.4. Errors

Some commands may generate a warning or an error which will be shown with a color code. Important errors will be displayed in red, normal errors in fucsia and warnings in turquoise. These errors are generated by internal procedure `errorLog()` which accepts two parameters: the line to be displayed and an integer code with the level of the error. For example, a severe error is displayed in this way:

```
errorLog("The observing engine component raised an exception which has been logged", 3)
```

aries21 also catches software errors generated when calling functions. The error system provides two levels of information, one for the user and one for the developer. This behaviour can be switched on or off with `warnings()` command which accepts one parameter (`on` or `off`):

```
warnings(on)
```

but will only affect commands typed after the switch. If the error is not catched, IPython will generate a stream of colored lines with the name of the called functions and the number of the line in the file. If this happens please inform the aries21 software developers (for the time being the author of this report).

3. Starting up the aries shell

The aries shell may be started from any place in the developer account by typing: `aries`. The command line interface requires several environment variables to operate correctly, which should be located in `.bash_profile`. Below we show an example where required variables are exported:

```
# Directory where data are stored
export ARIESDATA=$HOME/data
# Directory where modules are stored
export ARIESROOT=$HOME/introot
# Directory where catalogs are stored
export ARIESCATA=$HOME/aries21/Catalogs
#
export ARIES_PROJECT_ID="N-00.C-0004-2008"
```

- Environment variable `ARIESCATA` is used to locate the directory where system catalogs are looked for
- `ARIES_PROJECT_ID` serves for classifying the projects. The identification for the project is a formatted string which will be setup by the staff of the observatory. During the commissioning of the antenna we are using `N-00.C-000X-20YY`. N stands for "internal OAN project". X is a digit which is incremented when the author estimates that tests for major version changes have been acomplished and new ones are required and YY is a two digit year number since 2000. In this way the data can be clasified minimizing the clutter.
- Data are stored in a different host and in folder `ARIESDATA` and in a subdirectory labelled with the `ARIES_PROJECT_ID` value.
- Environment variable `ARIESROOT` is required by the whole ACS system and not only by the aries shell.

When the command line interface starts, it launches a number of tasks.

1. It initializes all required components. This phase may take some time to complete and hence delay the start time.
2. It boots the antenna. This phase also takes some time since many actions are performed here. The boot time depends on the current antenna status.
3. It initializes the shell and therefore runs several procedures inside, like checking the NTP for servers "ariesobs", and "ariesacu" and looking for the DUT1 in the IERS internet server.
4. It asks for the operator and observer names. The startup pauses here until the user enters two letters for each request.
5. Finally it shows the status of the antenna, along several lines. One line per motor (or mirror) in which each line displays the status, position and position error for each motor.

If the `ARIES_PROJECT_ID` variable is not available, the shell will request a valid one to the user.

Below is included an example of a successful startup. If any phase described before does not complete successfully, a message will appear in the startup and the shell may throw errors and possibly will not complete the rest of the process:

```
This step will delay several seconds ....
Initializing required components....
Startup sequence for the antenna ....
   Setting tolerance to 2'' (HPBW = 18''). You may change it manually
   Setting tolerance to 8'' (HPBW = 72''). You may change it manually
   Setting tolerance to 8'' (HPBW = 72''). You may change it manually
   Setting tolerance to 19'' (HPBW = 189''). You may change it manually
Please tune the MITEQ to 12750 MHz to avoid interferences
   Setting tolerance to 27'' (HPBW = 265''). You may change it manually
```

```
Please tune the MITEQ to 12750 MHz to avoid interferences
   Setting tolerance to 32'' (HPBW = 318''). You may change it manually
   Setting tolerance to 70'' (HPBW = 696''). You may change it manually
   Setting tolerance to 50'' (HPBW = 497''). You may change it manually
   Setting tolerance to 14'' (HPBW = 133''). You may change it manually
Equatorial offsets cannot be used with a horizontal source. Resetting reference offsets
  Reference position: 0.000000 0.000000 arcsecs
Re-setting focus offsets to 0.0
   Pointing corrections due to focus are: 0.000000 0.000000
   (focus) Final pointing corrections are: 0.000000 0.000000
   Re-setting accumulated pcorr CA (Az) offset to 0.0 arcsec.
   Re-setting accumulated pcorr IE (El) offset to 0.0 arcsec.

Welcome to the ARIES Command Line Interface v1.134 2011-06-22. (c) MPIfR, OAN
Type help(aries) to get help. Hit CTRL-D to quit.

NTP Time offset @ ariesobs is correct :  -0.00024 secs
NTP Time offset @ ariesfits is correct : -0.000011 secs
DUT1: -0.300000 secs
M1:     stopped. Current position:  312.43452, 18.15619
M2:     slewing. Current position:   0.00  -0.00  -0.00  -0.00   0.00 Errors:   0.00  -0
        El Table offset position:   -6.00  17.02  -8.45  -0.02   0.00
M2PS:   standby. Current position:    4.48964 Error:    0.00000
M3POS:  stopped. Current position:  161.85668 Error:    0.05329
M3R:    standby. Current position:   89.91217 Error:    0.08783
M4A:    standby. Current position:   -0.73566 Error:    0.00000
M4B:    standby. Current position:    0.98095 Error:    0.00000
```

4. Commands for operating the antenna

This is a list of low level commands that command the antenna. These commands usually take effect after typing them although some of them may require some time to complete. In that case the system will warn the user but will not block the prompt except in some cases.

4.1. `stow`

```
stow()
```

This command stows the antenna in a survival position and inserts the stow pins. The position for stow in elevation is 89.57 degrees. There are two available stow positions in azimuth: 45 and 405 degrees and this command will choose the nearest. The time to fully stow the antenna once it has reached its final position is 15 seconds. When issuing this command all pointing corrections (pointing model, manual pointing corrections and refraction) are disabled, any tracking table cleared, the vertex closed and the antenna is sent to the closest stow position. The user may get some information about these operations when issuing this command. In order to execute this command the main drives should be active first. Failure to do so will generate an error.

4.2. `unstow`

```
unstow()
```

This command unstows the antenna from the survival position withdrawing the stow pins. The time to fully unstow the antenna is 15 seconds. The antenna remains in the stow position until another command sends it to a different position. In order to execute this command the antenna does not need to be active first. This command is not the opposite to the "stow()" command, since neither the vertex is opened nor the pointing model is loaded.

4.3. `reset`

```
reset('motor_system')
```

This command resets the specified motor system. It should be used when a motor drive generates an error and once it has been fixed. Sometimes the error does not require any operation to be performed by the OAN personnel since the error may be temporary. This command can be used in any circumstance (even if there are no errors present) if the motor drive is not active. After the command the motor remains in standby.

By default this command resets the main drives (azimuth and elevation). Other motor drives can be specified using the appropiate parameter. Allowed parameters are:

- 'm1' for the main drives,
- 'm2' for the subreflector drives,
- 'm2ps' for the primary - secondary motor drive,
- 'm3p' for the Nasmyth position drive,
- 'm3r' for the Nasmyth redirection drive which points towards M4 or M4',
- 'm4a' for the secondary Nasmyth redirection drive single dish branch,
- 'm4b' for the secondary Nasmyth redirection drive VLBI branch.

4.4. `activate`

```
activate('motor_system')
```

This command activates the specified motor system. By default it will activate the main drives, azimuth and elevation. Other motor drives can be specified as a parameter. Allowed parameters are:

- 'm1' for the main drives,
- 'm2' for the subreflector drives,
- 'm2ps' for the primary - secondary motor drive,
- 'm3p' for the Nasmyth position drive,

- 'm3r' for the Nasmyth redirection drive which points towards M4 or M4',
- 'm4a' for the secondary Nasmyth redirection drive single dish branch,
- 'm4b' for the secondary Nasmyth redirection drive VLBI branch.

Each motor drive may have a different activation time. Main drives tipically require 7 seconds for security reasons (the beacon is lighted and the acustic alarm activated). All the other systems tipically require 1 second.

4.5. `standby`

```
standby('motor_system')
```

This command deactivates the specified motor system. Brakes are applied and amplifiers are switched off. By default it will deactivate the main drives: azimuth and elevation. Other motor drives can be specified as a parameter. Allowed parameters are:

- 'm1' for the main drives,
- 'm2' for the subreflector drives,
- 'm2ps' for the primary - secondary motor drive,
- 'm3p' for the Nasmyth position drive,
- 'm3r' for the Nasmyth redirection drive which points towards M4 or M4',
- 'm4a' for the secondary Nasmyth redirection drive single dish branch,
- 'm4b' for the secondary Nasmyth redirection drive VLBI branch.

4.6. `subref`

```
subref('m2tablemode')
```

This command loads an offset versus elevation M2 table. This table has been obtained after careful focus measurements at different frequencies, but was mainly determined at 22 GHz. Allowed parameters are:

- 'astro' for astronomy observations. All axis, except X move depending on elevation.
- 'geo' for geodetic VLBI observations. All axis remain stopped except Y which depends on elevation. In this way the focus remains unchanged at any position.

4.7. `local`

```
local()
```

This command transfers the control of the ACU from the shell to the Local Control Panel (LCP) in the control room. This command is very useful before closing the interactive shell and the whole ACS system.

4.8. remote

```
remote()
```

This command gets the control of the ACU from any Local Control Panel (LCP). If the control is in the Hand Held Panel (HHP) go first to the servo room and switch the HHP key to the central position. After this operation the control will be under the LCPs and it will be possible to issue a `remote()` command and get the remote control of the antenna.

4.9. checkControl

```
checkControl()
```

This command reports where the ACU control is. Valid answers are: “remote”, “LCP Control”, “LCP Servos”, “LCP Receivers”, “HHP”, “Diagnosis”.

4.10. closeVertex

```
closeVertex()
```

This command closes the vertex. The system takes 70 seconds to close the 8 petals, but it does not block the command shell.

4.11. openVertex

```
openVertex()
```

This command opens the vertex. The system takes 70 seconds to open the 8 petals, but it does not block the command shell. The command informs the user about the estimated time to fully open the vertex.

4.12. vertex

```
getVertex()
```

This command informs about the state of the vertex: “opened”, “closed”, “opening” or “closing”.

4.13. gotoHor

```
gotoHor('azimuth', 'elevation')
```

This command sends the antenna to a given horizontal position. Allowed positions for azimuth are -60 to 425, and for elevation 3 to 90. Positions out of this range will generate an error. The antenna moves at a maximum speed of 1 deg/s in elevation and 2 degs/s in azimuth to reach the commanded position. Pointing corrections already active are applied when using this command.

4.14. stop

```
stop('motor_system')
```

If this command is used without parameters it stops azimuth and elevation axis and clears any

tracking table. Sometimes this command is useful to reset previous observations in the ACU that could not finish or were not properly completed and cleared out. It is possible to pass a parameter to stop any other motor.

- 'm1' for the main drives,
- 'm2' for the subreflector drives,
- 'm2ps' for the primary - secondary motor drive,
- 'm3p' for the Nasmyth position drive,
- 'm3r' for the Nasmyth redirection drive which points towards M4 or M4',
- 'm4a' for the secondary Nasmyth redirection drive single dish branch,
- 'm4b' for the secondary Nasmyth redirection drive VLBI branch.

4.15. `cancel`

```
cancel()
```

This command stops the current scan and removes any future subscans within the scan, clears any tracking and superposition table and stops any movement in the azimuth and elevation axis. It is very useful to clear some errors in the ACU related to tracking tables, or to unexpected software errors in the observing engine. This is the command to abort a running observation.

4.16. `platform`

```
platform()
```

This command sends the antenna to position (45,5) after removing all pointing corrections (including the pointing model). At this position it is possible to use the elevator platform to access the subreflector. It is very important to use `standby()` once the antenna has reached its final position to avoid accidents.

4.17. `pcorr`

```
pcorr(azCol0='0', deltaEl0='0', mode='abs', origin='user')
```

Apply a manual pointing correction. Units are arcsecs. In the default mode input values are absolute. Corrections are immediately applied to the antenna and stored in the preparation scan. This means that if the antenna is executing a scan and a correction is sent, it is immediately applied, and this value will be stored in next scan information, not in the current one.

- `azCol` Collimation error in azimuth [arcsecs]. This is the horizontal angle in the sky.
- `deltaEl` Collimation error in elevation [arcsecs]. This is the vertical angle in the sky.
- `mode` Correction mode: "rel", corrections are relative to previous values, "abs", absolute, overriding the previous values.

- `origin` Correction origin. It may take values ”user” and ”focus”. In both cases the pointing correction is applied but the information is stored in a different place. Usually the required mode is ”user”.

4.18. `pcorrections()`

`pcorrections()`

This command shows the applied pointing corrections. Below we show an example:

```
ARIES> pcorrections()
Total pointing corrections (arcsecs): Az = 2516, El = -465 arcsecs
Refraction correction (arcsecs): El = 68
Model pointing corrections (arcsecs): Az = 2641, El = -602 arcsecs
Manual pointing corrections (arcsecs): ColAz = -94, El = 68 arcsecs
Focus pointing corrections (arcsecs): ColAz = 0, El = 0 arcsecs
Pointing model parameters(arcsecs): P1 = 2570, P2 = 54 P3 = 0, P4 = 0, P5 = 0, P7 = -955, P8 = 0 , P9 = 465
```

The corrections are displayed in several lines. Each line is self explanatory

4.19. `pcorr_reset`

`pcorr_reset()`

This commands sets manual pointing corrections to 0.

4.20. `pmodel()`

`pmodel(p1=0, p2=0, p3=0, p4=0, p5=0, p6=0, p7=0, p8=0, p9=0)`

Loads a pointing model immediately. Parameters are in arcsecs. See IT-OAN-2007-26 for a full explanation of the parameters used.

- `p1`. Azimuth encoder offset.
- `p2`. Azimuth collimation error and Nasmyth positioning errors.
- `p3`. Lack of orthogonality between azimuth and elevation axis.
- `p4`. Tilt of azimuth axis alonf an E-W direction.
- `p5`. Tilt of azimuth axis alonf a N-S direction.
- `p7`. Elevation encoder offset.
- `p8`. Gravitational deformation and Nasmyth positioning errors.
- `p9`. Gravitational deformation and Nasmyth positioning errors.

4.21. `refraction`

`refraction(mode='on')`

This command switches on/off the pointing correction due to refraction.

4.22. status

```
status()
```

This command shows the status of the antenna in different lines, using one line per motor. Each line displays the motor status, its position and the positioning error.

```
M1:     stopped. Current position:   97.20722, 57.30607
M2:     tracking. Current position:  -0.00   0.00   0.00   0.00  -0.00 Errors:  -0.00
        El Table offset position:   -6.00  -7.45 -21.15   0.00   0.00
M2PS:   standby. Current position:    4.48037 Error:    0.00000
M3POS:  stopped. Current position:  122.72752 Error:    0.03244
M3R:    slewing. Current position:   89.91424 Error:    0.08576
M4A:    standby. Current position:   -0.92655 Error:    0.00000
M4B:    slewing. Current position:    0.90490 Error:    0.00000
```

Focus corrections in M2 (subreflector), appear in the second line. The 5 values are X, Y, Z, tilt in X and tilt in Y.

4.23. fmode

```
fmode(mode)
```

Sets the antenna to primary or secondary focus. Mode can be any of these two possibilities: “secondary” or “primary”. It takes 4 minutes 30 seconds to switch from one mode to the other. The subreflector is set to 0, before moving from primary to secondary or viceversa. When the transition is completed, the subreflector is sent to its position (elevation dependent).

4.24. fpos

```
fpos(x, y, z, xt, yt)
```

Sets the position of the subreflector. Units are mm for x, y and z axis and degrees for xt and yt [degs]. Commanded values are absolute. By default this command sets all values to 0. It may take a long time to move the subreflector to its final position. Use command `status()` to obtain information on the position of the subreflector.

4.25. debug

```
debug(mode='on')
```

Switches on or off the debug level

4.26. warning

```
warning(mode='on')
```

Switches on or off the warning level

4.27. logAries

```
logAries(logline, target=0, color=False)
```

Logs information from aries into the opened log file. If the target is 0, it prints a line to the screen and to a log file, if it is 1, it only prints the line to the screen, if it is 2, it only prints the line to the logfile. The log file is controlled by ipython. Every session of aries21 opens a new file named aries21.log.xyz in the HOME directory, where xyz is a 3 digit number. The older the log the larger the number.

4.28. errorLog

`errorLog(logLine, errorLevel)` Prints the error log with the appropiate color according to level. errorLevel: 1 is a warning, 2 is an error, 3 is an important error.

5. General commands

5.1. operator_id

```
operator_id(id='NN')
```

This command is used to supply two initials for the antenna operator. This value is stored in the scan information and in the final FITS file with results.

5.2. observer_id

```
observer_id(id='NN')
```

This command is used to supply two initials for the observer. This value is stored in the scan information and in the final FITS file with results.

5.3. project_id

```
project_id(pid='O-PP.C-NNNN-YYYY', createDir=False)
```

Assigns a name to the current project. There is a folder where the results for each project are stored. If the directory does not exist, it may be created by setting the second parameter to `True`. The meaning of the letters is as follows:

- `O` origin of the project. Possible values are:
  - `N` OAN
  - `S` Spain
  - `V` VLBI
  - `X` external
  - `K` Key program
  - `A` Aries (staff time)

    - T Technical / Maintenance, / Calibration
- PP = Not appliable in this case. (ESO period.)
- C = ESO category (A-F):
    - A Cosmology
    - B Galaxies and Galactic Nuclei
    - C Interstellar Medium, star formation, Planetary Systems
    - D Stellar Evolution
    - E Undefined
    - F Undefined
- NNNN Sequential Number
- YYYY year

5.4. save_history

```
save_history(file='ariesHistory.py')
```
Writes the command history to the specified file.

6. Scripting commands

6.1. run

```
run(file)
```
Runs an ARIES script allowing for all IPython magics. The whole path has to be provided. The commands should be written in the specified file.

6.2. waitabs

```
waitabs(endTime, verbose = "no", verbperiod = 120)
```
Waits until the absolute time specified in the command line is reached and blocks the command flow. This command can be aborted with a CTRL-C.

endTime is an ephemeris structure which contains the year, month, day and time of the day. All commands return the start and end time in this format. In case of need it is possible to creat an ephemeris datetime structure by issuing, for example:

```
ARIES> dt = ephem.dateTime(year, month, day, hour, minute, second, millisecond)
```

and use variable dt inside waitabs

"verbose" is a flag that specifies if you want this command to print periodic information on the status of the antenna while the shell is blocked. By default it is "no". "verbperiod" is the period, in seconds, with which the status information is printed.

6.3. `waitrel`

```
waitrel(ss = 120, verbose = "no", verbperiod = 120)
```

Waits for ss seconds in the command flow. This command may be aborted with a CTRL-C. “ss” is the number of seconds to wait. By default it is 120 seconds. “verbose” is a flag that specifies if you want this command to print periodic information on the status of the antenna while the shell is blocked. By default it is “no”. “verbperiod” is the period, in seconds, with which the status information is printed.

7. Frontends and backends

Frontend and backend configuration is done in an object oriented mode and may confuse observers not used to these programming philosophy. As a first step the observer should create the frontend or backend and then use its methods to configure it. Methods for each frontend are called by writting the frontend name followed by a dot and the name of the procedure. We describe below how this is acomplished. The frontend and backend configuration is not applied until a pattern command is sent to the antenna.

7.1. `frontends`

```
frontends(names=[])
```

Sets the frontend(s) to be used for the next scan. This command clears all backend assignments made earlier. Known frontends are: ’het2’, ’hetch’, ’het5’, ’het6’, ’het8’, ’het12p’, ’het22l’, ’het22u’, ’het45’, ’het87’ and ’optskw’. ’het’ stands for heterodyne receiver, and the number after the frequency in GHz. ’optskw’ stands for optical camera. At the time of this report no continuum frontend was available at the 40m radiotelescope.

If only one fronted is to be used:

```
frontends('name')
```

If more than one frontend is used:

```
frontends(['name1', 'name2', ...]).
```

The previous methods creates a heterodyne frontend and/or the optical frontend object if they are included in the frontend list. The object will allow to command and monitor the frontend. The methods used for heterodyne frontends are different to the optical ones and are described below.

7.2. `heterodynefrontend.attenuation`

```
attenuation(attenuationdb=0)
```

Sets the relative IF receiver attenuation in dBs. The final IF attenuation for the receiver is the sum of this value plus value ”ifAttenuationRCPSD” or ”ifAttenuationLCPSD”in the database. So it should usually be 0. This value is common for both polarizations. If there is a need to set different attenuations to both polarizations, the difference should be set in the database.

7.3. heterodynefrontend.line

```
line(name=None, frequency=None, sideband='USB', unit='GHz')
```

Sets the line name and frequency. If no frequency is provided, it is read from the line catalog(s)

- `name` Transition name
- `frequency` Sky frequency in the specified units
- `sideband` Receiver sideband ('LSB', 'USB')
- `unit` Frequency units. It may take the following values:'GHz' or 'MHz'

7.4. heterodynefrontend.connectFeed

```
connectFeed(feedNumber, mode, backendName, section)
```

Connect the feed to a given section of a backend. The method sets the used feed of the receiver and the backend section used. The feedNumber is the feed to connect. Count starts from 1. The `mode` may take two values: "one", connects one feed at a time, or "all" connects all feeds. The latter option is not implemented. In dual polarization receivers, feed 1 is RCP and 2 is LCP

7.5. heterodynefrontend.numberOfUsedFeeds

```
numberOfUsedFeeds()
```

Returns the number of used feeds

7.6. heterodynefrontend.checkAvailableFeed

```
checkAvailableFeed(feedNumber)
```

Returns 1 if the Feed is not connected and hence available.

7.7. heterodynefrontend.usedFeeds

```
usedFeeds()
```

Returns an array with used feeds

7.8. heterodynefrontend.resetFeedConnections

```
resetFeedConnections(feedNumber, mode='one')
```

Resets a feed connection. The feed is removed from the internal list. The feed number starts from 1. The mode may be "one" which resets the specified feed, or "all" which resets all feeds connected to this frontend.

7.9. heterodynefrontend.connections

```
connections()
```

Prints the current connections of this receiver.

7.10. heterodynefrontend.obsfrequency

```
obsfrequency()
```

Displays the current observing frequency for that receiver. The observing frequency is the LO frequency plus half the bandwidth of the IF if observed in the upper side band or minus half of the IF band if observing in the lower side band.

7.11. heterodynefrontend.backends

```
backends(backlist=[[polarization, backendName, section], [polarization,
backendName, section]])
```

Sets the backends to be connected to the selected feed frontend. Currently only dual feed frontends, with different polarizations are supported. It is necessary to specify the backend section to which the selected feed frontend is connected. For the time being only one connection between one section and one feed is allowed, regardless of the type of backend (continuum or spectral). RCP is usually feed 1 and LCP feed 2.

- Allowed polarization values are: 'rcp', 'lcp', 'h', 'v'.
- Allowed backend names are: 'oay14', 'pbe', 'ffts', 'vlbi', pebble', 'holofft'
- The secion number indicates the input connector to which the receiver is connected to. For example, the 'pbe' has 8 sections, while 'oay14' has 1 section. Spectral backend 'ffts' has got 8 sections.

If this command is executed several times with different backends, only the last one will be effective.

7.12. opticalfrontend.setfocus

```
setfocus(position, mode='abs')
```

Sets the focuser position in steps and controls if the steps are absolute or manual. Central position is 3500. Currently the focus is at position = 6700 at 9 C. Larger values separate the CCD from the lens. Valid values are in the interval [1:7000]. Valid modes: 'abs or 'rel'.

7.13. opticalfrontend.focus

```
focus()
```

Gets the position of the focuser in steps. Valid interval is [1,7000]. Larger values separate the CCD from the lens.

7.14. opticalfrontend.temperature

```
temperature()
```

Gets the temperature from the PT100 in the subreflector cabin. There may be a small difference between the environment temperature of the focuser and the temperture probe one, since the focuser is covered with a plastic sleeve. Temperature units are degs C.

7.15. opticalfrontend.backends

```
backends('name')
```

Only backend which is allowed to be connected is 'ccd'.

7.16. opticalBackend

```
opticalBackend('name')
```

Creates an optical backend. The observer **should not** execute this command since the creation of all available backends is done by the ARIES shell. The only available optical backend is the Mintron camera plus the video grabber PCI card, which is coded as 'ccd'.

7.17. knownFrontend

```
knownFrontend('frontName')
```

Returns true if the frontend name is known, and false in any other case.

7.18. continuumBackend

```
continuumBackend('name')
```

Creates a continuum backend. The observer **should not** execute this command since the creation of all available backends is done by the ARIES shell. The setup of this class is not applied immediately to the backend. The data are stored in the scan and commanded when the next scan is loaded. Available continuum backends are: "oay14" and "pbe". Continuum backends may have several sections. A section is an input which can be connected to a frontend. For example "pbe" backend has got 8 sections while "oay14" has only got 1.

7.19. continuumBackend.resetSections

```
resetSections(self, connections=0)
```

Resets the specified number of connections to this backend, starting from the first one. Connections is the number of connections to be reset

7.20. continuumBackend.connectedSections

```
connectedSections()
```

Displays the number of sections connected and hence in use.

7.21. continuumBackend.setIntegrationTime

```
setIntegrationTime(dumpTime=1)
```

Sets the integration time. By default the integration time is 1 second. Units are seconds.

7.22. `continuumBackend.setGain`

```
setGain(gain=1)
```

Sets the gain. By default the gain is 1. This method does not make sense on backends (like ”oay14”) where the gain is not adjustable.

7.23. `spectralBackend`

```
spectralBackend('name')
```

Creates an spectral backend object that allows to command and monitor it with the methods specified later. The observer **should not** execute this command since the creation of all available backends is done by the ARIES shell. The setup of this class is not applied immediately to the backend. The data are stored in the scan and commanded when the next scan is loaded. Available spectral backends are: ”pebbles”, ”holofft” and ”vsib”.

7.24. `spectralBackend.setIntegrationTime`

```
setIntegrationTime(dumpTime=1)
```

Sets the integration time. By default the integration time is 1 second. Units are seconds.

7.25. `spectralBackend.setSetup`

```
setSetup(bwidth, nchan, centralFreq = 600)
```

Sets the spectral setup for the backend: backend bandwidth in MHz, number of channels in MHz and frequency where the centre is placed in the IF.

7.26. `knownBackend`

```
knownBackend('backendName')
```

Returns true if the backend name is known, and false in any other case.

8. Catalogs: sources and lines

8.1. `tolerance`

```
tolerance(radius=8.0)
```

Sets the target acquiring accuracy in arcsec. Telescope positions outside this limits are taken into account for tracking. Some scans may be aborted if the tracking error is higher than the tolerance. Tolerance should be 10 % of the HPBW. This command is called by `frontends` when the method is called.

8.2. sourcecats

```
sourcecats(catalogs=['user.cat'], mode = 'keep')
```

Defines the source catalog(s) to be used when loading a source. Do not forget to use the brackets and the quotation marks to surround the name of the catalog. The shell uses system catalogs which cannot be removed. However your catalogs will always have priority. If you have previously defined some catalogs you may drop them or keep them. This behaviour is controlled by parameter "mode", which accepts "keep" or "new".

You can define several catalogs at a time by creating python lists. For example:

```
sourcecats(['name1', 'name2', ...])
```

It may be necessary you use the whole path.

The typical format for a source catalog is as follows:

```
Orion|BN-KL     EQ             2000.0  05:35:14.16     -05:22:21.5     LSR      8.0    ! Comments
SgrB2|B2M       EQ             2000.0  17:47:20.41     -28:23:07.25    LSR     60.0    ! Comments
Intelsat901     INTELSAT       341.9969,-0.0004,-0.000242,0.01177,0.0009,-0.0369,0.0006,-0.0187,-0.0029,-0.03
Hispasat1c      NORAD          1,26071U,00007A,09127.72116993,-.00000219,00000-0,10000-3,0,2710  2,26071,0.06
Hispasat606     HISPASAT       /home/almamgr/aries21/Catalogs/hispasat_10_2008.dat  line2
Hispasat_1C     HISPASAT       /home/almamgr/aries21/Catalogs/H1C_ARIES.pnt    line2
Hispasat_1D     HISPASAT       /home/almamgr/aries21/Catalogs/H1D_ARIES.pnt    line2
IS1002          HISPASAT       /home/almamgr/aries21/Catalogs/IS1002.DAT       line2
Stow45          HO             45:00:00        90:00:00                                ! Comments
hor1            HO             176:00:00       43:00:00        LSR     0.0             ! Comments
hor2            HO             135:00:00       50:00:00                                ! Comments
horUP           HO             45:00:00        89:00:00                                !
herschel        EQ             2000.0  06:28:33.43     21:55:12.3      LSR     0.0     ! For 12:15 25/1/2010
voyager1        EQ             2000.0  17:09:10.03     11:55:46.3      LSR     0.0     ! For 12:15 25/1/2010
```

Sources in equatorial coordinates require the epoch, right ascention and declination plus the LSR velocity. Comments can be placed after a "!". It is possible to insert sources in horizontal coordinates (azimuth and elevation), coordinates in a file which contains time, azimuth and elevation. For this latter case one should use the keyword HISPASAT. Ephemeris for satellites in INTELSAT and NORAD systems may be inserted using two fields. Each field may contain several parameters separated by comas. In this latter case it is best not to include these ephemeris and allow the program fetch the coordinates in Internet since it is a faster and more reliable method.

8.3. linecats

```
linecats(catalogs=['user.lin'], mode = 'keep')
```

Defines line catalog(s) to be used when setting up the observing frequency. By default previous catalogs are kept. It is possible to remove the previous ones except the system catalog. This behaviour is controlled with the mode parameter, , which accepts "keep" or "new".

You can define several catalogs at a time by creating python lists. For example:

```
sourcecats(['name1', 'name2', ...])
```

It may be necessary you use the whole path.

The format for a line catalog is, for example, as follows:

```
! Line          Frequency       Unit    Band
! Pablo de Vicente
!
! C receiver
VLBI5            4908.000       MHz     USB
TEST5a           4728.000       MHz     USB
TEST5b           4611.000       MHz     USB
VLBI6            6660.000       MHz     USB
```

8.4. source

source(name='', x=('<coord>', '<unit>'), y=('<coord>', '<unit>'), systemS='EQ', epoch=2000.0, velocity=None, frame='LSR')

Set target source coordinates, the system (RA/Dec or Az/El), the epoch, its velocity and the reference frame. If no name is given the source infromation is retrieved from the catalog. If there is more than one entry in the available catalogs the system will offer the user to choose one.

- coord Coordinate definition e.g. '12:34:56.789' or 12.345678
- unit Angle unit ('degs', 'hours', 'rads', 'hms', 'dms')
- system Coordinate system ('EQ'(uatorial), 'HO'(rizontal))
- velocity Radial velocity in km/s
- frame Radial velocity frame ('LSR')

This command is usually executed filling only the name of the source. The system will look for a source in the supplied catalog files. If it does not find it, it will look in the NORAD database for a satellite. If it finds there it will load the ephemeris from Internet. The latter case requires Internet connection to work correctly.

This command displays the current position of the source and its distance to the sun. See an example below.

Below another example in which the satellite was found in the NORAD database:

9. Patterns

9.1. reference()

reference(x=-1000.0, y=0.0, time=0.0, on2off=1, unit='arcsec', mode='rel', system='EQ', epoch=2000.0)

Defines a reference position for on-off patterns and for the calibration.

- x Offset in longitude in the specified units.

- `y` Offset in latitude in the specified units.
- `time` Integration time when used for OFF positions
- `on2off` Number of on's per 1 off
- `unit` units for x and y. It can be 'arcsec', 'arcmin', 'hms', 'dms', 'deg' or 'rad'
- `mode` It can be 'rel' (relative) or 'abs' (absolute).
- `system` System used for the longitude and latitude.
- `epoch` Epoch for the chosen system if it is EQ. Disregarded if it is HO.

9.2. `calibrate()`

```
calibrate(mode='noisediode', timePerPhase=1)
```

Sets up a calibration scan on the source and on a reference position. The calibration scan is composed of several subscans dependent on the calibration mode whose duration is given by parameter `timePerPhase`.

- `mode` Can be "hot", "cold' or "noisediode". Only the 3mm receiver has a hot and cold load, all the other receivers use a noise diode.
  - A 'noisedeiode' scan is composed of 5 subscans: Two subscans on the reference position, one with diode on and the other with the diode on, and two subscans on the source, with the diode on, and with it off. The last subscan is to get the zero of the backend. The latter is acomplished by disconnecting the input and attenuating 20 dB.
  - A 'hot' scan is composed of 4 subscans: the antenna in the reference position, hot load in front of the feeds, and the antenna on the reference position again. The last scan is used to measure the zero of the backend.
  - A 'cold' scan is composed of 5 subscans: the antenna in the reference position, hot load in front of the feeds, cold load in front of the feeds and the antenna on the reference position again. The last subscan is used to obtain the zero of the backend.
- `timePerPhase`. Time in each phase. If the backend integration time is less than this one, it is possible to obtain several measurements in the same phase. It may be necessary to give a time for the phase slightly larger than the backend integration time.

9.3. `setTrajectoryParams()`

```
setTrajectoryParams(northCrossing=True, useHorizontalTables=False, timePerLine
maxTrackingErrors=10)
```

It allows to change the usage of main tracking tables at the ACU. This command should not be used by the observer except in rare ocasions.

- `northCrossing`. If True the antenna will cross the north if it is the shortest way to reach its target position.
- `horizontalTables`. If True the antenna will use Az, El tables. Default is no, and in this case equatorial tables shall be used.
- `timePerLineMs`. Time spacing in ms if using horizontal tables. This is only appliable for main tracking tables.
- `maxTrackingErrors`. Maximum number of allowed tracking errors before aborting tracking. Each tracking error is checked every 250 ms.

9.4. `trajectoryParams()`

```
trajectoryParams()
```

Returns the trajectory params currently used. These parameters are set up with method `setTrajectoryParam`

9.5. `on`

```
on(intTime=30)
```

Tracks a source for the given integration time in seconds. If the offsets are different than 0, they will be taken into account and this procedure will warn the observer. If after this method, the observer uses the same backend for pointing, be aware that the backend will use this integration time, unless the observer specifies a differente one.

9.6. `vlbi`

```
vlbi()
```

Tracks the specified source until it sets or a hardware limit is reached. This procedure will cancel the previous observation before loading this one. Offsets are taken into account and if they are not zero this procedure will warn the observer.

9.7. `onoff`

```
onoff(intTime=30)
```

The antenna tracks on the source for the specified time and then moves to the off position and stays there for the integration time. The off position is the one specified using command `reference()`. The integration time is in seconds.

9.8. `point`

```
point(length=512.0, unit='arcsec', timePerArm=60.0, mode='otf',
pointsPerArm=10, direction='x', zigzag=0)
```

Sets up a cross pointing observation with given arm length in the specified units and time in seconds per arm. The default mode is OTF. Alternatively, 'ras'(ter) mode can be specified. In

that case the 'points' parameter specifies the number of positions per arm and the integration applies to each position. zigzag = 0 is a simple cross, zigzag = 1 is a double cross. The backend integration time has to be set previously.

9.9. otf

`otf(xlen=350.0, ylen=350.0, step=70.0, timeperStroke=50, direction='x', zigzag=0, unit='arcsec', system='HO', epoch=2000.0)`
Sets up a rectangular On-The-Fly pattern given by sizes and steps in both directions. On-The-Fly means that the telescope moves continuously, drifting around the source from the start offset to the final one. After one drift is completed the antenna shifts a small amount in the perpendicular direction of the drift and makes a new one. The shift is given by paramter step. If `zigzag=1`, the OTF map is done bi-directionally, reducing the telescope overhead. The map is started at the most negative offset in X and Y and it is completed moving towards higher values.

- `xlen` Length of stroke in X axis direction in the specified units (by default arcsecs)
- `ylen` Length of stroke in Y axis direction in the specified units (by default arcsecs)
- `nstrokes` Angular separation between strokes in the specified units (by default arcsecs)
- `timeperStroke` time needed to complete a stroke (seconds).
- `direction` Direction of the stroke. Possibilities available: '-x', 'x', '-y', 'y'
- `system` X/Y direction may be in a horizontal coord system or in an equatorial one.
- `epoch` If the system is equatorial, the epoch determines it.

9.10. holography

`holography(npoints=180, step=10, timeperpoint= 0.25, subscanspercal=1, pointscenter= 10, timeStart=60, direction='x', zigzag=1, unit='arcsec', system='HO', epoch=2000.0)`
It performs an holography square on the fly map around the satellite position. The pattern begins staying on source for a given time that will allow the backend to setup correctly. Then it will make horizontal or vertical drifts alternating them with onsource tracking for some seconds. Parameters are used to configure the map size, the number of drifts per onsource tracking and time in each phase. The map is started at the most negative offset in X and Y and it is completed moving towards higher values.

- `npoints`. Number of integration points. Beware that being an OTF the drift length is $(npoints - 1) \cdot step$
- `step`. Distance between points in the specified units (arcsecs)
- `timeperpoint`. Integration time per point in seconds.

- `subscanspercal`. Number of subscans (drifts) per subscan in the center of the satellite
- `pointscenter`. Number of integrations at the center (on the satellite).
- `direction`. Direction of each drift. It may be 'x' for horizontal drifts or 'y' for vertical ones.
- `zigzag`. If 1 the antenna performs next subscan where the previous finished and in the opposite direction, but shifted an offset (one step) in the perpendicular axis.
- `unit`. Units for the step length
- `system`. Reference system for the map.
- `epoch`. In case the map is in EQ coordinates.

This command does not configure the integration time for the backend. The observer should do that previously.

9.11. `raster`

```
raster()
```

9.12. `skydip`

```
skydip(az='current', startel=5.0, endel=88, points=20, timePerPoint=1.0)
```

Sets up a skydip scan. By default the scan is performed at the current azimuth. Optionally, an azimuth value can be given in degrees. Defaults to an initial elevation of 5 degs and a final elevation of 80 degs, so that we avoid entering the prelimit switch zone.

- `az`. Azimuth at which we perform the skydip. [degs]
- `startel`. Elevation at which the skydip starts [degs.
- `endel`. Elevation at which the skydip finishes [degs].
- `points`. Number of points where the antenna stops to integrate.
- `timePerPoint`. Integration time per point.

9.13. `focus`

```
focus(startPos=-12, endPos=12, integrationTime=1, axis='z', points=2)
```

Set up a focus scan using N points, starting at startPos and ending at endPos. The scan lasts for the time given in integrationTime and the movement is done along the axis specified. If the number of points is 2, the subreflector moves from one position to the other without stopping. This command does not allow to move more than one axis at a time, although the ACU allows it. This scan is done while tracking a source on the sky. Unist for start position and end position are mm fi the axis is X, Y or Z and arcsecs if it is a tilt.

9.14. cycles

```
cyclesc(cyc=0)
```

Sets the number of repetitions of the pattern. Only implemented for onoff at the time of this report.

10. Others

10.1. tsys

```
tsys(febe)
```

Displays the computed Tsys from the last calibration scan for the specified frontend-continuum backend combination, for example: ”HET22L-PBE”. If the receiver is a dual pol the system temperature for both pols will be displayed.

10.2. trec

```
trec(frontend, backend)
```

Gets the Trec for the specified frontend measured with the specified backend. For example: ”HET87,PBE”.

10.3. calresults

```
calresults(flux)
```

Displays system temperature and the antenna aperture efficiency from the latest calibration scan. Eficciency is computed from the provided flux in Jy, and assuming a point like source.

10.4. wx

```
wx()
```

Prints information on the local weather

10.5. opacity()

```
opacity()
```

Displays the atmospheric opacity and precipitable water at the frequency of the selected frontend.

10.6. pointClass

```
pointClass(startWindow, endWindow)
```

Returns the pointing corrections for the last pointing scan in arcsecs. Both parameters allow to define a window for fitting a polynomial baseline avoiding the detected signal. The start and end should be in arcsecs.

11. Examples of use cases

11.1. Single dish pointing

This is an example of pointing using a dual polarization receiver connected to sections 1 and 3 of continuum backend PBE.

```
setTrajectoryParams(True, False, 60000, 6)
project_id("N-00.C-0010-2011", True)
sourcecats(['/home/almamgr/aries21/Catalogs/user.cat', '/home/almamgr/aries21/Catalogs/h
frontends("het22u")
#het22u.line("MIDNH3")
het22u.line("NH334")
het22u.backends([["rcp","pbe",1],["lcp","pbe",3]])
pbe.setIntegrationTime(1.0)
startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
waitabs(endTime)
```

11.2. On off

Example of onoff being 30 seconds in the on position and 30 seconds in the off position. The backend used is the FFT.

```
frontends("het22u")
het22u.line("NH334")
het22u.backends([["rcp","ffts",3],["lcp","ffts",4]])
ffts.setIntegrationTime(5)
ffts.setSetup(500,16384)
(az,el) = source("CEPA")
reference(-600,0,8,1,'arcsec', 'abs', 'EQ')
startTime, endTime = onoff(30)
waitabs(endTime)
```

11.3. Calibration

Calibration is done using the noise diode for 7 seconds per subscan and backend PBE.

```
setTrajectoryParams(True, False, 60000, 6)
project_id("N-00.C-0010-2011", True)
sourcecats(['/home/almamgr/aries21/Catalogs/user.cat', '/home/almamgr/aries21/Catalogs/h
frontends("het22u")
#het22u.line("MIDNH3")
het22u.line("NH334")
het22u.backends([["rcp","pbe",1],["lcp","pbe",3]])
pbe.setIntegrationTime(1.0)
startTime, endTime = calibrate("noisediode", 7)
waitabs(endTime)
```

11.4. Holography

The following macro goes to primary focus, blocks the subreflector, does pointing and corrects it, and enters into a loop in which a pointing, holography map and a movement to extend the grease of the gearings are executed.

```
sltp = fmode("primary")
waitrel(sltp)
project_id("T-00.E-0013-2010", True)
sourcecats(['/home/almamgr/aries21/Catalogs/user.cat'])
setTrajectoryParams(True, True, 40000, 10)
#
# Go to the satellite
#
az,el = source("INTELSAT 10-02")
# Usada por Pepe. Requiere sesion de punteria unas horas antes
#az,el = source("IS1002")
slt = gotoHor(az,el)
waitrel(slt)
frontends('het12p')
het12p.line('HOLOIS1002')
het12p.backends([['rcp','oay14',1]])
oay14.setIntegrationTime(1)
#
# Correct subref offsets and block the subreflector
#
fcorr(1.5,'x')
waitrel(10)
fcorr(-7.5,'y')
waitrel(20)
fcorr(-6,'z')
waitrel(20)
fcorr(229,'xtilt')
waitrel(25)
fcorr(-27,'ytilt')
waitrel(15)
blockSubreflector()
waitrel(5)
#
# Two pointings. First correct elevation only. The correct both
# A better solution would be: if abs(pel) > 5*abs(paz) correct pel
#
startTime, endTime = point(1200,'arcsec',60,'otf',10,'no','-x',1)
waitabs(endTime)
paz,pel = pointClass(-500,500)
# Usado por Pablo.
pcorr(0,pel)
cyclemax = 6
ncycles = 0

while ncycles < cyclemax:
        #
        # Pointing
```

```
#
frontends('het12p')
het12p.line('HOLOIS1002')
het12p.backends([['rcp','oay14',1]])
oay14.setIntegrationTime(1)
startTime, endTime = point(1200,'arcsec',60,'otf',10,'no','-x',1)
waitabs(endTime)
waitrel(5)
paz,pel = pointClass(-400,400)
pcorr(paz,pel,'rel')
waitrel(5)
#
# Holo cycle
#
frontends('het12p')
het12p.line('HOLOIS1002')
het12p.backends([['rcp','holofft',1]])
holofft.setIntegrationTime(0.5)
try:
        startTime, endTime = holography(128, 100, 0.5, 1, 10, 60, 'x', 0)
        waitabs(endTime)
        waitrel(10)
except Exception, e:
        waitrel(30)
        startTime, endTime = holography(128, 100, 0.5, 1, 10, 60, 'x', 0)
        waitabs(endTime)
        waitrel(10)

#
# Move the grease
#
cancel()
slt1 = gotoHor(355,80)
waitrel(slt1)
waitrel(5)
az,el = source("INTELSAT 10-02")
slt2 = gotoHor(az,el)
waitrel(slt2)
#
ncycles = ncycles + 1
```

11.5. Automatic script for pointing

Example of a pointing session which does pointing on a number of sources and repeats the same loop after it is finished.

```
setTrajectoryParams(True, False, 60000, 6)
sourcecats(['/home/almamgr/aries21/Catalogs/user.cat', '/home/almamgr/aries21/Catalogs/h
frontends("het22u")
#het22u.line("MIDNH3")
het22u.line("NH334")
het22u.backends([["rcp","pbe",1],["lcp","pbe",3]])
```

```
pbe.setIntegrationTime(1.0)
reference(-600,0,8,1,'arcsec', 'abs', 'EQ')
#
cyclesMax = 80
cycles = 0
elMin = 7.0
while cycles < cyclesMax:
        lastsource="None"
        logLine = "******************  Loop Cycle/Loop Max Cycles: %d/%d   ***********
        logAries(logLine)
        #
        #
        (az,el) = source("J0319+415")
        if el > elMin:
                try:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        nowDT = mx.DateTime.now()
                        logAries(str(nowDT))
                        waitrel(8)
                        try:
                                ts = tsys("het22l-pbe")
                                if ts > 150 or ts < 20:
                                        startTime, endTime = calibrate("noisediode", 6)
                                        waitabs(endTime)
                                        waitrel(8)
                        except Exception, e:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)

                except Exception, e:
                        waitrel(10)
                #
                try:
                        startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                        nowDT = mx.DateTime.now()
                        logAries(str(nowDT))
                        waitabs(endTime)
                        waitrel(8)
                        pcorrections()
                        lastsource="J0319+415"
                except Exception, e:
                        waitrel(10)
        else:
                logLine = " "
                logAries(logLine)
                logLine = " !!!!   ---   Imposible source  J0319+415 --- !!!"
                logAries(logLine)
                logLine = " "
                logAries(logLine)
        #
        #
```

```
(az,el)=source("J0423-013")
if el > elMin:
        try:
                startTime, endTime = calibrate("noisediode", 6)
                waitabs(endTime)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitrel(8)
                try:
                        ts = tsys("het22l-pbe")
                        if ts > 150 or ts < 20:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        waitrel(8)
        except Exception, e:
                waitrel(10)
        #
        try:
                startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitabs(endTime)
                waitrel(8)
                pcorrections()
                lastsource="J0423-013"
        except Exception, e:
                waitrel(10)
else:
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   Imposible source  J0423-013 --- !!!"
        logAries(logLine)
        logLine = " "
        logAries(logLine)
(az,el) = source("J0437+296")
if el > elMin:
        try:
                startTime, endTime = calibrate("noisediode", 6)
                waitabs(endTime)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitrel(8)
                try:
                        ts = tsys("het22l-pbe")
                        if ts > 150 or ts < 20:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
```

```
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        waitrel(8)
                except Exception, e:
                        waitrel(10)
                #
                try:
                        startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                        nowDT = mx.DateTime.now()
                        logAries(str(nowDT))
                        waitabs(endTime)
                        waitrel(8)
                        pcorrections()
                        lastsource="J0437+296"
                except Exception, e:
                        waitrel(10)
        else:
                logLine = " "
                logAries(logLine)
                logLine = " !!!!   ---   Imposible source  J0437+296 --- !!!"
                logAries(logLine)
                logLine = " "
                logAries(logLine)
        (az,el) = source("J1230+123")
        if el > elMin:
                try:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        nowDT = mx.DateTime.now()
                        logAries(str(nowDT))
                        waitrel(8)
                        try:
                                ts = tsys("het22l-pbe")
                                if ts > 150 or ts < 20:
                                        startTime, endTime = calibrate("noisediode", 6)
                                        waitabs(endTime)
                                        waitrel(8)
                        except Exception, e:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
                        waitrel(10)
                #
                try:
                        startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                        nowDT = mx.DateTime.now()
                        logAries(str(nowDT))
                        waitabs(endTime)
                        waitrel(8)
                        pcorrections()
                        lastsource="J1230+123"
                except Exception, e:
```

```
                waitrel(10)
else:
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   Imposible source  J1230+123 --- !!!"
        logAries(logLine)
        logLine = " "
        logAries(logLine)
(az,el)=source("J1229+020")
if el > elMin:
        try:
                startTime, endTime = calibrate("noisediode", 6)
                waitabs(endTime)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitrel(8)
                try:
                        ts = tsys("het22l-pbe")
                        if ts > 150 or ts < 20:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        waitrel(8)
        except Exception, e:
                waitrel(10)
        #
        try:
                startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitabs(endTime)
                waitrel(8)
                pcorrections()
                lastsource="J1229+020"
        except Exception, e:
                waitrel(10)
else:
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   Imposible source  J1229+020 --- !!!"
        logAries(logLine)
        logLine = " "
        logAries(logLine)
(az,el)=source("J1337-129")
if el > elMin:
        try:
                startTime, endTime = calibrate("noisediode", 6)
                waitabs(endTime)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
```

```
                waitrel(8)
                try:
                        ts = tsys("het22l-pbe")
                        if ts > 150 or ts < 20:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        waitrel(8)
        except Exception, e:
                waitrel(10)
        #
        try:
                startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitabs(endTime)
                waitrel(8)
                pcorrections()
                lastsource=source("J1337-129")
        except Exception, e:
                waitrel(10)
else:
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   Imposible source  J1337-129  --- !!!"
        logAries(logLine)
        logLine = " "
        logAries(logLine)
(az,el)=source("dr21")
if el > elMin:
        try:
                startTime, endTime = calibrate("noisediode", 6)
                waitabs(endTime)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitrel(8)
                try:
                        ts = tsys("het22l-pbe")
                        if ts > 150 or ts < 20:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        waitrel(8)
        except Exception, e:
                waitrel(10)
        #
        try:
```

```
                startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitabs(endTime)
                waitrel(8)
                pcorrections()
                lastsource=source("dr21")
        except Exception, e:
                waitrel(10)
else:
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   Imposible source  dr21  --- !!!"
        logAries(logLine)
        logLine = " "
        logAries(logLine)
(az,el)=source("Venus")
if el > elMin:
        try:
                startTime, endTime = calibrate("noisediode", 6)
                waitabs(endTime)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitrel(8)
                try:
                        ts = tsys("het22l-pbe")
                        if ts > 150 or ts < 20:
                                startTime, endTime = calibrate("noisediode", 6)
                                waitabs(endTime)
                                waitrel(8)
                except Exception, e:
                        startTime, endTime = calibrate("noisediode", 6)
                        waitabs(endTime)
                        waitrel(8)
        except Exception, e:
                waitrel(10)
        #
        try:
                startTime, endTime = point(600,'arcsec',60,'otf',10,'no','-x',1)
                nowDT = mx.DateTime.now()
                logAries(str(nowDT))
                waitabs(endTime)
                waitrel(8)
                pcorrections()
                lastsource=source("Venus")
        except Exception, e:
                waitrel(10)
else:
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   Imposible source  Jupiter  --- !!!"
        logAries(logLine)
        logLine = " "
```

```
        logAries(logLine)
# ----------------------------------
cycles = cycles + 1
if lastsource == "None":
        logLine = " "
        logAries(logLine)
        logLine = " !!!!   ---   no pointing sources above 6 degs. Sleeping 120
        logAries(logLine)
        logLine = " "
        logAries(logLine)
        waitrel(120)
```

Highlights:

- `cyclesMax` is a variable defined by the user the first time it is used. The ARIES shell is a customized ipython shell and therefore has got all facilities from python. The user may define any variable that does not conflict the system or the shell ones. It is also possible to get the value of ARIES shell variables.
- `while` is a python command. As we said in the previous item, the ARIES shell is a python shell.
- `sleep` is command from the time python module. Its argument is in seconds. The shell freezes and blocks the input for that time interval.
- `for` is also a python command.
- `startTime, endTime = point(512,'arcsec',60,'otf')`. Any pattern command returns a python tuple with the start and end time. These time variables contain an array with hour, minute, second and day of year. These time variables can be used to program wait intervals for later commands.
- `waitabs()`. It blocks the input flow until the specified absolute time has arrived.
- `waitrel()`. It blocks the input flow until the specified relative time in seconds has arrived.