LBT PROJECT
2x8,4m TELESCOPE

Instrument - Telescope Control Document
<table>
<thead>
<tr>
<th><strong>Signature</strong></th>
<th><strong>Date</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepared</td>
<td>Jose Borelli</td>
</tr>
<tr>
<td>Approved</td>
<td></td>
</tr>
<tr>
<td>Reviewed</td>
<td></td>
</tr>
</tbody>
</table>
1. Revision History

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Changes</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>02-Mar-2007</td>
<td>First draft</td>
<td>Jose Borelli</td>
</tr>
<tr>
<td>b</td>
<td>15-Mar-2007</td>
<td>Second draft</td>
<td>Jose Borelli</td>
</tr>
</tbody>
</table>
2. Table Of Contents

1. Revision History .................................................................................................................. 3
2. Table Of Contents ................................................................................................................ 4
3. List Of Abbreviations ......................................................................................................... 6
4. About this document ............................................................................................................. 7
   4.1. Purpose ........................................................................................................................... 7
   4.2. Notes ................................................................................................................................ 7
5. Introduction .......................................................................................................................... 8
6. Functionality and Observation modes ............................................................................... 8
   6.1. Control functionality ....................................................................................................... 9
   6.1.1. Authorizing with the TCS ........................................................................................ 9
   6.1.2. Giving up control .................................................................................................... 10
   6.1.3. Status ....................................................................................................................... 10
   6.2. Basic functionality .......................................................................................................... 10
   6.2.1. Pointing .................................................................................................................... 10
   6.2.2. Tracking .................................................................................................................. 11
   6.2.3. Offset ...................................................................................................................... 11
   6.2.4. Focusing .................................................................................................................. 12
   6.2.5. Guiding .................................................................................................................... 12
   6.3. Alignment functionality ................................................................................................. 13
6.4. Adaptive Optic System ................................................................................................... 14
7. Command set at the IIF-TCS ............................................................................................ 16
   7.1. AOS_PresetAO ............................................................................................................ 27
   7.2. AOS_AcquireRefAO .................................................................................................... 30
   7.3. AOS_RefineAO ............................................................................................................ 32
   7.4. AOS_StartAO .............................................................................................................. 35
   7.5. AOS_OffsetXY ............................................................................................................ 37
   7.6. AOS_OffsetZ ............................................................................................................... 39
   7.7. AOS_CorrectModes ..................................................................................................... 41
   7.8. AOS_StopAO ................................................................................................................ 43
   7.9. AOS_PauseAO ............................................................................................................. 45
   7.10. AOS_ResumeAO ......................................................................................................... 47
   7.11. AOS_UserPanic .......................................................................................................... 49
   7.12. Authorize ..................................................................................................................... 51
   7.13. Deauthorize ................................................................................................................ 53
   7.15. GetParameter .............................................................................................................. 57
   7.16. LogEvent ..................................................................................................................... 58
   7.17. Move ............................................................................................................................ 60
   7.18. MoveFocus .................................................................................................................. 63
   7.19. MoveXY ....................................................................................................................... 65
   7.20. MoveXYZ ...................................................................................................................... 67
7.21.OffsetGuiding........................................................................................................69
7.22.OffsetPointing........................................................................................................71
7.23.PresetTelescope......................................................................................................73
7.24.Preset_guiding.......................................................................................................76
7.25.RotatePrimary.........................................................................................................78
7.26.RotateZ.....................................................................................................................80
7.27.Rot_Set_Rotator.....................................................................................................82
7.28.Rot_Force_Wrap.....................................................................................................84
7.29.Rot_Track................................................................................................................86
7.30.Rot_Hold..................................................................................................................88
7.31.Rot_Adjust_Position..............................................................................................90
7.32.Rot_TimeToIntercept.............................................................................................92
7.33.SendWavefront.......................................................................................................94
7.34.SetMultiParameter...............................................................................................96
7.35.SetParameter..........................................................................................................98
7.36.Standby..................................................................................................................100
7.37.Start_Guiding.........................................................................................................102
7.38.StepFocus................................................................................................................103
7.39.Stop_Guiding..........................................................................................................105
7.40.Tip/Tilt....................................................................................................................106
7.41.Update_Guidestar.................................................................................................108
8.Process flow. Usage...................................................................................................110
8.1.Including IIF into the project..................................................................................110
8.2.Creating IIF and Authorizing with TCS.................................................................110
8.3.Command request process ....................................................................................112
8.4.Command result evaluation......................................................................................113
8.5.De-authorizing the instruments. Destroying IIF instance.......................................115
8.6.Full example............................................................................................................116
9.References..................................................................................................................120
Appendix A : Global definitions ..................................................................................121
3. List Of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGW</td>
<td>Auto Guiding and Wavefront sensing</td>
</tr>
<tr>
<td>ALT</td>
<td>Altitude</td>
</tr>
<tr>
<td>AOS</td>
<td>Adaptive Optic System</td>
</tr>
<tr>
<td>AOS-Sup</td>
<td>AOS supervisor</td>
</tr>
<tr>
<td>AZ</td>
<td>Azimuth</td>
</tr>
<tr>
<td>CSQ</td>
<td>Instrument Interface Server</td>
</tr>
<tr>
<td>DEC</td>
<td>Declination</td>
</tr>
<tr>
<td>GCS</td>
<td>Guiding Control System</td>
</tr>
<tr>
<td>ICS</td>
<td>Instrument Control Software</td>
</tr>
<tr>
<td>IIF</td>
<td>Instrument Interface</td>
</tr>
<tr>
<td>IRA</td>
<td>Initial Rotation Angle</td>
</tr>
<tr>
<td>LBT</td>
<td>Large Binocular Telescope</td>
</tr>
<tr>
<td>LBTI</td>
<td>Large Binocular Telescope Interferometer</td>
</tr>
<tr>
<td>LBTO</td>
<td>LBT Organization</td>
</tr>
<tr>
<td>LINC</td>
<td>LBT Interferometric beam Combiner</td>
</tr>
<tr>
<td>LN</td>
<td>Linc Nirvana</td>
</tr>
<tr>
<td>LUCIFER</td>
<td>LBT NIR spectroscopic Utility with Camera and Integral-Field Unit for Extragalactic Research</td>
</tr>
<tr>
<td>MODS</td>
<td>Multiple Object double spectrograph</td>
</tr>
<tr>
<td>NIRVANA</td>
<td>Near-Infrared / Visible Adaptive iNterferometer for Astronomy</td>
</tr>
<tr>
<td>OPE</td>
<td>OPtical Element</td>
</tr>
<tr>
<td>PCS</td>
<td>Pointing Control System</td>
</tr>
<tr>
<td>PEPSI</td>
<td>Postdam Echelle Polarimetric and spectroscopy instrument</td>
</tr>
<tr>
<td>RA</td>
<td>Right Ascension</td>
</tr>
<tr>
<td>SFP</td>
<td>Standard Focal Plane</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Defined</td>
</tr>
<tr>
<td>TCS</td>
<td>Telescope Control System</td>
</tr>
<tr>
<td>WV</td>
<td>Wave Front</td>
</tr>
</tbody>
</table>
4. **About this document**

The Instrument Interface (IIF) is the software interface which allows the instrument software to communicate with the Telescope Control System (TCS), in order to issue commands which provide control, allow a transparent communication with the different subsystem and acquire status information. This document specifies the Instrument-Telescope control software, the different structures and the command set.

4.1. **Purpose**

The purpose of this document is to serve as a reference manual for software developers that want to use the Instrument Interface library to communicate with the TCS, describing the set of commands that LBT will provide to all the instruments.

4.2. **Notes**

The document was divided into different sections. Each section has a flat structure and an unnumbered sequence of sub sections. The aim is to present each command or new topic in a confined space so that it can be quickly grasped. The list of commands presented in section 7 are in alphabetical order.
5. Introduction

The Instrument Interface is distributed to the LBT instrument software teams as a set of libraries. These libraries must be included into the instrument control software projects in order to issue commands. The instrument is required to identify itself to the TCS. This is done by specifying a predefined, unique ID value during the instantiation of an IIF object. This globally unique identifier is composed of an instrument ID (1-24 characters) plus a string, which determines the focal station and the telescope side to be controlled by the commanding instrument.

When an instrument calls an IIF command, this action triggers processing inside the TCS. The instrument then has the option of halting the execution of its program thread until the requested TCS-internal processing has finished or, alternatively, the instruments' control code can proceed with its own program thread and decide to periodically query the status of the IIF command.

6. Functionality and Observation modes

All the astronomical instruments require similar basic functionalities from the TCS, which are shown in figure 1. Within this chapter we describe the functionalities in a general form.
6.1. Control functionality

6.1.1. Authorizing with the TCS

Under normal operating conditions, only a single instrument can have full control of a telescope side; this control is obtained through the authorization command. When an instrument is authorized for a telescope side, it is the only instrument which may successfully issue a majority of the commands.
6.1.2. Giving up control

When the instrument is not using anymore one or both sides of the telescope, the deauthorize command must be issued, in order to yield all control of the corresponding side to the TCS. Note too, that multiple authorizations/deauthorizations from the same instrument are allowed by the CSQ, maintaining a count. A final deauthorize happens when the count goes to zero.

6.1.3. Status

The instruments are able to ask for status of the TCS parameters and the telescope situation. A status request could be on remaining ranges of optical elements (primary, secondary, etc.), actual coordinates (the telescope is pointing on), enclosure (open or closed), ambient temperature, etc. All this information is not needed in real-time and can have some delay in reporting as it is changing slowly. Such information can be requested through the TCS-IIF. Also, the instrument is able to set a parameter or a group of parameters in the data dictionary, in order to give status information about the instrument to the TCS.

6.2. Basic functionality

6.2.1. Pointing

Pointing defines the position the telescope looks to. Primarily the telescope mounting is moved to this position, secondary, and especially with a binocular telescope, the mirrors have to be adjusted to select the same target.

In order to begin an observation cycle by slewing the telescope to the target position, IIF has the \texttt{presetTelescope()} command. This IIF command has a large number of arguments, which are described in detail in the section 7.23.

Pointing sequence

1. The instrument sends the telescope the preset information for each side (Left, Right or both) that is to be preset. Presetting both sides only makes sense for dual-side instruments being controlled through
a single interface.

2. The telescope checks if the instrument is authorized.

3. The telescope checks the preset parameters for sanity.

4. If a single side is being preset, the preset will fail if the other side is in Preset state, and the requested preset would force it out its Preset state.

5. The Telescope performs the Guide Star Acquisition, picking a guide star from the set of passed guide stars in the order they were passed.

6. If steps 2 through 4 succeed, the specified side of the telescope goes into the Preset state. In consequence, the telescope is on a new position, tracking, guiding and controlling active optic as well. Otherwise, the preset operation fails, and the telescope side(s) being preset enters the idle state.

7. The Telescope informs the Instrument about success or failure of the preset operation.

### 6.2.2. Tracking

Tracking means to follow the science target in open loop, without guiding. Is is possible to follow a non sidereal target, and in some occasions it is possible to switch-off the tracking.

### 6.2.3. Offset

During an observation it is often needed to offset the telescope, this means to move a small distance in RA and DEC. This might happen in two different ways, Non-guided offset or Guided offset. In the first one, the telescope moves a small delta RA and DEC but the guiding loop opens because the distance is too large or the accuracy is not so relevant. In the second case, the telescope moves a small delta RA and DEC, but the guiding loop is kept closed. This increases the accuracy. The guiding system has to take care of the jump the guide star will experience and has to reference the position of the guide star to its initial position. IIF provides two commands for this purpose (more details in sections 8.7 and 8.8):
6.2.4. Focusing

The optomechanics of telescope and instrument are normally not that perfect to be in focus without adjusting some parts within the optical path. Also, as the structure of the telescope and instrument changes with different temperature the focus has to be adjusted from time to time. Focusing needs information from the instrument. With LBT in adaptive mode this will be handled by the AOS.

Positioning the Focus
1. The instrument sends the telescope a request to move the respective OPE to a new absolute focus position, specified by:
   (a) the new position (in millimeters) of the optical element in the z-axis.
   (b) the OPE this command applies for.
2. The telescope checks if the positioning information requested in step 1 is within range.
3. In case of success of step 2, the telescope positions the focus to the desired position.
4. The telescope confirms success or failure of the operation to the Instrument.

Incrementing/Decrementing the Focus
1. The instrument sends to the telescope a request to change the respective primary mirror's focus position relative to its current position, specified by:
   (a) the difference $\Delta$ (in millimeters) to the current position of the optical element in z-axis.
   (b) the OPE this command applies for.
2. The telescope checks if the positioning information requested in step 1 is within range.
3. In case of success of step 2, the telescope adapts the focus by the requested difference $\Delta$.
4. The telescope confirms success or failure of the operation to the instrument.

More information about these commands in section 7.X

6.2.5. Guiding

The telescope provides a non perfect open loop tracking of the mounting. To optimize the tracking for errors due to friction, mechanical imperfection or flexure, a guiding system takes care.
In order for the guiding control system (GCS) to perform its duties,
delivering guiding and wavefront sensing information to the TCS, the GCS controls off-axis AGW units and their cameras for image acquisition.

### 6.3. Alignment functionality

The optical axis of the instrument and the telescope have to be aligned, as well as the right and left side in respect of each other. Such alignment finally happens in the combined focus with the help of stars. The telescope to instrument alignment requires adjustments of the optical elements (OPE). It is possible to move the primary (M1), the secondary (M2) and tertiary (M3) mirror in all degrees of freedom. IIF provides several commands to perform these duties, which are described in the next sections.

![Figure 6.2: Use cases to move the telescope for alignment / adjustments purpose.]
6.4. Adaptive Optic System

The Adaptive Optics Subsystem (AOS) is the subsystem of TCS providing all the functionalities needed for interaction between the LBT Adaptive Optics system and the rest of the telescope, including instruments.

AO System Operating Modes

When operating in support of an observation the AO System will provide four modes of operation:

- **FIX-AO. Fixed Mode Operation.** It is the seeing limited mode where the Adaptive Secondary mirror holds a fixed "flat" shape defined by a pre-calibrated vector of mirror commands. Depending on the particular kind of observation a specific "flat" vector may be selected.
- **TTM-AO. Tip-Tilt Mode Operation.** It is an AO mode with only tip-tilt correction performed by the secondary mirror.
- **ACE-AO. Auto Configured Adaptive Optics Operation.** It is the full AO corrected mode, with AO loop parameters automatically selected by the AO System based on reference source characteristics.
- **ICE-AO. Interactively Configured Adaptive Optics Operation.** It is a full
AO corrected mode where the observer is given the possibility to adjust AO loop parameters.

TCS-IIF provides a set of commands, that correspond exactly to the AOS commands, in order to handle this subsystem. More details in sections 2.17 to 2.27 (AOS_ prefix) and reference [3] as well.
7. Command set at the IIF-TCS

Prior to describe the different IIF commands we will describe some structures and classes the instruments will need. These classes and structures are distributed in several files. Nevertheless, including LBT/IIF.h will be enough for the instruments to use the libraries.

In order to understand how to include and to use them, please see section 8. The global definitions and constants showed in this document are defined in the file LBT/IIFGlobal.h.

Coordinates (LBT/Position.h)

Coord type

- double coord1
  Purpose: to define the first coordinate of the specified system in RA, ALT or Rho.
  Description: double value to represent the first coordinate of position.
  Unit: TBD
  Range or possible value: TBD
  Default value: 0

- double coord2
  Purpose: to define the second coordinate of the specified system in DEC, AZ or Theta.
  Description: double value to represent the second coordinate of position.
  Unit: TBD
  Range or possible value: TBD
  Default value: 0

- const char * system
  Purpose: to define the coordinate system of reference for the coordinates.
  Description: string value representing the coordinate system. It could be:
  - COORD_RADEC: right ascension, declination.
  - COORD_ALTAZ: Altitude, Azimuth.
  - COORD_FOCAL: Focal Plane.
  Unit: unitless
  Range or possible value: COORD_RADEC | COORD_ALTAZ | COORD_FOCAL
  Default value: COORD_RADEC

The Proper Motion (LBT/Position.h)

The proper motion of a star is the measurement of its change in position in the sky over time after improper motions are accounted for.

ProperMotion type
The apparent magnitude (LBT/Position.h)

The apparent magnitude \((m)\) of a star, planet or other celestial body is a measure of its apparent brightness as seen by an observer on Earth.

**Magnitude type**

- **double apparentMagnitude**
  
  **Purpose:** to define the apparent brightness of the star as seen by an observer on Earth.
  
  **Description:** double value representing the apparent magnitude.
  
  **Unit:** unitless
  
  **Range or possible value:** \([-27.0000, 40.0000]\) | APPARENTMAG_NONE
  
  **Default value:** APPARENTMAG_NONE

- **const char* filter**
  
  **Purpose:** to define the wavelength filter for the magnitude.
  
  **Description:** string representing the wavelength filter.
  
  **Unit:** unitless
  
  **Range or possible value:** "V" | "U" | "B" "R" | "TBD" | FILTERMAG_NONE
  
  **Default value:** FILTERMAG_NONE

- **double color**
  
  **Purpose:** to define the color index of the object.
  
  **Description:** double value representing the color index.
  
  **Unit:**
  
  **Range or possible value:** COLOR_NONE | [TBD]
  
  **Default value:** COLOR_NONE

- **const char* colorType**
  
  **Purpose:** to define the color type of the object.
  
  **Description:** string value representing the color type.
  
  **Unit:** unitless
  
  **Range or possible value:** "U-B" | "B-V" | COLORTYPE_NONE | TBD
  
  **Default value:** COLORTYPE_NONE
The Position class *(LBT/Position.h)*

Defining positions of observation target objects, guide stars and offsets, Position can exist in different coordinate systems, but all coordinates must be specified in radians.

The constructor of the class has the following parameters:

- **double coord1**
  
  **Purpose:** to define the first coordinate of the specified system in RA, ALT or Rho.
  
  **Description:** double value to represent the first coordinate of position.
  
  **Unit:** radian
  
  **Range or possible value:** [-1.570796327 , 6.283185307]
  
  **Default value:** 0

- **double coord2**
  
  **Purpose:** to define the second coordinate of the specified system in DEC, AZ or Theta.
  
  **Description:** double value to represent the second coordinate of position.
  
  **Unit:** radian
  
  **Range or possible value:** [-1.570796327 , 6.283185307]
  
  **Default value:** 0

- **string system**
  
  **Purpose:** to define the coordinate system of reference for the coordinates. Could be:
  
  - COORD_RADEC: right ascension, declination.
  - COORD_ALTAZ: Altitude, Azimuth.
  - COORD_FOCAL: Focal Plane.
  
  **Description:** string to represent the coordinate system of reference for the coordinates.
  
  **Unit:** unitless
  
  **Range or possible value:** COORD_RADEC | COORD_ALTAZ | COORD_FOCAL
  
  **Default value:** COORD_RADEC

- **string equinox**
  
  **Purpose:** to define the value of the equinox.
  
  **Description:** string value representing the equinox (only affects the RADEC system).
  
  **Unit:** year
  
  **Range or possible value:**
  
  **Default value:** “J2000.0”

- **string epoch**
  
  **Purpose:** to define the value used in conjunction with the proper motion to compute the precession of the target coordinates.
  
  **Description:** string that represents the epoch of the given coordinates.
  
  **Unit:** year
  
  **Range or possible value:** Valid date.
  
  **Default value:** “2000.0”

- **ProperMotionType *propmotion (optional)**
  
  **Purpose:** to specify the potential proper motion of the celestial object described by the position.
  
  **Description:** pointer to a ProperMotionType struct (see definitions above).
  
  **Unit:** See ProperMotionType definition above.
Range or possible value: see definition above.
Default value: NULL (do not use proper motion).

- **MagnitudeType** *magnitude (optional)*
  
  **Purpose:** to specify the apparent magnitude and filter of the celestial object.
  **Description:** pointer to a MagnitudeType struct (see definitions above).
  **Unit:** See description above
  **Range or possible value:** See description above
  **Default value:** NULL (do not use the magnitude).

- **unsigned int** wavelength (optional)
  
  **Purpose:** to compute the object's atmospheric refraction correction.
  **Description:** integer value specifying the effective wavelength of the target.
  **Unit:** nanometer.
  **Range or possible value:** [300, 20000]
  **Default value:** 0 (do not use the wavelength).

**Position public member functions**

```c
Position (double coord1=0.0, double coord2=0.0,
const char *system=COORD_RADEC,
const char *equinox=\"B2000.0\",
const char *epoch=\"2000.0\",
ProperMotionType *propmotion=NULL,
MagnitudeType *mag=NULL,
unsigned int wavelength=0)
```

The class's parameterized Constructor.

```c
virtual ~Position ()
```

The class's Destructor.

```c
Position (const Position &)
```

The class's Constructor.

```c
bool & operator= (const Position &)
```

operators =, ==, !=

```c
double Getcoord1 () const
```

Getter method for the coord1 property of this class.

```c
double Getcoord2 () const
```

Getter method for the coord2 property of this class.

```c
Char* | string Getcoordsystem () const
```

Getter method for the coordinate system property of this class.

```c
Char* | string Getequinox () const
```

Getter method for the equinox property of this class.

```c
Char* | string Getepoch () const
```

Getter method for the epoch property of this class.
ProperMotionType *Getpropermotion ()
Getter method for the propermotion property of this class.

MagnitudeType *Getmagnitude ()
Getter method for the magnitude property of this class.

unsigned int GetWavelength () const
Getter method for the wavelength property of this class.

ProperMotionType getProperMotion () const
Getter method for the propermotion property of this class.

MagnitudeType getMagnitude () const
Getter method for the magnitude property of this class.

void Setcoord1 (double newVal)
Setter method for the coord1 property of this class.

void Setcoord2 (double newVal)
Setter method for the coord2 property of this class.

void Setcoordsystem (char *newVal)
Setter method for the coordsystem property of this class.

void Setequinox (char *newVal)
Setter method for the equinox property of this class.

void Setpropermotion (ProperMotionType *newVal)
Setter method for the propermotion property of this class.

void Setmagnitude (MagnitudeType *newVal)
Setter method for the magnitude property of this class.

void Setwavelength (unsigned int newVal)
Setter method for the wavelength property of this class.

bool hasMagnitude ()
Return true if the magnitude was specified. Otherwise false.

bool hasProperMotion ()
Return true if the propermotion was specified. Otherwise false.

bool hasWavelength ()
Return true if the wavelength was specified.
The Result class (LBT/Result.h)

This class defines the results that are returned to the Instrument actor when commands are sent to the Telescope (to the CSQ). Results are immediately returned by the respective IIF methods. This means that the operation inside the TCS initiated by the IIF command may not have finished by the time the IIF method call returns. The status of an initiated operation can be inquired using the command handle inside the Result instance, as we describe in section 8. However it is possible to halt the program thread’s execution until the TCS has finished the command processing (“busy wait”). To achieve this, the IIF user simply calls the block() method on the command handle that is associated with the requested command.

Result attributes

- **CSQHandle* commandHandle**
  
  **Purpose:** to define the command handle associated with the Result instance.
  
  **Description:** reference to a CSQHandle object.
  
  **Unit:** unitless.
  
  **Range or possible value:** -
  
  **Default value:** NULL.

- **int resCode**
  
  **Purpose:** to define the result code, to indicate the success of sending the command to the TCS.

  **Description:** integer value representing the result code.

  **Unit:** unitless.

  **Range or possible value:** RESCODE_OK | RESCODE_FAIL

  **Default value:** RESCODE_FAIL

- **string resString**
  
  **Purpose:** to define the result string to indicate the success of sending the command to the TCS.

  **Description:** string variable defining the result string.

  **Unit:** unitless.

  **Range or possible value:** RESSTRING_OK | RESSTRING_FAIL

  **Default value:** RESSTRING_FAIL

- **double ttc**
  
  **Purpose:** to define the expected time to complete the operation in the TCS.

  **Description:** double value to represents the “time to complete” attribute.

  **Unit:** [TBD]

  **Range or possible value:** [TBD]

  **Default value:** -1.0 | [TBD]
Result public member functions

Result (CSQHandle *handle=NULL, int resCode=RESCODE_FAIL,
       string resString=RESSTRING_FAIL, double ttc=-1.0)
The class's parametized Constructor.

*GetcommandHandle ()
Getter method for the commandHandle property of this class.

GetresultCode ()
Getter method for the resultCode property of this class.

GetresultString ()
Getter method for the resultString property of this class.

GettimeToComplete ()
Getter method for the ttc (time to complete) property of this class.
The Status class (LBT/StatusInfo.h)

Defines the requested status information for the processing request (specified by a handle), and (in case of completion of the command) the structure returned by the processing request.

StatusInfo public member functions

```c
string GetCommandResult();
Retrieves the (string) result of the CSQ command associated with the StatusInfo instance, which is STATE_NORESULT, in case the command handle is not assigned.

string GetCommandStatus();
Retrieves the status string of the CSQ command associated with the StatusInfo instance, or a string indicating that there is no associated command.

double GetEstimateTTC();
Retrieves the latest estimate for the time to completion of the CSQ command associated with the StatusInfo instance, or -1.0 indicating that the operation has failed.
```

StatusInfo definitions

STATE_RUNNING "RUNNING"
Definition of the string indicating that a command is still being executed by the CSQ.

STATE_CANCELED "CANCELED"
Definition of the string indicating that a CSQ command has been previously canceled.

STATE_SUCCESS "SUCCESS"
Definition of the string indicating that the CSQ has successfully finished processing the associated command.

STATE_FAILURE "FAILED"
Definition of the string indicating that an error has occurred during execution of the associated command by the CSQ.

STATE_WRONGHANDLE "NO ASSOCIATED COMMAND HANDLE"
Definition of the string indicating there is no associated command for the specified command handle.

STATE_NORESULT "NO COMMAND RESULT"
Definition of the string indicating there is no valid result string that can be returned as the command's result.
The MultiDDEntry class (LBT/MultiDDEntry.h)

This class is used to create a list of data dictionary entries that will be requested to the TCS. Also, can be used to set a block of entries into the data dictionary. The instrument control software (ICS) needs a MultiDDEntry object in order to use the commands SetMultiParameter and SetMultiParameter as well (described in section 7.14 and 7.34).

```
MultiDDEntry();
MultiDDEntry( queue<string> entries);
MultiDDEntry( queue<string> entries,
              queue<string> values);

virtual ~MultiDDEntry();
MultiDDEntry& operator= (MultiDDEntry &);
```

Copy-constructor and assignment-operator

```
void SetMultiEntries ( queue<string> entries);
void SetMultiValues ( queue<string> values);
void SetMultiEntries ( queue<string> entries,
                      queue<string> values);

void SetMultiEntries ( queue<string> entries);
void SetMultiValues ( queue<string> values);
void SetMultiEntries ( queue<string> entries,
                      queue<string> values);

void PushEntry(string entry);
void PushEntry(string entry, string value);
void PushValue(string value);

void PopEntry();
void PopValue();

void FrontEntry() const;
void FrontValue() const;

unsigned int size();
unsigned int sizeEntry();
unsigned int sizeValue();
```

Methods related to MultiDDEntry: 
- SetMultiEntries: Sets the entries or values directly.
- PushEntry and PushValue: Add new entries or values.
- PopEntry and PopValue: Remove the current entry or value.
- FrontEntry and FrontValue: Get the current entry or value.
- empty, emptyEntry, emptyValue: Check if the queue is empty.
- size, sizeEntry, sizeValue: Get the number of entries or values.
<table>
<thead>
<tr>
<th>void</th>
<th>Clear();</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear method, cleans the queues. The number of entries in the queues after Clear() is 0.</td>
<td></td>
</tr>
</tbody>
</table>
**Wavefront class** (LBT/Wavefront.h)

This class contains information about the wave front correction to account for the atmospheric aberration. The ICS uses this class in order to issue the `SendWavefront()` command. See more details in section 7.33

**Public Member Functions**

- `WaveFront (double coeffs[POLY_ORDER])`
  The class's parametized Constructor.

- `WaveFront ()`
  The class's Constructor.

- `virtual ~WaveFront ()`
  The class's Destructor.

- `WaveFront (const WaveFront &)`

- `WaveFront & operator= (const WaveFront &)`

- `bool operator== (const WaveFront &) const`

- `bool operator!= (const WaveFront &) const`

- `double * GetCoefficients (double dest[POLY_ORDER]) const`
  Getter method for the Coefficients property of the instance.

- `double GetCoefficient (unsigned int order)`
  Returning a single coefficient of the instance.

- `void SetCoefficients (double coeffs[POLY_ORDER])`
  Setter method for the Coefficients property of the instance.

- `void SetCoefficient (double coeff, int order)`
  Setting a single coefficient of the instance to a new value.

- `vector<double> getCoefficients ()`

- `double getCoefficient (unsigned int order)`

- `void setCoefficients (vector<double> coeffs)`

- `void setCoefficient (double coeff, int order)`
7.1. AOS_PresetAO

Description

The AOS_PresetAO command is issued in observation service status in order to prepare the AO system for an observation in adaptive mode. More details in reference [3], section 2.1.3

Syntax

```
Result * AOS_PresetAO(
    const char * AOMODE,
    const char * WFS,
    float SOCOORD1, float SOCOORD2,
    float ROCOORD1, float ROCOORD2,
    float ROTANGLE, float MAG,
    float COLOR [, float MRO],
    float WSPEED, float WANGLE )
```

Attributes

- **const char * AOMODE(in)**
  - **Purpose**: to specify the AO mode, either TTM-AO, ACE-AO or ICE-AO
  - **Description**: string variable to represent the AO mode.
  - **Unit**: unitless
  - **Range or possible value**: TTM-AO | ACE-AO | ICE-AO
  - **Default value**: 

- **const char * WFS(in)**
  - **Purpose**: to specify the source of WFS data.
  - **Description**: string variable
  - **Unit**: unitless
  - **Range or possible value**: TBD
  - **Default value**: 

- **float SOCOORD1, SOCOORD2 (in)**
  - **Purpose**: to specify the position of the scientific object in focal plane coordinates.
  - **Description**: double value.
  - **Unit**: mm
  - **Range or possible value**: [TBD.00, TBD.00]
  - **Default value**: 0.00

- **float ROCOORD1, ROCOORD2 (in)**
  - **Purpose**: to specify the position of the reference object in focal plane coordinates.
  - **Description**: double value
  - **Unit**: mm
  - **Range or possible value**: [-TBD.00, TBD.00]
  - **Default value**: 0.00

- **float ROTANGLE (in)**
Purpose: to specify the angular position of the rotator.
**Description:** double value
**Unit:** radians
**Range or possible value:** [-TBD.00, TBD.00]
**Default value:** 0.00

- **float MAG(in)**
  Purpose: to specify the magnitude of the reference star.
  **Description:** double value
  **Unit:** unitless
  **Range or possible value:** [-TBD, TBD]
  **Default value:** 0

- **float RO(in, optional)**
  Purpose: to specify the estimated value of RO.
  **Description:** double value
  **Unit:** unitless
  **Range or possible value:** [-TBD, TBD]
  **Default value:** 0

- **float WSPEED(in)**
  Purpose: to specify the wind speed.
  **Description:** double value
  **Unit:** m/s
  **Range or possible value:** [-TBD, TBD]
  **Default value:** 0

- **float WANGLE(in)**
  Purpose: to specify the wind direction.
  **Description:** double value
  **Unit:** radians
  **Range or possible value:** [-TBD, TBD]
  **Default value:** 0

Return values

- **Result * RESULT (returned)**
  **Purpose:** to evaluate if the command was accepted and successfully executed.
  **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle which can be used to query for further information.
  **Unit:** unitless
  **Range or possible value:** Result code = RRESCODE_FAIL: an error occurred.
  RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The Telescope must be in observation service status.
After execution

- AOS will wait for a change of the corresponding AO variable (TBD at the moment) reflected also in the data dictionary.
- AOS perform all set up operation needed, except acquisition of the reference object, which is performed with the subsequent AOS_AcuireRefAO command.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

Examples

... 

aResult = anIIF->AOS_PresetAO(TBD); 
...
7.2. AOS_AcquireRefAO

**Description**

AOS_AcquireRefAO, issued after a AOS_presetAO, requests the AOS to proceed into the reference object acquisition, in order to find the reference star within the field of view of the technical viewer. More details in reference [3], section 2.1.4

**Syntax**

```
Result * AOS_AcquireAO(const char * SIDE)
```

**Attributes**

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, "left", "right" or "both".
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: –

**Return values**

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be in observation service status.
- The instrument must check that the previous PresetAO command has been successfully completed, the telescope has reached the pointing position and the guiding system is operating.

**After execution**

- AOS will issue either a StartAO or a RefineAO command.
• The command returns the computed AO loop parameters, that can be retrieved via `CommandReturn`'s `getResultCount()` and `GetResultDescription(int n)` methods. This parameters are TBD. More information in reference [3], section 2.1.4.

Examples

```java
....
aResult = anIIF->TAOS_AcquireRef(SIDE_RIGHT);
....
```

7.3. AOS_RefineAO

Description

The AOS_RefineAO command is used to support the ICE-AO operating mode. It may be used to request the AOS to modify the value of some loop parameter before closing the loop.
More details in reference [3], section 2.1.5

Syntax

```c
Result * AOS_RefineAO(   int NMODES, float ITIME,
                           int NBINS, float TTMOD,
                           const char * F1SPEC,
                           const char * F2SPEC,
                           const char * SIDE)
```

Attributes

- **int NMODES**(in)
  - **Purpose**: to specify the number of correct modes.
  - **Description**: integer variable
  - **Unit**: unitless
  - **Range or possible value**: TBD
  - **Default value**: TBD
- **float ITIME**(in)
  - **Purpose**: to specify the CCD integration time.
  - **Description**: float value.
  - **Unit**: s
  - **Range or possible value**: [TBD, TBD]
  - **Default value**: 0
- **int NBINS**(in)
  - **Purpose**: to specify the CCD binning
  - **Description**: integer value
  - **Unit**:
  - **Range or possible value**: [-TBD, TBD]
  - **Default value**: 0
- **float TTMOD**(in)
  - **Purpose**: to specify the Tip-Tilt internal mirror modulation.
  - **Description**: float value
  - **Unit**: TBD
  - **Range or possible value**: [-TBD, TBD]
  - **Default value**: 0.00
- **const char * F1SPEC**(in)
  - **Purpose**: to specify the selected position of filter wheel number 1.
Description: string value.
Unit: unitless
Range or possible value: TBD
Default value: TBD

- const char * F2SPEC (in)
  Purpose: to specify the selected position of filter wheel number 2.
  Description: string value.
  Unit: unitless
  Range or possible value: TBD
  Default value: TBD

- const char * SIDE (in)
  Purpose: to specify the side this command applies to.
  Description: string to define the side, “left”, “right” or “both”.
  Unit: unitless
  Range or possible value: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  Default value: –

Return values

- Result * RESULT (returned)
  Purpose: to evaluate if the command was accepted and successfully executed.
  Description: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  Unit: unitless
  Range or possible value: Result code =
  RESCODE_FAIL: an error occurred.
  RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be in observation service status.
- PresetAO must be done, AquireAO must be done.

After execution

- AOS will wait for a change of the corresponding AO variable, reflected also in the data dictionary.
- AOS perform all set up operation needed, except acquisition of the reference object, which is performed with the subsequent AOS_AcuireRefAO command.
- The command returns the computed AO loop parameters, that can be retrieved via CommandReturn's getResultCount() and GetResultDescription(int n) methods.
  This parameters are TBD. More information in reference [3], section 2.1.5.
Examples

.....

aResult = anIIF->AOS_RefinAO(TBD);

.....
7.4. AOS_StartAO

Description

This command is used to request the closing of the AO loop. More details in reference [3], section 2.1.6

Syntax

```
Result * AOS_StartAO(const char * SIDE )
```

Attributes

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: –

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: Command accepted.

Preconditions

- The instrument must be authorized.
- The Telescope must be in observation service status.
- The instrument must check that the previous PresetAO command has been successfully completed, the telescope has reached the pointing position and the guiding system is operating.
- AOS must be fully ready to close the loop.

After execution

- Loop closed.
- We don't expect the TCS to send back any parameter. The evaluation
of the command's result will let us know if the command was successfully executed by the AOS.

Examples

....
aResult = anIIF->AOS_StartAO(SIDE_LEFT);
....
7.5. AOS_OffsetXY

Description

AOS_OffsetXY is issued in observation service status in order to offset the pointing of the AOS. This command is meaningful only in closed loop mode. More details in reference [3], section 2.1.7

Syntax

```
Result * AOS_OffsetXY( float XOFF, float YOFF,
                        const char * SIDE )
```

Attributes

- **float XOFF (in)**
  
  Purpose: to specify the requested position offset in X.
  
  Description: float value.
  
  Unit: mm
  
  Range or possible value: [TBD, TBD]
  
  Default value: 0

- **float YOFF (in)**
  
  Purpose: to specify the requested position offset in Y.
  
  Description: float value
  
  Unit: TBD
  
  Range or possible value: [-TBD, TBD]
  
  Default value: 0.00

- **const char * SIDE (in)**
  
  Purpose: to specify the side this command applies to.
  
  Description: string to define the side, “left”, “right” or “both”.
  
  Unit: unitless
  
  Range or possible value: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  
  Default value: -

Return values

- **Result * RESULT (returned)**
  
  Purpose: to evaluate if the command was accepted and successfully executed.
  
  Description: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  
  Unit: unitless
  
  Range or possible value: Result code = RESCODE_FAIL: an error occurred.
  RESCODE_OK: command accepted.

Preconditions
• The instrument must be authorized.
• The telescope must be in observation service status.
• AOS is operating in closed loop mode.

After execution

• Secondary mirror follows the offset, so this operation results in an offset of the field on the scientific camera.
• We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the AOS.

Examples

....
aResult = anIIF->AOS_OffsetXY(1.255, -0.815, SIDE_BOTH);
....
7.6. AOS_OffsetZ

**Description**

This command is issued in Observation service status in order to offset the focus of the AOS. It is meaningful only in closed loop mode. More details in reference [3], section 2.1.8

**Syntax**

```
Result * AOS_OffsetZ( float ZOFF, const char * SIDE )
```

**Attributes**

- **float ZOFF (in)**
  - **Purpose**: to specify the requested focus offset.
  - **Description**: float value.
  - **Unit**: mm
  - **Range or possible value**: [TBD, TBD]
  - **Default value**: 0

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

**Return values**

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred.
  - **RESCODE_OK**: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be in observation service status.
- AOS is operating in closed loop mode.

**After execution**
• Secondary mirror follows the offset, so this operation results in a change of focus on the scientific camera.
• We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the AOS.

Examples

....
aResult = anIIF->AOS_OffsetZ(1.955, SIDE_LEFT);
...

7.7. AOS_CorrectModes

Description

This command is used in observation service status to apply a modal correction on the mirror shape.
More details in reference [3], section 2.1.9

Syntax

| Result | AOS_CorrectModes(float DELTAM[762], const char * SIDE) |

Attributes

- **float DELTAM[762]** *(in)*
  - **Purpose**: to specify the modes correction vector.
  - **Description**: float vector with 762 elements.
  - **Unit**: TBD
  - **Range or possible value**: [TBD, TBD]
  - **Default value**: 0

- **const char * SIDE** *(in)*
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: –

Return values

- **Result * RESULT** *(returned)*
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred.
    RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be in observation service status.

After execution
- AOS sends the related request message to AO-Sup. The mirror change the shape.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the AOS.

Examples

```plaintext
....
float DeltaM[762];
....
aResult = anIIF->AOS_CorrectModes(DeltaM, SIDE_LEFT);
....
```
7.8. AOS_StopAO

Description

This command is used to stop the current operation. After this command any setting defined by a previous AOS_PresetAO is canceled. More details in reference [3], section 2.1.10

Syntax

```
Result * AOS_StopAO( const char * MSG,
                      const char * SIDE )
```

Attributes

- **const char * MSG (in)**
  - Purpose: to specify the reason for stopping.
  - Description: string value to represent the message for stopping.
  - Unit: unitless
  - Range or possible value: –
  - Default value: –

- **const char * SIDE (in)**
  - Purpose: to specify the side this command applies to.
  - Description: string to define the side, “left”, “right” or “both”.
  - Unit: unitless
  - Range or possible value: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - Default value: –

Return values

- **Result * RESULT (returned)**
  - Purpose: to evaluate if the command was accepted and successfully executed.
  - Description: Result structure received from the TCS with the result code, a text describing the result code and a request handle which can be used to query for further information.
  - Unit: unitless
  - Range or possible value: Result code = RESCODE_FAIL: an error occurred.
                            RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be in observation service status.
- AOS is in closed loop mode.
After execution

- AOS send the request message to AO-Sup, which properly updates the related variable in the data dictionary.
- AOS is in open loop mode.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the AOS.

Examples

```c
aResult = anIIF->AOS_StopAO("I'm tired", SIDE_LEFT);
```

...
7.9. AOS_PauseAO

**Description**

This command is issued to temporarily suspend the current operation. More details in reference [3], section 2.1.11

**Syntax**

```
Result * AOS_PauseAO( const char * SIDE )
```

**Attributes**

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

**Return values**

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be in observation service status.
- AOS is in closed loop mode.

**After execution**

- AOS sends the request message to AO-Sup, which properly updates the related status in the data dictionary.
- AOS is in pause mode.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the AOS.
Examples

....
result = anIIF->AOS_PauseAO(SIDE_BOTH);
....
7.10. AOS_ResumeAO

**Description**

This command resumes suspended operation after a AOS_PAUSEAO. More details in reference [3], section 2.1.12

**Syntax**

```c
Result * AOS_RESUMEAO( const char * SIDE )
```

**Attributes**

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** –

**Return values**

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RECODE_FAIL: an error occurred. RECODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
- A previous AOS_PAUSEAO must be issued.
- AOS is in pause mode.

**After execution**

- AOS will send the request message to AO-Sup. The it will properly update the related variable in the data dictionary.
- The loop is closed.
- We don’t expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was
successfully executed by the AOS.

Examples

....

aResult = anIIF->AOS_ResumeAO(SIDE_LEFT);
....
7.11. AOS_UserPanic

Description

This command is issued whenever some TCS subsystem, including an instrument, detects any dangerous condition, and decides to perform a fast shutdown. More details in reference [3], section 2.1.14

Syntax

```c
Result * AOS_UserPanic(const char * MSG
const char * SIDE )
```

Attributes

- **const char * MSG (in)**
  - **Purpose**: to specify the reason for panic.
  - **Description**: string value to represent the message for panic.
  - **Unit**: unitless
  - **Range or possible value**: –
  - **Default value**: -

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: –

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred.
    RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
After execution

- AOS sends the corresponding request message to AO-Sup.
- The connection is immediately closed.
- AOS shutdown ASAP\(^1\).
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the AOS.

Examples

....

aResult = anIIF->AOS_UserPanic("Ahhhh What I did !!!!", SIDE_LEFT);
....

\(^1\) After the acknowledge no other interaction will be possible between AOS and AO-Sup.
7.12. Authorize

**Description**

This command allows the instrument to authorize itself to control one or both sides of the Telescope. For further information, please see section 6.1.1, 6.1.2 and section 8 as well.

**Syntax**

```cpp
bool Authorize()
```

**Return values**

- **bool (returned)**
  - **Purpose:** to evaluate if the instrument was authorized to control the specified side. See section 8.1 for further details.
  - **Description:** bool value.
  - **Unit:** unitless
  - **Range or possible value:**
    - `true` authorization was granted by the TCS for the specified instrument and side on the IIF instance.
    - `false` The authorization failed, most likely another instrument already controls one or all of the Telescope sides specified in the failing IIF instance’s constructor.

**Preconditions**

- The instruments must create a new instance of the IIF class, identifying themselves by two pieces of information, their unique name (i.e. “MODS1” or “LUCIFER2”) and their focal station. (See section 8.1 for further details).

**After execution**

- Instruments are ready to initiate command processing inside the TCS via the IIF.

**Examples**

```cpp
....
IIF * anIIF;
try
{
  anIIF = new IIF("prime left", "LBCBLUE"); //pretend to the left-side,
```
catch (int e)
{
    cout << "The IIF failed to initialize, consult the SYSLOG for reasons";
    exit(-1);
}

//Permission for commanding the respective telescope sides is requested, and
//-upon failure to achieve authorization- the program is exited
if (!anIIF->Authorize())
{
    cout << "Authorize() was not successful ... terminating test program.";
    exit(-1);
}
cout << "Authorize() successful" << endl;
7.13. Deauthorize

Description

The Deauthorize command tells the TCS that the instrument is not using anymore one or both sides of the telescope, and yield all control to the TCS.

Syntax

```c
Result * Deauthorize(const char * SIDE)
```

Attributes

- **const char * SIDE (in)**
  - **Purpose**: value specifying the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL indicating an error RESCODE_OK The command was accepted.

Preconditions

- The instruments must be previously authorized.

After execution

- The CSQ count for the specified side is decremented. When the count is zero, this side transits into a “idle” state and another instrument can take control of it.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.
Examples

... 

aResult = anIIF->Deauthorize(SIDE_RIGHT);

...
7.14. GetMultiParameter

**Description**

This command is used to read a block of entries from the data dictionary in one shot.

**Syntax**

```
Result * GetMultiParameter(MultiDDEntry MULTIENTRIES)
```

**Attributes**

- **MultiDDEntry MULTIENTRIES (in)**  
  **Purpose:** to define the list of parameters to be read by the command from the data dictionary.  
  **Description:** MultiDDEntry structure to represent the list of data dictionary entries. See MultiDDEntry class above.  
  **Unit:** unitless  
  **Range or possible value:** Valid Data Dictionary entries.  
  **Default value:** –

**Return values**

- **Result * RESULT (returned)**  
  **Purpose:** to evaluate if the command was accepted and successfully executed.  
  **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.  
  **Unit:** unitless  
  **Range or possible value:** Result code = RESCODE_FAIL indicating an error or RESCODE_OK The command was accepted.

**Preconditions**

- None

**After execution**

- The command returns a list with the values of the data dictionary entries on the TCS side, that can be retrieved via CommandReturn's getResultCount() and GetResultDescription(int n) methods. More details in section 8.4
Examples

...  
MultiDDEntry DDEntries;
DDEntries.PushEntry("pmc.side[0].elevationAngle");
DDEntries.PushEntry("pmc.side[1].elevationAngle");
aResult = anIIF->GetMultiParameter(DDEntries);
...


7.15. GetParameter

Description

GetParameter request telescope status by querying a data dictionary entry.

Syntax

```c
Result * GetParameter( const char * DDENTRY )
```

Attributes

- **const char * DDENTRY (in)**
  - **Purpose**: to specify the data dictionary entry to be requested.
  - **Description**: string value.
  - **Unit**: unitless
  - **Range or possible value**: Valid Data Dictionary entry.
  - **Default value**: -

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL indicating an error or RESCODE_OK The command was accepted.

Preconditions

- None

After execution

- The command returns the value of the data dictionary entry on the TCS side, that can be retrieved via CommandReturn's getResultCount() and GetResultDescription(int n) methods.

Examples

```c
... 
aResult = anIIF->GetParameter("pmc.side[0].elevationAngle");
... 
```
7.16. LogEvent

Description

This command is issued to log a message inside the TCS's logging system.

Syntax

\[
\text{Result} * \text{LogEvent}( \text{const char} * \text{EVENTVALUE}, \\
\text{const char} * \text{EVENTDESCRIPTION} )
\]

Attributes

- **const char * EVENTVALUE (in)**
  - **Purpose**: to specify an event name to be logged in the TCS's logging system.
  - **Description**: string to define event value.
  - **Unit**: unitless
  - **Range or possible value**: –
  - **Default value**: –

- **const char * EVENTDESCRIPTION (in)**
  - **Purpose**: to specify a descriptive text that the instrument desires to be logged.
  - **Description**: string to define the event description.
  - **Unit**: unitless
  - **Range or possible value**: –
  - **Default value**: –

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.

Preconditions

- The instruments must be previously authorized.

After execution

- The telescope processes the logging request. The string is recorded in the format: “CSQ.<InstrumentID>.EventName.EventDescription” plus the date, time and host.
• We don't expect the TCS to send back any parameter. The evaluation of the command’s result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

**Examples**

...  

```cpp
aResult = anIIF->LogEvent("My_First_log", "Hello World!");
...
```
7.17. Move

Description

This command has the same functionality that MoveXY, TipTilt, StepFocus and MoveFocus. See the respective section for further details.

Syntax

```c
Result * Move (double X, double Y, double Z,
    double RX, double RY, double RZ,
    int D_FLAG, const char * MOVE_TYPE,
    const char * PSF_MODE, double TIME,
    const char * SIDE)
```

Attributes

- **double x, y, z (in)**
  
  **Purpose:** to specify the naked focal plane movements. MOVE_TYPE will determine if are absolute or relative values. More information about the coordinate system in Figure 6.3.
  
  **Description:** double value.
  
  **Unit:** millimeters
  
  **Range or possible value:** m1: [-4.000,4.000], m2:TBD m3:TBD
  
  **Default value:** TBD

- **double RX, RY, RZ (in)**
  
  **Purpose:** to specify the relative movements. MOVE_TYPE will determine if are absolute or relative values. More information about the coordinate system in Figure 6.3.
  
  **Description:** double value.
  
  **Unit:** micro radians
  
  **Range or possible value:** TBD
  
  **Default value:** TBD

- **int D_FLAG (in)**
  
  **Purpose:** 6 bits with a bit for each of the preceding 6 variables. A bit value of 1 means that the corresponding variable's value should be applied.
  
  **Description:** integer to represent the mask or flag, needed for absolute movements.
  
  **Unit:** unitless
  
  **Range or possible value:** [00000000, 00111111] bin.
  
  **Default value:**

- **string MOVE_TYPE (in)**
  
  **Purpose:** to determine if the movements are absolute or relative.
  
  **Description:** string to represent the move type, “absolute” or “relative”.
  
  **Unit:** unitless
  
  **Range or possible value:** MV_REL | MV_ABS
  
  **Default value:** TBD
• **string PSF_MODE (in)**
  
  **Purpose:** to determine which optical element(s) to move.
  
  **Description:** string to represent the PSF mode, “default”: TCS decides the OPE to move, “M1”, primary mirror, “M2” secondary mirror or “M1M2, both move together without a change in scale.
  
  **Unit:** unitless
  
  **Range or possible value:** DEFAULT | M1 | M2 | M1M2
  
  **Default value:** DEFAULT

• **double TIME (in)**
  
  **Purpose:** to specify the absolute time in the future for when to apply the move command. 0 = ASAP.
  
  **Description:** double value.
  
  **Unit:** TBD
  
  **Range or possible value:** TBD
  
  **Default value:** 0

• **const char * SIDE (in)**
  
  **Purpose:** to specify the side this command applies to.
  
  **Description:** string to define the side, “left”, “right” or “both”.
  
  **Unit:** unitless
  
  **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  
  **Default value:** -

Return values

• **Result * RESULT (returned)**
  
  **Purpose:** to evaluate if the command was accepted and successfully executed.
  
  **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  
  **Unit:** unitless
  
  **Range or possible value:** Result code = RESCODE_FAIL: an error occurred.
  
  RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The OPE can be moved.

After execution

- OPE is on a new position.
- We don’t expect the TCS to send back any parameter. The evaluation of the command’s result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.
Examples

... 

aResult = anIIF->Move( 1.42, 1.03, 1.0, 1.0, 0.4, 0.4, 255, 
                         MV_REL, M1, 0, SIDE_LEFT );

...

7.18. MoveFocus

Description

MoveFocus moves an optical element to a new absolute position z to adjust or to define a new focus position. (More information about the coordinate system in figure 6.3)

Syntax

```
Result * MoveFocus(double ABSPOS, const char * SIDE, 
                    const char * OPE)
```

Attributes

- **double ABSPOS (in)**
  - **Purpose**: to specify the new absolute position Z of the respective optical element.
  - **Description**: double value.
  - **Unit**: millimeter
  - **Range or possible value**: [-4.000 , 4.000]
  - **Default value**: 0

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

- **const char * OPE (in)**
  - **Purpose**: to specify the optical element which should move in Z direction. The OPE could be the primary, the secondary or the tertiary mirror.
  - **Description**: string to define the optical element (OPE), “primary”, “secondary” or “tertiary”.
  - **Unit**: unitless
  - **Range or possible value**: M1 | M2 | M3
  - **Default value**: M1

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.
Preconditions

- The instrument must be authorized.
- The Telescope must be observing
- The OPE can be moved
- The telescope is tracking and/or guiding and/or controlling AO.

After execution

- OPE is on a new position.
- Telescope is observing.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

Examples

... 
    aResult = anIIF->MoveFocus (1.42, SIDE_LEFT, M1);
    ...
    ...

7.19. MoveXY

Description

The MoveXY command moves an OPE in X or Y direction, relative to the current position. (More information about the coordinate system in figure 6.3)

Syntax

\[
\text{Result} \ast \text{MoveXY}( \ double \ XMOTION, \ double \ YMOTION, \\
const \ char \ast \ SIDE, \ const \ char \ast \ OPE)
\]

Attributes

- **double XMOTION (in)**
  - **Purpose:** to specify the motion along the x axis, relative to the current position with a precision of micro meters.
  - **Description:** double value to represent the motion along x axis.
  - **Unit:** millimeter
  - **Range or possible value:** [-4.000, 4.000]
  - **Default value:** 0

- **double YMOTION (in)**
  - **Purpose:** to specify the motion along the y axis, relative to the current position with a precision of micro meters.
  - **Description:** double value to represent the motion along y axis.
  - **Unit:** millimeter
  - **Range or possible value:** [-4.000, 4.000]
  - **Default value:** 0

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

- **const char * OPE (in)**
  - **Purpose:** to define the optical element this command applies to. The OPE could be the primary, the secondary or the tertiary mirror.
  - **Description:** string to define the optical element (OPE), “primary” or “secondary”.
  - **Unit:** unitless
  - **Range or possible value:** M1 | M2
  - **Default value:** M1

Return values

- **Result * RESULT (returned)**
**Purpose:** to evaluate if the command was accepted and successfully executed.

**Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.

**Unit:** unitless

**Range or possible value:** Result code =

- RESCODE_FAIL: an error occurred.
- RESCODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be observing
- The OPE can be moved
- The telescope is tracking and/or guiding and/or controlling AO.

**After execution**

- OPE is on a new position.
- Telescope is observing.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

**Examples**

```
....
aResult = anIIF->MoveXY(1.000, -0.810 , SIDE_LEFT, M1);
...
```
7.20. MoveXYZ

Description

The MoveXYZ command moves the primary and secondary together in X, Y and Z direction. The movement is relative and not synchronized between the OPE. (More information about the coordinate system in figure 6.3)

Syntax

```
Result * MoveXYZ( double RELX, double RELY, double RELZ,
                   const char * SIDE )
```

Attributes

- **double RELX (in)**
  - **Purpose**: to specify the relative movement in X direction.
  - **Description**: double value.
  - **Unit**: millimeters
  - **Range or possible value**: m1: [-4.000,4.000], m2:[TBD]
  - **Default value**: 0

- **double RELY (in)**
  - **Purpose**: to specify the relative movement in Y direction.
  - **Description**: double value.
  - **Unit**: millimeters
  - **Range or possible value**: m1: [-4.000,4.000], m2:[TBD]
  - **Default value**: 0

- **double RELZ (in)**
  - **Purpose**: to specify the relative movement in X, Y, and Z direction.
  - **Description**: double value.
  - **Unit**: millimeters
  - **Range or possible value**: m1: [TBD,TBD], m2:[TBD]
  - **Default value**: 0

- **const char * SIDE (in)**
  - **Purpose**: to define the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

Return values

- **Result * RESULT (returned)**
  - **Purpose**: to evaluate if the command was accepted and successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to
query for further information.

Unit: unitless
Range or possible value: Result code =
RESCODE_FAIL: an error occurred.
RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be observing
- The OPE can be moved
- The telescope is tracking and/or guiding and/or controlling AO.

After execution

- OPE is on a new position.
- Telescope is observing.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

Examples

....

aResult = anIIF->MoveXYZ(1.000, -0.810 , SIDE_LEFT, M1);
...

....
7.21. OffsetGuiding

**Description**

OffsetGuiding moves the telescope a small distance without changing the value of the pointing coordinates. At the moment, this command is only needed by LBC, LN and PEPSI.

**Syntax**

```c
Result * OffsetGuiding( Position * OFFSET, const char * SIDE [, double ROTANGLE])
```

**Attributes**

- **Position * OFFSET (in)**
  - **Purpose:** to specify the desired guiding offset from the current position to the science object.
  - **Description:** Position object to define the guiding offset.
  - **Unit:** see Position class above.
  - **Range or possible value:** see Position class above.
  - **Default value:** -

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

- **double ROTANGLE (in, optional)**
  - **Purpose:** to specify the value in radians of the optional rotation angle.
  - **Description:** double value.
  - **Unit:** radian
  - **Range or possible value:** [-6.283185307, 6.283185307]
  - **Default value:** 0 (do not use ROTANGLE)

**Return values**

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.
Preconditions

- The instrument must be authorized.
- The telescope must be observing
- The telescope can be moved
- The telescope is tracking and/or guiding and/or controlling AO.

After execution

- The telescope is on a new position, tracking, guiding and controlling the AOS.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

Examples

... MagnitudeType magnitude = {-27.0, "IV", 0, "B-U"}; offset = new Position(2.354, .1234, COORD_RADEC, "B2003.50", "2003.123", NULL, &magnitude, 3421); //call OffsetGuiding() aResult = anIIF->offsetGuiding(offset, SIDE_LEFT, -1.1); ...
7.22. OffsetPointing

Description

OffsetPointing moves the telescope a small distance, setting the value of the telescope pointing coordinates to the new position. This command uses the existing epoch, wavelength, and the setup defined in the previous preset command.

Syntax

```c
Result * OffsetPointing( double ROTANGLE, coord OFFSET,
                         const char * OFF_OBJ,
                         bool CORRECT_POINTING_FLAG,
                         const char * SIDE )
```

Attributes

- **double** ROTANGLE *(in)*
  - **Purpose**: to specify the value in radians of the rotation angle.
  - **Description**: double value.
  - **Unit**: radian
  - **Range or possible value**: [-6.283185307, 6.283185307]
  - **Default value**: -

- **coord** OFFSET *(in)*
  - **Purpose**: to specify the desired offset from the current position to the science object.
  - **Description**: Position object to define the offset.
  - **Unit**: see Position class above.
  - **Range or possible value**: see Position class above.
  - **Default value**: -

- **const char** * OFF_OBJ *(in)*
  - **Purpose**: to define the element this command applies to.
  - **Description**: string to define the object, “mount”, “m1”, “m2”, “m3”, “hexapod”, or “guider”.
  - **Unit**: unitless
  - **Range or possible value**: MOUNT | M1 | M2 | M3 | HEXAPOD | GUIDER
  - **Default value**: M1

- **bool** CORRECT_POINTING_FLAG *(in)*
  - **Purpose**: to determine if the target position should be changed or not.
  - **Description**: bool variable. True = move the guide stage but do not change target RA and DEC. False = Update RA and DEC (dither).
  - **Unit**: unitless
  - **Range or possible value**: true | false
  - **Default value**: true

- **const char** * SIDE *(in)*
  - **Purpose**: to specify the side this command applies to.
**Description**: string to define the side, “left”, “right” or “both”.

**Unit**: unitless

**Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH

**Default value**: -

**Return values**

- **Result ** RESULT (returned)

  **Purpose**: to evaluate if the command was accepted and successfully executed.

  **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.

  **Unit**: unitless

  **Range or possible value**: Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be observing
- The telescope can be moved
- The telescope is tracking and/or guiding and/or controlling AO.

**After execution**

- The telescope is on a new position, tracking, guiding and controlling the AOS.
- The pointing coordinates in the data dictionary do not change.
- New pointing coordinates.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

**Examples**

...  
offset = {1.65, 4.35, COORD_ALTAZ }
//call OffsetPointing()
aResult = anIIF->**OffsetPointing**(28.4567891e-2, offset, M1, true, SIDE_LEFT);
...
7.23. PresetTelescope

Description

Slew the telescope into position in order to begin an observation cycle.

![Diagram](image)

**Figure 7.1**: Positions in the focal plane for `PresetTelescope`.

Syntax

```c
Result * PresetTelescope(double ROTANGLE,
    const char* TRACKMODE,
    Position* TARGET,
    Position** GUIDESTARS,
    const char* MODE,
    const char* SIDE,
    [Position* OFFSET][,Position* HOTSPOT])
```

Attributes

- **double ROTANGLE (in)**
  - **Purpose**: to specify the value in radians for the initial rotation angle.
  - **Description**: double value to represents the IRA.
  - **Unit**: radian
  - **Range or possible value**: [-6.283185307, 6.283185307]
  - **Default value**: 0

- **const char * TRACKMODE (in)**
  - **Purpose**: to specify the tracking mode of the instrument rotator.
  - **Description**: string that represent the tracking mode. This could be: TRACK_VERT (“vertical”) using the parallactic angle, TRACK_PSTN (“position”) relative to the north or TRACK_IDLE (“idle”) for those instruments that will not use the rotator at all.
  - **Unit**: radian
  - **Range or possible value**: TRACK_VERT | TRACK_PSTN | TRACK_IDLE
  - **Default value**: -

- **Position * TARGET (in)**
Purpose: to specify the right ascension, declination, epoch, coordinate system (COORD_RADEC, COORD_ALTAZ, or COORD_FOCAL), presumed magnitude, presumed wavelength and proper motion of the science target.

Description: Position object to define the target.
Unit: See Position class above.
Range or possible value: See Position class above.
Default value: -

- **GUIDESTARS (in)**
  
  Purpose: to specify the right ascension, declination, epoch, coordinate system (COORD_RADEC, COORD_ALTAZ, or COORD_FOCAL), presumed magnitude, presumed wavelength and proper motion of the set of guide stars.
  
  Description: list of positions to define the set of guide stars.
  Unit: See Position class above.
  Range or possible value: See Position class above.
  Default value: -

- **MODE (in)**
  
  Purpose: to specify the operation mode of the telescope.
  
  Description: string representing the telescope mode operation: either “passive”, “active”, “guided”, “adaptive” or “interferometric”
  
  Unit: unitless
  Range or possible value: MODE_PSV | MODE_ACT | MODE_GUD | MODE_ADP | MODE_INF
  
  Default value: -

- **SIDE (in)**
  
  Purpose: to specify the side this command applies to.
  
  Description: string to define the side, “left”, “right” or “both”.
  
  Unit: unitless
  Range or possible value: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  
  Default value: -

- **OFFSET (in, optional)**
  
  Purpose: to specify the offset for the target in RA and DEC, ALT and AZ, or SFP coordinates.
  
  Description: Position object to define the pointing offset.
  
  Unit: see Position class above.
  Range or possible value: see Position class above.
  
  Default value: NULL

- **HOTSPOT (in, optional)**
  
  Purpose: to specify the reference position in the focal plane, by default, the center of the focal plane. See figure 7.1
  
  Description: Position object.
  
  Unit: see Position class above.
  Range or possible value: see Position class above.
  
  Default value: NULL

Return values

- **RESULT (returned)**
  
  Purpose: to evaluate if the command was successfully executed.
  
  Description: Result structure received from the TCS with the result code, a text
describing the result code and a request handle, which can be used to query for further information.

**Unit:** unitless

**Range or possible value:**
- Result code = RESCODE_FAIL: an error occurred.
- RESCODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be ready to move, to track and control the active optic system (AOS).

**After execution**

- The telescope is on a new position and tracking.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

**Examples**

```c
....
MagnitudeType magnitude = {-27.0, "IV", 1.0, "U-B"};
ProperMotionType propm = {0.123, 0.234};
Position *offset = new Position( 2.354, .1234, COORD_RADEC, "B2003.50", "2007", NULL, &magnitude, 3421);

Position *hotspot = new Position( 3.5, .6, COORD_SFP, "-1", &propm, &magnitude, 0);

Position *target = new Position( 3.5, .6, COORD_RADEC, "J2003.123", "2007", &propm, &magnitude1, 2020);

offset = new Position( 2.354, .1234, COORD_RADEC, "B2003.50", "2007", NULL, &magnitude, 3421);
guidestars = (Position **) malloc(3 * sizeof(Position *));
guidestars[0] = target;
guidestars[1] = offset;
guidestars[2] = (Position *) NULL;

aResult = anIIF->PresetTelescope( 1.0, "position", target, guidestars, "active", "left");

....
```
7.24. Preset_guiding

Description

This command is issued to start the guiding.

Syntax

```
Result * Preset_Guiding ( Position ** guideStars,
const char * SIDE )
```

Attributes

- **Position ** guideStars (in)
  
  **Purpose**: to specify the list of guide stars.
  
  **Description**: Position structure to define the guide star list.
  
  **Unit**: unitless
  
  **Range or possible value**: -
  
  **Default value**: 0

- **const char * SIDE (in)**
  
  **Purpose**: to define the side this command applies to.
  
  **Description**: string to define the side, “left”, “right” or “both”.
  
  **Unit**: unitless
  
  **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  
  **Default value**: -

Return values

- **Result * RESULT (returned)**
  
  **Purpose**: to evaluate if the command was successfully executed.
  
  **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  
  **Unit**: unitless
  
  **Range or possible value**: Result code = RESCODE_FAIL: an error occurred.
  RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be observing and tracking.
- The guide unit must be ready to be closed.
After execution

- The guide star is acquired.
- The telescope is observing and guiding.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

Examples

....
....
7.25. RotatePrimary

Description

The RotatePrimary command rotates the primary mirror around a fixed reference point.

![Primary mirror rotation diagram.](image)

**Figure 7.2**: Primary mirror rotation diagram.

Syntax

```
Result * RotatePrimary(double DISTANCE, double ANGLE,
                        double DIRECTION, const char * SIDE)
```

Attributes

- **double DISTANCE (in)**
  - **Purpose**: to specify the rotation reference point’s distance above the mirror.
  - **Description**: double value
  - **Unit**: millimeter
  - **Range or possible value**: [999.999 , 99999.999]
  - **Default value**: 0

- **double ANGLE (in)**
  - **Purpose**: to specify the value of the rotation angle with a precision of micro radians.
  - **Description**: double value representing the angle.
  - **Unit**: radians
  - **Range or possible value**: [-0.000 , 999.999]
  - **Default value**: 0

- **double DIRECTION (in)**
  - **Purpose**: to specify the direction of the rotation.
  - **Description**: double value to represent the direction (between 0 and 2pi).
  - **Unit**: radians
  - **Range or possible value**: [0.000000000 , 6.283185307]
  - **Default value**: 0.000000000

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”. 
Unit: unitless
Range or possible value: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
Default value: -

Return values

• **Result * RESULT (returned)**
  
  Purpose: to evaluate if the command was accepted and successfully executed.
  
  Description: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.

  Unit: unitless
  Range or possible value: Result code = RESCODE_FAIL: an error occurred.
  RESCODE_OK: command accepted.

Preconditions

• The instrument must be authorized.
• The telescope must be observing
• The OPE can be moved
• The telescope is tracking and/or guiding and/or controlling AO.

After execution

• Primary mirror is on a new position.
• Telescope is observing.
• We don’t expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

Examples

```c
aResult = anIIF->RotatePrimary(1200.567, 110.499, 3.141592654, SIDE_LEFT);
...
```
7.26.RotateZ

Description

Using this command, the tertiary mirror is rotated to adjust the incoming beam angle for the instrument.

Syntax

```
Result * RotateZ ( double ANGLE, const char * SIDE)
```

Attributes

- **double ANGLE (in)**
  - **Purpose:** to specify the value of the angle to be rotated.
  - **Description:** double value.
  - **Unit:** radians
  - **Range or possible value:** TBD
  - **Default value:** 0

- **const char * SIDE (in)**
  - **Purpose:** to define the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: Command accepted.

Preconditions

- The instrument must be authorized.
- The OPE can be moved.

After execution

- Tertiary mirror (M3) is on a new position.
• We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

Examples

....
    aResult = anIIF->RotateZ(1.35, SIDE_LEFT);
    ....
7.27. Rot_Set_Rotator

**Description**

This command is issued to activate or deactivate the rotator associated with the specified focal station.

**Syntax**

```c
Result * Rot_Set_Rotator ( const char * FOCALSTATION,
                           const char * SIDE,
                           bool ENABLE )
```

**Attributes**

- **const char * FOCALSTATION (in)**
  - **Purpose:** to define the focal station.
  - **Description:** string value.
  - **Unit:** unitless
  - **Range or possible value:** FS_LDG | FS_LFBG | FS_LCBG | FS_LRBG | FS_RDG | FS_RFBG | FS_RCBG | FS_RRBG
  - **Default value:** -

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

- **bool ENABLE (in)**
  - **Purpose:** to determine whether to activate or deactivate the rotator associated with the specified focal station.
  - **Description:** bool variable.
  - **Unit:** unitless
  - **Range or possible value:** true | false
  - **Default value:** true

**Return values**

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred.
                             RESCODE_OK: Command accepted.
Preconditions

- The instrument must be authorized.

After execution

- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the TCS.

Examples

....

aResult = anIIF->Rot_SetRotator(FS_DIRECTGREGORIAN, SIDE_LEFT, true);

....
7.28. Rot_Force_Wrap

**Description**

See MCSPU rotator control document.

**Syntax**

```c
Result * Rot_Force_Wrap (   const char * FOCALSTATION,
                            const char * SIDE,
                            int FORCE )
```

**Attributes**

- **const char * FOCALSTATION (in)**
  - **Purpose:** to specify the focal station.
  - **Unit:** unitless
  - **Range or possible value:** FS_LDG | FS_LFBG | FS_LCG | FS_LRGB | FS_RDG | FS_RFBG | FS_RCG | FS_RRGB
  - **Default value:**

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string defining the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

- **int FORCE (in)**
  - **Purpose:** to specify the direction of the force.
  - **Description:** The force argument has possible values of: -1 = negative (counter-clock wise) wrap, 0 = no preference (use the default which is to follow the shortest path), and 1 = positive (clock wise) wrap.
  - **Unit:** unitless
  - **Range or possible value:** -1 | 0 | 1
  - **Default value:** 0

**Return values**

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
Unit: unitless
Range or possible value: Result code = RESCODE_FAIL: an error occurred.
                    RESCODE_OK: command accepted.

Preconditions

• The instrument must be authorized.
• ******

After execution

• The rotator is tracking.
• We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

Examples

....
aResult = anIIF->Rot_forceWrap(FS_DIRECTGREGORIAN, SIDE_LEFT, -1);
....
7.29. Rot_Track

Description

This command activates the tracking action for the specified rotator. See reference [4], section 7.1 for further details.

Syntax

```
Result * Rot_Track ( const char * FOCALSTATION,
                      const char * SIDE )
```

Attributes

- **const char * FOCALSTATION (in)**
  - **Purpose:** to specify the focal station.
  - **Unit:** unitless
  - **Range or possible value:** FS_LDG | FS_LFBG | FS_LCBG | FS_LRBG | FS_RDG | FS_RFBG | FS_RCBG | FS_RRBG
  - **Default value:**

- **const char * SIDE (in)**
  - **Purpose:** to define the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred.
                              RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
The rotator is tracking.

**After execution**

- The rotator stops.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the TCS.

**Examples**

```c
aResult = anIIF->Rot_Track(FS_DIRECTGREGORIAN, SIDE_LEFT);
```
7.30. Rot_Hold

Description

This command puts the specified rotator in a hold state. See reference [4] for further details.

Syntax

```
Result * Rot_Hold ( const char * FOCALSTATION,
                     const char * SIDE )
```

Attributes

- **const char * FOCALSTATION (in)**
  - **Purpose:** to specify the focal station.
  - **Unit:** unitless
  - **Range or possible value:** FS_LDG | FS_LFBG | FS_LCBG | FS_LRBG | FS_RDG | FS_RFBG | FS_RCBG | FS_RRBG
  - **Default value:** -

- **const char * SIDE (in)**
  - **Purpose:** to define the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred.
                             RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
After execution

- *****

- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

Examples

... 
aResult = anIIF->Rot_Hold(FS_DIRECTGREGORIAN, SIDE_LEFT);
...
7.31. Rot_Adjust_Position

Description

This command allows the instrument to input fine adjustments to the current RA, Dec, and rotator angle values. More information in reference [4], section 7.1.

Syntax

```plaintext
Result * Rot_Adjust_Position ( double DELTA_RA,
                               double DELTA_DEC,
                               double DELTA_ROTANGLE )
```

Attributes

- **double DELTA_RA (in)**
  - **Purpose:**
  - **Description:** double value.
  - **Unit:** Radian
  - **Range or possible value:** TBD
  - **Default value:** 0

- **double DELTA_DEC (in)**
  - **Purpose:**
  - **Description:** double value.
  - **Unit:** Radian
  - **Range or possible value:** TBD
  - **Default value:** 0

- **double DELTA_ROTANGLE (in)**
  - **Purpose:**
  - **Description:** double value.
  - **Unit:** Radian
  - **Range or possible value:** TBD
  - **Default value:** 0

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.
Preconditions

- The instrument must be authorized.
- *****

After execution

- *****
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

Examples

....
aResult = anIIF->Rot_AdjustPosition(1.00, 0.098, 2.02 );
....
7.32. Rot_TimeToIntercept

Description

This command is issued to determine the amount of time left on the instrument and azimuth cable wraps, as well as a time to interception with the horizon. More information in reference [4], section 7.1.

Syntax

```
Result * Rot_TimeToIntercept(
    const char * FOCALSTATION,
    const char * SIDE,
    double DELTA_RA,
    double DELTA_DEC,
    float EXPOSURELENGTH)
```

Attributes

- **const char * FOCALSTATION (in)**
  - **Purpose:** to specify the focal station.
  - **Unit:** unitless
  - **Range or possible value:** FS_LDG | FS_LFBG | FS_LCBG | FS_LRBG | FS_RDG | FS_RFBG | FS_RCBG | FS_RRBG
  - **Default value:**

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:**

- **double DELTA_RA (in)**
  - **Purpose:**
  - **Description:** double value.
  - **Unit:** Radian
  - **Range or possible value:** TBD
  - **Default value:** 0

- **double DELTA_DEC (in)**
  - **Purpose:**
  - **Description:** double value.
  - **Unit:** Radian
  - **Range or possible value:** TBD
Default value: 0

- **float** EXPOSURELENGTH (in)
  
  **Purpose:** to compute estimates for the time remaining on the instrument and azimuth cable wraps, as well as an estimate of the amount of time remaining before the specified target dips below the horizon.
  
  **Description:** float value defining the exposure length.
  
  **Unit:** second
  
  **Range or possible value:** TBD
  
  **Default value:** TBD

**Return values**

- **Result ** RESULT (returned)
  
  **Purpose:** to evaluate if the command was accepted and successfully executed.
  
  **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  
  **Unit:** unitless
  
  **Range or possible value:** Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.

**Preconditions**

- The instrument must be authorized.
  
  *****

**After execution**

- *****
  
  - We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

**Examples**

....

aResult = anIIF->Rot_TimeToIntercept(FS_LDG, SIDE_LEFT, 2.245, 1.231, 100 );

....
7.33.SendWavefront

**Description**

SendWavefront sends an array of zernike coefficients to be compensated by the primary mirror actuators. Reducing atmospheric aberrations, this command should improve the image quality.

**Syntax**

```c
Result * SendWavefront( Wavefront * POLYNOM,
                        const char * OPE,
                        const char * SIDE )
```

**Attributes**

- **Wavefront * POLYNOM (in)**
  - **Purpose:** to define the 28 Zernike polynomial coefficients with a precision of 0.01 nm for wavefront correction.
  - **Description:** Wavefront object to define the Zernike coefficients. See Wavefront class above.
  - **Unit:** nanometers
  - **Range or possible value:** [-99999.99, 99999.99]
  - **Default value:** 0 (all coefficients of the WF instance are pre-populated with 0.0)

- **const char * OPE (in)**
  - **Purpose:** to specify the optical element this command applies to. The OPE could be the primary mirror or the secondary mirror.
  - **Description:** string to define the optical element (OPE), “primary” or “secondary”.
  - **Unit:** unitless
  - **Range or possible value:** M1 | M2
  - **Default value:** M1

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** -

**Return values**

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
**Range or possible value:** Result code =

- **RESCODE_FAIL:** an error occurred.
- **RESCODE_OK:** command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be observing.
- The telescope can be moved.
- The telescope is tracking and/or guiding.

**After execution**

- Mirror in new shape, means better image quality.
- Telescope observing.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the TCS.

**Examples**

```java
... WaveFront *wfs = new WaveFront(); // Dummy instance. All coefficients 0.0
wfs->setCoefficient(812.0, 1); /* set one coefficient (#9) non-zero */
wfs->setCoefficient(999.4, 2); /* set one coefficient (#9) non-zero */
wfs->setCoefficient(1120.0, 3); /* set one coefficient (#9) non-zero */
wfs->setCoefficient(1102.8, 4); /* set one coefficient (#9) non-zero */

// Call SendWavefront()
aResult = anIIF->SendWavefront(wfs, M1, SIDE_LEFT);
...```

7.34.SetMultiParameter

Description

SetMultiParameter sets the values of the specified data dictionary entries on the TCS in one shot. Note, that the instrument only has permission to modify its own predefined entries.

Syntax

```
Result * SetMultiParameter( MultiDDEntry MULTIENTRIES )
```

Attributes

- **MultiDDEntry MULTIENTRIES (in)**
  - **Purpose:** to define the structure specifying the list of parameters and values to set in the data dictionary.
  - **Description:** MultiDDEntry structure to represent the list of data dictionary entries and their respective values.
  - **Unit:** unitless
  - **Range or possible value:** Valid Data Dictionary entries / values.
  - **Default value:** -

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.

Preconditions

- None

After execution

- New values for the specified data dictionary entries.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.
Examples

...  
MultiDDEntry DDEntries;
DDEntries.PushEntry("LINC.side[0].cooler", "ON");
DDEntries.PushEntry("LINC.side[0].cooler", "OFF");
aResult = anIIF->SetMultiParameter(DDEntries);
...
7.35. SetParameter

**Description**

Allows the instruments to set a new value of a data dictionary entry on the TCS.

**Syntax**

```
Result * SetParameter( const char * DDENTRY,
                       const char * DDENTRYVALUE )
```

**Attributes**

- **const char * DDENTRY (in)**
  - **Purpose:** to specify the name of the data dictionary entry to set with a new value.
  - **Description:** string to define the data dictionary entry.
  - **Unit:** unitless
  - **Range or possible value:** instrument valid Data Dictionary entry.
  - **Default value:** –

- **const char * DDENTRYVALUE (in)**
  - **Purpose:** to specify the new value of the data dictionary entry.
  - **Description:** string variable to define the new value of the data dictionary entry.
  - **Unit:** unitless
  - **Range or possible value:** Valid Value.
  - **Default value:** –

**Return values**

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code =
    - RESCODE_FAIL: an error occurred.
    - RESCODE_OK: command accepted.

**Preconditions**

- None

**After execution**

- The command sets the new value of the data dictionary entry on the TCS side.
• We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the TCS.

Examples

... 
arResult = anIIF->SetParameter("MODS.side[0].coolers","OFF");
...

...
7.36. Standby

Description

Standby command tells the TCS that currently the instrument is not using one or both of the telescope sides, but the side(s) will not given over to another instrument for control.

Syntax

\[
\text{Result * Standby}(\text{const char * SIDE, int LEVEL})
\]

Attributes

- \textbf{const char * SIDE (in)}
  - \textbf{Purpose:} to specify the side this command applies to.
  - \textbf{Description:} string to define the side, “left”, “right” or “both”.
  - \textbf{Unit:} unitless
  - \textbf{Range or possible value:} SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - \textbf{Default value:} -

- \textbf{int LEVEL (in)}
  - \textbf{Purpose:} to specify the new value of the standby level
  - \textbf{Description:} integer value to represent the standby level.
  - \textbf{Unit:} unitless
  - \textbf{Range or possible value:} TBD
  - \textbf{Default value:} TBD

Return values

- \textbf{Result * RESULT (returned)}
  - \textbf{Purpose:} to evaluate if the command was accepted and successfully executed.
  - \textbf{Description:} Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - \textbf{Unit:} unitless
  - \textbf{Range or possible value:} Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: Command accepted.

Preconditions

- The instruments must be previously authorized.

After execution

- The specified side is in an “standby” state. No other instrument can
take control of it.

- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the TCS.

Examples

... 
aResult = anIIF->Standby(SIDE_RIGHT, 4); 
...
7.37. Start_Guiding

Description

Syntax

```c
Result * Start_Guiding()
```

Attributes

Return values

- **Result * RESULT (returned)**
  
  **Purpose:** to evaluate if the command was accepted and successfully executed.
  
  **Description:** Result structure received from the TCS with the result code, a text
  describing the result code and a request handle, which can be used to
  query for further information.
  
  **Unit:** unitless
  
  **Range or possible value:** Result code = RESCODE_FAIL: an error occurred.
  RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- Preset telescope must be done.
- Telescope is tracking

After execution

- Telescope is tracking and guiding.
- We don't expect the TCS to send back any parameter. The evaluation
  of the command's result will let us know if the command was
  successfully executed by the AOS.

Examples

```c
    aResult = anIIF->AOS_Start_Guiding();
    ....
```
7.38. StepFocus

Description

StepFocus moves the respective focus position, by moving the OPE a given distance in the direction of the optical surface's Z axis.

Syntax

```
Result * StepFocus( double RELPOS, const char * SIDE,
                    const char * OPE )
```

Attributes

- **double RELPOS (in)**
  - **Purpose:** to specify the new relative position Z of the respective optical element.
  - **Description:** double value to represent the delta to the current position.
  - **Unit:** millimeter
  - **Range or possible value:** [-4.000 , 4.000]
  - **Default value:** 0

- **const char * SIDE (in)**
  - **Purpose:** to specify the side this command applies to.
  - **Description:** string to define the side, “left”, “right” or “both”.
  - **Unit:** unitless
  - **Range or possible value:** SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value:** –

- **const char * OPE (in)**
  - **Purpose:** to specify the optical element which should move in Z. The OPE could be the primary mirror, the secondary or the tertiary mirror.
  - **Description:** string to define the optical element (OPE), “primary”, “secondary” or “tertiary”.
  - **Unit:** unitless
  - **Range or possible value:** M1 | M2 | M3
  - **Default value:** M1

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit:** unitless
  - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred. RESCODE_OK: command accepted.
Preconditions

- The instrument must be authorized.
- The telescope must be observing.
- The OPE can be moved.
- The telescope is tracking and might be guiding or controlling AOS.

After execution

- OPE is on a new position.
- Telescope is observing, tracking and guiding.
- We don't expect the TCS to send back any parameter. The evaluation of the command's result will let us know if the command was successfully executed by the TCS.

Examples

... 

aResult = anIIF->StepFocus(-1.42, SIDE_LEFT, M1);
...
7.39. Stop_Guiding

Description

Syntax

```c
Result * Stop_Guiding()
```

Attributes

Return values

- **Result * RESULT (returned)**
  - **Purpose:** to evaluate if the command was accepted and successfully executed.
  - **Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
    - **Unit:** unitless
    - **Range or possible value:** Result code = RESCODE_FAIL: an error occurred.
      RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be observing and guiding.

After execution

- The telescope is observing, tracking and not guiding.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the AOS.

Examples

```c
aResult = anIIF->Stop_Guiding();
```
7.40. Tip/Tilt

Description

The TipTilt command moves an OPE in tip or tilt direction, relative to the current position.

Syntax

```
Result * TipTilt( double XROTATION, double YROTATION,
                  const char * SIDE, const char * OPE )
```

Attributes

- **double XROTATION (in)**
  - **Purpose**: to specify the angle to rotate around x axis, relative to the current position with a precision of 0.001 micro radians.
  - **Description**: double value to represent the angle to rotate around x axis.
  - **Unit**: micro radians
  - **Range or possible value**: [-999.999 , 999.999]
  - **Default value**: 0

- **double YROTATION (in)**
  - **Purpose**: to specify the angle to rotate around y axis, relative to the current position with a precision of 0.001 micro radians.
  - **Description**: double value to represent the angle to rotate around y axis.
  - **Unit**: micro radians
  - **Range or possible value**: [-999.999 , 999.999]
  - **Default value**: 0

- **const char * SIDE (in)**
  - **Purpose**: to specify the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

- **const char * OPE (in)**
  - **Purpose**: to specify the optical element this command applies to. The OPE could be the primary, the secondary or the tertiary mirror.
  - **Description**: string to define the optical element (OPE), “primary”, “secondary” or “tertiary”.
  - **Unit**: unitless
  - **Range or possible value**: M1 | M2 | M3
  - **Default value**: M1

Return values

- **Result * RESULT (returned)**
**Purpose:** to evaluate if the command was accepted and successfully executed.

**Description:** Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.

**Unit:** unitless

**Range or possible value:**
- Result code = `RESCODE_FAIL`: an error occurred.
- `RESCODE_OK`: command accepted.

**Preconditions**

- The instrument must be authorized.
- The telescope must be observing.
- The OPE can be moved
- The telescope is tracking and might be guiding or controlling AO.

**After execution**

- OPE is on a new position.
- Telescope is observing.
- We don't expect the TCS to send back any parameter. The evaluation of the command’s result will let us know if the command was successfully executed by the TCS.

**Examples**

```c
....
aResult = anIIF->TipTilt(11.255, -0.815, SIDE_LEFT, M1);
...
```
7.41. Update_Guidestar

Description

Syntax

```c
Result * Update_Guidestar ( Position ** guideStars,
                            const char * SIDE )
```

Attributes

- **Position ** guideStars (in)
  - **Purpose**: to specify the list of guide stars.
  - **Description**: Position structure to define the guide star list.
  - **Unit**: unitless
  - **Range or possible value**: -
  - **Default value**: 0

- **const char * SIDE (in)
  - **Purpose**: to define the side this command applies to.
  - **Description**: string to define the side, “left”, “right” or “both”.
  - **Unit**: unitless
  - **Range or possible value**: SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH
  - **Default value**: -

Return values

- **Result * RESULT (returned)
  - **Purpose**: to evaluate if the command was successfully executed.
  - **Description**: Result structure received from the TCS with the result code, a text describing the result code and a request handle, which can be used to query for further information.
  - **Unit**: unitless
  - **Range or possible value**: Result code =
    - RESCODE_FAIL: an error occurred.
    - RESCODE_OK: command accepted.

Preconditions

- The instrument must be authorized.
- The telescope must be ready to move, to track and control the active optic system.

After execution

- ******
- We don't expect the TCS to send back any parameter. The evaluation
of the command's result (described in detail on section 8.2) will let us know if the command was successfully executed by the TCS.

Examples

....
Position *offset = new Position( 2.354, .1234, COORD_RADEC, "B2003.50", "2007",NULL, &magnitude, 3421);
Position *hotspot = new Position( 3.5, .6, COORD_SFP, "-1", &propm, &magnitude, 0);
Position *target = new Position( 3.5, .6, COORD_RADEC, "J2003.123","2007", &propm, &magntitude1, 2020);
guidestars = (Position **) malloc(3 * sizeof(Position *));
guidestars[0] = target;
guidestars[1] = offset;
guidestars[2] = (Position *) NULL;

aResult = anIIF->Update_Guidstar( guidestars , SIDE_LEFT );
....
8. Process flow. Usage

The following section gives detailed information concerning the programmatic use of the IIF classes and data types, describing a walk-through of the coding steps that should be followed by the instrument teams when use the IIF libraries to implement the processes for an entire observation cycle.

8.1. Including IIF into the project

To access the IIF’s classes through object-oriented C++ code, the instrument code developer includes the header file `IIF.h` (located inside `/usr/local/include/LBT`). The include directive for the C++ source file equivalent to the example in the above section read as follows:

```cpp
#include <LBT/IIF.h>
```

To link the IIF libraries into a C++ project, one can use the same dynamic or static library files, `libLBTIIFlib.so` or `libLBTIIFlib.a`, respectively. An example for a C++ compiler and linker call to link in the IIF libraries follows:

```bash
g++ -I/usr/local/include/LBT -Wall iiftest_c++.cpp \
-o iiftest_c++ -lLBTIIFlib
```

For further details about the procedure to compile and install the IIF libraries into the instrument control computer, see reference [2].

8.2. Creating IIF and Authorizing with TCS

The first step in using the IIF libraries is to create a new instance of the `IIF` class. When creating an IIF instance, instruments identify themselves by two pieces of information:

1. their unique name (i.e., 'LBC', or 'LUCIFER2'),
2. their focal station, which consists of
   a. the location of the instrument (i.e. 'prime', or 'direcGregorian_left'), plus
   b. the telescope side over which they intend to control (i.e., 'left', 'right', or 'both' for interferometric instruments).

The instrument control software now instantiates an IIF, in which instrument
ID and focalstation/side string must be correctly specified. Otherwise, an exception will be thrown.

<table>
<thead>
<tr>
<th>Instrument name</th>
<th>Focus and side</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUCIFER</td>
<td>bentGregorianFront both</td>
</tr>
<tr>
<td>LUCIFER1</td>
<td>bentGregorianFront left</td>
</tr>
<tr>
<td>LUCIFER2</td>
<td>bentGregorianFront right</td>
</tr>
<tr>
<td>LINC</td>
<td>bentGregorianBack left, bentGregorianBack right, bentGregorianBack both</td>
</tr>
<tr>
<td>LBTI</td>
<td>bentGregorianCenter left, bentGregorianCenter right, bentGregorianCenter both</td>
</tr>
<tr>
<td>PEPSI</td>
<td>directGregorian both</td>
</tr>
<tr>
<td>PEPSI1</td>
<td>directGregorian left</td>
</tr>
<tr>
<td>PEPSI2</td>
<td>directGregorian right</td>
</tr>
<tr>
<td>PEPSIFIBER</td>
<td>bentGregorianRearFiberFeed both</td>
</tr>
<tr>
<td>PEPSIFIBER1</td>
<td>bentGregorianRearFiberFeed left</td>
</tr>
<tr>
<td>PEPSIFIBER2</td>
<td>bentGregorianRearFiberFeed right</td>
</tr>
<tr>
<td>MODS</td>
<td>directGregorian both</td>
</tr>
<tr>
<td>MODS1</td>
<td>directGregorian left</td>
</tr>
<tr>
<td>MODS2</td>
<td>directGregorian right</td>
</tr>
<tr>
<td>LBC</td>
<td>prime both</td>
</tr>
<tr>
<td>LBCRED</td>
<td>prime right</td>
</tr>
<tr>
<td>LBCBLUE</td>
<td>prime left</td>
</tr>
<tr>
<td>MAT</td>
<td>prime left</td>
</tr>
</tbody>
</table>

Table 9.1: list of valid instrument “name – focal station” combinations.

An example of how to code this in C++ code:

```cpp
//Before any communication over the IIF, it needs to be initialized with //IDs determining the instruments focal station and its name.

IIF * anIIF;
try
{
  //pretend to be the left-side, prime focus blue-channel LBC.
  anIIF = new IIF("prime left", "LBCBLUE");
}
catch (int e)
{
  cout << "The IIF failed to initialize, consult the SYSLOG for reasons" << endl;
  exit(-1);
}
```
8.3. Command request process

In order to initiate command processing inside the TCS via the IIF, an application needs to use the IIF, Result and StatusInfo classes provided by the IIF libraries. These classes are described in detail in section 7.

After the creation of its own IIF instance (see details in section 8.1), each IIF command immediately returns a Result entity, while command processing might still be in process inside the TCS (i.e., the telescope is still slewing, while the IIF’s PresetTelescope() command call has already returned a Result instance).

Each Result instance, then contains a result code, which indicates if requesting the specified action(s) from the TCS was successfully sent to the TCS. If the result code indicates that there was a problem (i.e., it evaluates to RESCODE_FAIL as defined by the IIF library’s header files), then the programmer can find a textual indication of the reason for the failure in the Result’s reply code.

Otherwise, the Result instance’s result code evaluates to RESCODE_OK. In this successful case, another component of the result becomes meaningful, namely the command handle, which is a pointer that identifies the associated command processing inside the TCS. In the failure case above, this pointer is simply NULL, but if the command was successfully sent to the TCS, then the command handle has a value different from NULL.

Using a (non-null) command handle, the IIF’s GetCommandStatus() method can be used to inquire about the status of the respective command’s processing inside the TCS. This method returns a StatusInfo entity for a given command handle, which can henceforth be used to query the command’s processing status, an estimated time to complete, and an XML-based string representation of the command’s result once processing has finished inside the TCS. More details on this process can be found in section 8.4.

The UML activity diagram 1 shows the command request process.
8.4. Command result evaluation

Once command processing has finished on the TCS' side, the StatusInfo object that is associated with this command exhibits a status code that differs from STATE_RUNNING (See UML activity diagram 1); i.e., the command status property of the respective StatusInfo object is either STATE_SUCCESS, STATE_CANCELED, STATE_FAILURE, or STATE_WRONGHANDLE (as defined in file StatusInfo.h).

The programmer has two options in waiting for such finishing of the command processing: “synchronous” or “asynchronous” IIF command processing. In “synchronous” mode, the programmer will periodically execute the IIF’s
GetCommandStatus() method, in order to receive a new StatusInfo object, with an updated command status property. The “asynchronous” mode means that the programmer can choose to halt the program thread’s execution until the TCS has finished the command processing (“busy wait”). To achieve this, the IIF user simply calls the block() method on the command handle that is associated with the requested command.

Table below briefly describes the meaning of the status code values mentioned above.

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_RUNNING</td>
<td>The command is still being executed by the TCS</td>
</tr>
<tr>
<td>STATE_SUCCESS</td>
<td>The TCS has successfully finished processing the associated command; note that the actual actions requested to be performed by the telescope can still have failed.</td>
</tr>
<tr>
<td>STATE_CANCELED</td>
<td>The TCS command’s processing has been previously cancelled.</td>
</tr>
<tr>
<td>STATE_FAILURE</td>
<td>An error has occurred during execution of the associated command by the TCS.</td>
</tr>
<tr>
<td>STATE_WRONGHANDLE</td>
<td>There is no valid TCS-side command associated with the StatusInfo object’s command handle.</td>
</tr>
</tbody>
</table>

Table 9.1: Status codes in StatusInfo.

If the status code is either STATE_SUCCESS, STATE_CANCELED, or STATE_FAILURE, then it is worth the IIF users analyze the command’s result that was produced and returned by the TCS. In order to process the returned result, the programmer has to access the StatusInfo object’s command result property. The string inside the command result is an XML-string encoded CommandReturn object, as provided by the TCS’ Common Software library. This section will explain how to access these objects’ data as it will be most commonly accessed by the IIF user. For further details on the Common Software library and the CommandReturn class, please consult the respective sections of the LBT TCS Common Software documentation.

First, the IIF user generates a CommandReturn object. On this object, it calls the deseralize() method with the above mentioned StatusInfo object’s command result. As a consequence, the CommandReturn object is ready to access the associated TCS command’s results in a more convenient way. In order to decide if the actual actions inside the telescope control system,

NOTE: The command result will be STATE_NORESULT, if the user tries to retrieve the command result on a StatusInfo object whose command handle is invalid, or NULL.
which were triggered by firing off the associated TCS command, have been performed without any problems, one has to check the CommandReturn’s isError() method. If it returns false, then the processes inside the TCS showed no glitches, otherwise, there were problems or failures.

The actual results (in the success case), or respectively indications about errors (in the error case), are generally retrieved via CommandReturn’s getResultCount() and getResultDescription(int n) methods. The first method returns the number of results that can be queried via the second method, while the second method returns an STL string representation of the TCS command’s result, or error message respectively.

An exception to the rule that all results are accessed via getResultDescription() is made for the LBC IIF’s GetRotatorTrajectory() Here, if the command has finished successfully, the resulting time-angle pairs are accessed via CommandReturn’s getArgument() method. Error indications (i.e., if isError() returned true) can still be retrieved through getResultDescription().

8.5. De-authorizing the instruments. Destroying IIF instance

In order to let the TCS know that an instrument wants to surrender control of one or more telescope sides, the instrument control code must execute the IIF’s Deauthorize() command. The IIF user has to specify the telescope side(s) that it wants to release, i.e., SIDE_LEFT, SIDE_RIGHT, or SIDE_BOTH.

The value returned to the instrument by Deauthorize() is a Result pointer, whose handle can be queried for the result string to see if the deregistration was successful. Note that the result of calling GetCommandStatus() on the associated StatusInfo instance will return SUCCESS, even if the deregistration failed, as the command itself was successfully executed by the TCS. To check if the desired result was yield, the result string (accessed by StatusInfo’s GetCommandResult() method) should be checked for the string DEREGISTERED. The following notes on internal workings of the IIF code will explain more about the deregistration process. Inside of IIF::Deauthorize(), the Result instance returned by IIFDeauthorize::execute() and IIF::doExecute() is temporarily stored for later return. IIF::Deauthorize() only returns after it has updated an attribute that is internal to the IIF instance. This attribute (named __authorized) stores for each IIF object for which telescope side(s) the associated instrument was authorized. The following table shows how calling Deauthorize() affect the value of the __authorized attribute:
Deauthorize() | _authorized before | resulting _authorized |
--- | --- | --- |
SIDE_LEFT | SIDE_RIGHT, SIDE_BOTH | SIDE_RIGHT |
| SIDE_LEFT, SIDE_NONE | SIDE_NONE |
SIDE_RIGHT | SIDE_LEFT, SIDE_BOTH | SIDE_LEFT |
| SIDE_RIGHT, SIDE_NONE | SIDE_NONE |
SIDE_BOTH | SIDE_LEFT | SIDE_LEFT |
| SIDE_RIGHT | SIDE_RIGHT |
| SIDE_BOTH, SIDE_NONE | SIDE_NONE |
SIDE_NONE | any value | SIDE_NONE |

Please note that the IIF::Deauthorize() routine does not check the result string of the command handle returned by IIFDeauthorize::execute() for "DEREGISTERED" to decide how to set the new value of _authorized. The decision is solely based upon look-up in the above state table. The reason for this is that the instrument programmers should be empowered to run Deauthorize() - as opposed to Authorize() - in an asynchronous fashion, if they choses to do so.

### 8.6. Full example

The following code shows the entire process in order to communicate with the TCS, initializing IIF, sending commands over the TCS<->Instrument interface and printing the results and error messages to the standard output (stdout). You will find more examples ready to compile and use in the source code of the IIF (LBTO/TCS/IIF/Instrument/Examples).

```cpp
#include <LBT/IIF.h>
#include <LBT/CommandReturn.h>
#include <LBT/SysLog.h>

int main(int argc, char *argv[]) {
    // Before any communication over the IIF, it needs to be initialized with
    // IDs determining the instruments focal station and its name.
    IIF * anIIF;
    try {
        //pretend to be the left-side, prime focus blue-channel LBC
        anIIF = new IIF("prime left", "LBCBLUE");
    }
    catch (int e) {
        cout << "The IIF failed to initialize, consult the SYSLOG" << endl;
        exit(-1);
    }
} // end main
```
// Permission for commanding the respective telescope sides is requested, // and - upon failure to achieve authorization- the program is exited
if (!anIIF->Authorize())
{
    cout << "Authorize() was not successful ... terminating test program.";
    exit(-1);
}
cout << "Authorize() successful" << endl;

// Declare the result object reference needed as a container for the // information returned via the IIF.
Result* aResult;

// Initialize some objects to use as example
MagnitudeType magnitude1 = {-27.0, "IV",1.5,"B-V"};
MagnitudeType magnitude2 = {20.0, "R",2.0,"U-V"};
ProperMotionType propm1= {0.123, 0.234};
ProperMotionType propm2= {0.567, 0.789};
Position *target = new Position(3.5,1.6,COORD_RADEC,"J2003.123","2007", &propm1, &magnitude1, 2020);
Position *offset;
Position *hotspot = new Position(3.5, .6, COORD_SFP, "-1", "2007", &propm2, &magnitude1, 0);
Position **guidestars;

// Get some Parameters from the data dictionary
cout << "GetMultiParameter" << endl;
MultiDDEntry multiEntries;
multiEntries.PushEntry("csq.authorizedInstrument.left");
multiEntries.PushEntry("csq.authorizedCount.left");
multiEntries.PushEntry("csq.authorizedFocalStation.left");
multiEntries.PushEntry("csq.standbyLevel.left");
multiEntries.PushEntry("pmc.side[1].temperature");
aResult = anIIF->GetMultiParameter(multiEntries);

if (aResult->GetresultCode() != RESCODE_FAIL)
{
    aResult->GetcommandHandle()->block(); /* wait for TCS */
    StatusInfo* statusInf =
    anIIF->GetCommandStatus(aResult->GetcommandHandle());
cout << "GETMULTIPARAMETER command status: " <<
    statusInf->GetCommandStatus() << endl;
    CommandReturn cmdRet;
    cmdRet.deserialize(statusInf->GetCommandResult());
    if(cmdRet.isError())
        cout << "GETMULTIPARAMETER command result status: Error" << endl;
    else
        cout << "GETMULTIPARAMETER command result status: OK" << endl;
}
cout << "GETMULTIPARAMETER command result: ";
if(cmdRet.isError()) cout << "FAILED" << endl;
else cout << "SUCCESS" << endl;
for(int i=1; i<=cmdRet.getResultCount(); i++)
    cout << cmdRet.getResultDescription(i) << endl;
delete statusInf;
} else {
    cout << "GetMultiParameter execution failed: " <<
        aResult->GetresultString() << endl;
}
delete aResult; /* free memory for CSQHandle */

/* code snippet */

offset =
new Position(2.34, .124, COORD_RADEC, "B2003.50", "2007", NULL, &magnitude2,3421);

guidestars = (Position **) malloc(4 * sizeof(Position *));
guidestars[0] = target;
guidestars[1] = offset;
guidestars[2] = hotspot;
guidestars[3] = (Position *) NULL;

// Call PresetTelescope()
 aResult =
anIIF->PresetTelescope(1.0, "position", target, guidestars, "active", "left");

if (aResult->GetresultCode() != RESCODE_FAIL)
{
    aResult->GetcommandHandle()->block(); /* wait for TCS */
    StatusInfo* statusInf =
anIIF->GetCommandStatus(aResult->GetcommandHandle());
    cout << "PRESET_TELESCOPE command status: " <<
        statusInf->GetCommandStatus() << endl;
    CommandReturn cmdRet;
    cmdRet.deserialize(statusInf->GetCommandResult());
    if(cmdRet.isError())
        cout << "PRESET_TELESCOPE command result status: Error" << endl;
    else
        cout << "PRESET_TELESCOPE command result status: OK" << endl;
    cout << "PRESET_TELESCOPE command result: ";
    for(int i=1; i<=cmdRet.getResultCount(); i++)
        cout << cmdRet.getResultDescription(i) << endl;
    delete statusInf;
}
else{
    cout << "PresetTelescope execution failed: " <<
        aResult->GetresultString() << endl;
delete aResult; /* free memory for CSQHandle */
delete offset;
delete guidestars;

//----------------- Tell telescope that the instrument wants to idle ----------------
cout << "Deauthorize" << endl;

// Call Deauthorize(), this indicating that the observation has finished.

aResult = anIIF->Deauthorize("left");

if (aResult->GetresultCode() != RESCODE_FAIL)
{
aResult->GetcommandHandle()->block(); /* wait for TCS */

StatusInfo* statusInf =
anIIF->GetCommandStatus(aResult->GetcommandHandle());

cout << "DEAUTHORIZE command status: "
    << statusInf->GetCommandStatus() << endl;

cout << "DEAUTHORIZE command result: "
    << statusInf->GetCommandResult() << endl;

delete statusInf;
}
else
{
    cout << "Deauthorize execution failed: "
        << aResult->GetresultString() << endl;
}

delete aResult; /* free memory for CSQHandle */

//--------------------------------- Lead out ------------------------------------
// Clean up after yourself when done

delete anIIF;

// Return an exit code indicating that everything went OK.
    return 0;
9. References


Appendix A: Global definitions

File: LBT/IIFGlobal.h

Definition of the default focal station ID
#define FS_NONE "none"

Definition of the focal station ID for the prime focus
#define FS_PRIME "prime"

Definition of the focal station ID for the direct Gregorian focus
#define FS_DIRECTGREGORIAN "directGregorian"

Definition of the focal station ID for the bent Gregorian focus (back)
#define FS_BENTGREGBACK "bentGregorianBack"

Definition of the focal station ID for the bent Gregorian focus (front)
define FS_BENTGREGFRONT "bentGregorianFront"

Definition of the focal station ID for the bent Gregorian focus (center)
#define FS_BENTGREGCENTER "bentGregorianCenter"

Definition of the focal station ID for the fiber feed in the bent Gregorian focus
#define FS_BENTGREGREARFIBER "bentGregorianRearFiberFeed"

Definition of the default instrument ID
#define IS_NONE "NONE"

Definition of the instrument ID for the first LUCIFER
#define IS_LUCIFER1 "LUCIFER1"

Definition of the instrument ID for the second LUCIFER
#define IS_LUCIFER2 "LUCIFER2"

Definition of the instrument ID for a virtual LUCIFER instrument, controlling both sides
#define IS_LUCIFER "LUCIFER"

Definition of the instrument ID for LINC/NIRVANA
#define IS_LINC "LINC"

Definition of the instrument ID for LBTI
#define IS_LBTI "LBTI"

Definition of the instrument ID for the first PEPSI in direct Gregorian
#define IS_PEPSI1 "PEPSI1"

Definition of the instrument ID for the second PEPSI in direct Gregorian
#define IS_PEPSI2 "PEPSI2"

Definition of the instrument ID for a virtual PEPSI instrument, controlling both sides
#define IS_PEPSI "PEPSI"
Definition of the instrument ID for the first PEPSI in the bent Gregorian
#define IS_PEPSIFIBER1 "PEPSIFIBER1"

Definition of the instrument ID for the second PEPSI in the bent Gregorian
#define IS_PEPSIFIBER2 "PEPSIFIBER2"

Definition of the instrument ID for a virtual PEPSIFIBER instrument, controlling both sides
#define IS_PEPSIFIBER "PEPSIFIBER"

Definition of the instrument ID for the first MODS
#define IS_MODS1 "MODS1"

Definition of the instrument ID for a combined MODS
#define IS_MODS2 "MODS2"

Definition of the instrument ID for a virtual MODS instrument, controlling both sides
#define IS_MODS "MODS"

Definition of the instrument ID for red-channel LBC
#define IS_LBCRED "LBCRED"

Definition of the instrument ID for blue-channel LBC
#define IS_LBCBLUE "LBCBLUE"

Definition of the instrument ID for a virtual LBC instrument, controlling both sides
#define IS_LBC "LBC"

Definition of the instrument ID for MAT
#define IS_MAT "MAT"

Definition of the strings specifying the left telescope side
#define SIDE_LEFT "left"

Definition of the strings specifying the right telescope side
#define SIDE_RIGHT "right"

Definition of the strings specifying both telescope sides
#define SIDE_BOTH "both"

Definition of the strings specifying no association with a telescope side
#define SIDE_NONE "noSide"

Definition of the strings specifying instrument rotator's position mode
#define TRACK_PSTN "position"

Definition of the strings specifying instrument rotator's vertical mode
#define TRACK_VERT "vertical"

Definition of the strings specifying instrument rotator's idle mode
#define TRACK_IDLE "idle"

Definition of the strings specifying the telescope's passive mode of operation
#define MODE_PSV "passive"
Definition of the strings specifying the telescope's guided mode of operation
#define MODE_GUD "guided"

Definition of the strings specifying the telescope's adaptive mode of operation
#define MODE_ADP "adaptive"

Definition of the strings specifying the telescope's active mode of operation
#define MODE_ACT "active"

Definition of the strings specifying the telescope's interferometric mode of operation
#define MODE_INF "interferometric"

String constant that is returned by a call in IIF::Authorize, in case the IIF instance has successfully registered with the CSQ
#define INSTANCE_AUTHORIZED "AUTHORIZED"
<table>
<thead>
<tr>
<th>LBT PROJECT</th>
<th>Doc. No. : xxxxxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument-Telescope Control Document</td>
<td>Issue : b</td>
</tr>
<tr>
<td></td>
<td>Date :</td>
</tr>
</tbody>
</table>

---oOo --
Doc_info_start
Title: Instrument-Telescope Control Document
Document Type: Technical Manual
Source: Steward Observatory
Issued by: Jose Borelli
Date of Issue: 15-12-2006
Revised by:
Date of Revision:
Checked by:
Date of Check:
Accepted by:
Date of Acceptance:
Released by:
Date of Release:
File Type: Open office writer
Local Name:
Category: Telescope
Sub-Category: Telescope Controls
Assembly: Telescope Controls Software
Sub-Assembly:
Part Name:
CAN Designation:
Revision: a
Doc_info_end