Instrument Interface - IIF

LBTO

IIF API

Commands that have significantly changed in the last months

Jose L. Borelli

MPIA

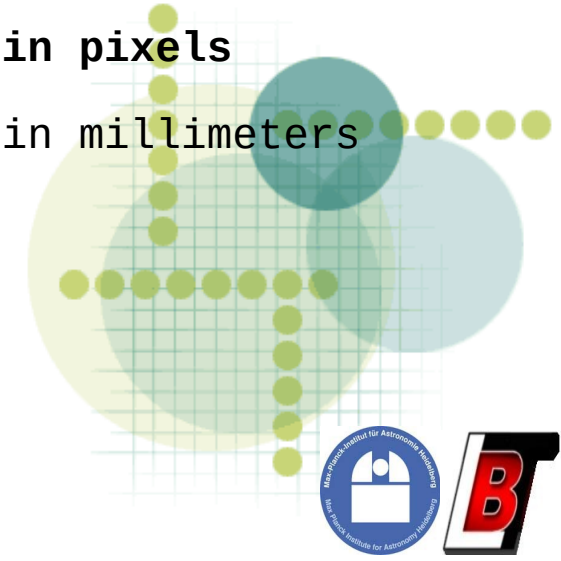
Overview

- New architecture aspects
  - Enumerated type variables vs. string variables
  - New classes and structures. Implications
- IIF commands that have significantly changed in the last months
- New IIF commands added to the set

Enumerated type variables

- Coordinates systems

```
CoordType
{
  COORD_RADEC_SKY,    * equatorial coordinates on the sky
  COORD_RADEC_FOCAL,  * equatorial coordinates on the focal plane
  COORD_ALTAZ,        * the ALT/AZ coordinates
  COORD_FOCAL_PIX,    * instrument focal plane coordinates in pixels
  COORD_FOCAL_MM      * instrument focal plane coordinates in millimeters
};
```



Enumerated type variables

- Operational modes of the telescope.

```
modeType
{
    MODE_STATIC,          *   Slew the Teles. to the requested location on sky, and
                              stop.
    MODE_TRACK,           *   Sends the Teles. to the coordinates, enters open-loop
                              tracking.
    MODE_GUIDE,           *   Trakcing and guiding are in operation. Requires a guide
                              star from the instrument.
    MODE_ACTIVE,          *   Tracking, guiding, and active optics are enganged. Ends
                              closed-loop in both XY guiding and wavefront sensing.
    MODE_ADAPTIVE,        *   Tracking, guiding, and adaptive optics are
                              enganged. Big-W must end up closed-loop at high speed.
    MODE_INTERFEROMETRIC * Closed-loop with Big-W on both sides. Implies
                              synchronization between the 2 sides.
};
```

Enumerated type variables

- Different Rotator Modes of the instruments

```
rotatorType (previously trackmode)

{
    ROTATE_PAR,         *   Represents the parallactic angle, which is
                            vertical with respect to the horizon.

    ROTATOR_PSTN,       *   Relative to the North

    ROTATOR_NATIVE,     *   To use the rotator's native reference
                            frame.

    ROTATOR_GRAV,       *   To use the gravitational angle.

    ROTATOR_IDLE,       *   For those instruments that will not use
                            the rotator at all.

};
```

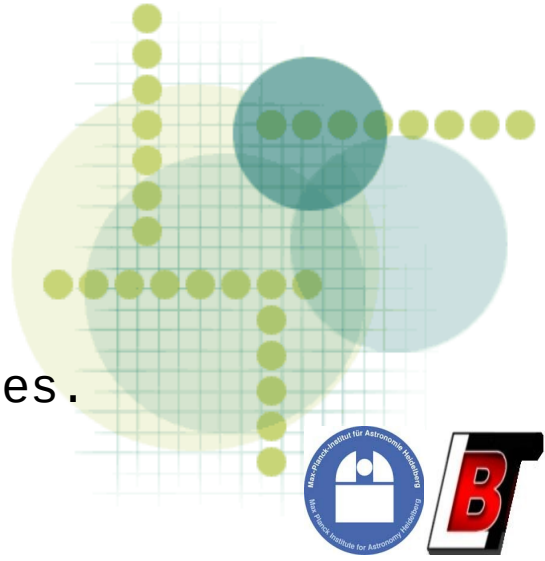
Enumerated type variables

- Others enumerated variables used as arguments in many of the IIF commands
  - `opeType`, to define the optical elements.
  - `moveType`, to define the type of movement, relative or absolute.
  - `equinoxType`, to chose between the equatorial coordinate system based on the mean dynamical equator and equinox of the J2000 epoch, and ICRS (International Celestial Reference System).
  - `AOmodeType`, to represent the different AO operational modes.
  - `filterType`, representing the different filters, U, V, etc.
  - `colorType`, to represent the different color types of the object, U-B, B-V, etc.

The Offset class

- The Offset class, used by the commands `PresetTelescope`, `OffsetPointing` and `OffsetGuiding`, is nothing else that two delta terms.
- These are relative deltas, referenced to the last coordinates provided.
- The constructor has the following parameters:

  - `double coord1` * to define the offset in RA, ALT or xi.
  - `double coord2` * to define the offset in DEC, AZ, or eta.
  - `coordType system` * the coordinate system of reference for the coordinates.



The Offset class

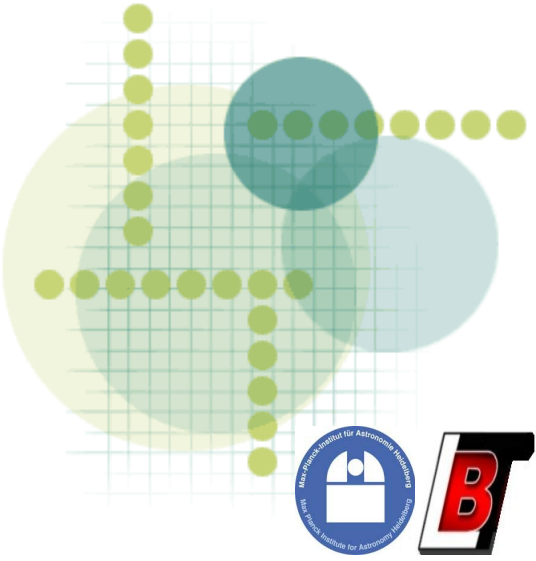
Units and ranges

Parameters	Coord. System	Unit	Range
double coord1	COORD_RADEC_SKY	radians	-PI to PI
	COORD_RADEC_FOCAL	radians	-PI to PI
	COORD_ALTAZ	radians	-PI/2 to PI/2
	COORD_FOCAL_PIX	pixels	def. by the instrument
	COORD_FOCAL_MM	millimiters	def. by the instrument
double coord2	COORD_RADEC_SKY	radians	-PI to PI
	COORD_RADEC_FOCAL	radians	-PI to PI
	COORD_ALTAZ	radians	-PI to PI
	COORD_FOCAL_PIX	pixels	def. by the instrument
	COORD_FOCAL_MM	millimiters	def. by the instrument

The Hotspot class

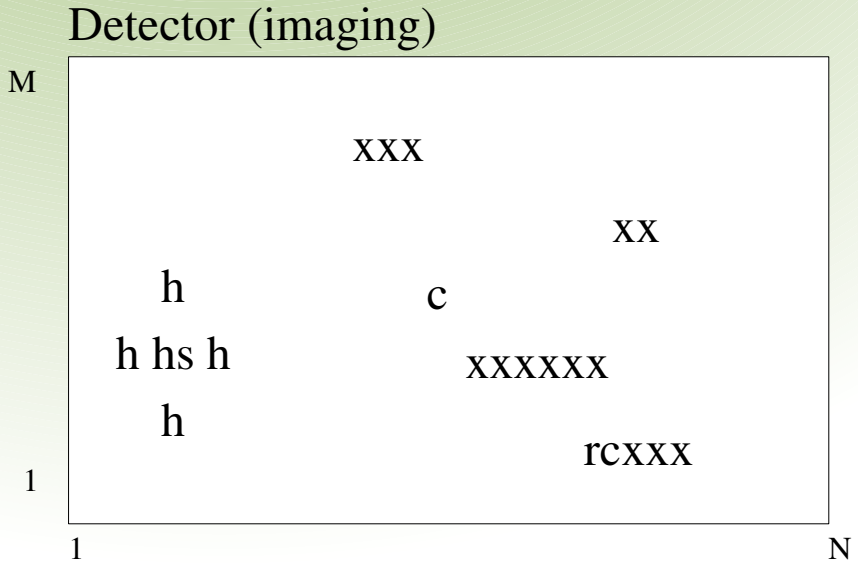
- The Hotspot is given in instrument focal plane coordinates. These are an absolute X and Y values specific to the detector.
- The units are in pixels and the ranges are defined by the instruments.
- The constructor has the following parameters:

  - `double coord1` * to define the X coordinate of the hotspot
  - `double coord2` * to define the Y coordinate of the hotspot



The Hotspot class

- Example: Detector during observation. The hotspot (hs) is the best position on the detector.



c = detector center

rc = rotation center

hs = hotspot(x=3,y=3)

h = dither positions around the hs

x = bad pixels

The Position class

- This class defines positions and all astronomically relevant information about the science target to be observed and the guide-stars.
- Position arguments:

- `double coord1` * defines the first coordinate of the specified system in RA, ALT or xi.
- `double coord2` * defines the second coordinate of the specified system in DEC, AZ or eta.
- **`coordType system`** * defines the coordinate system of reference for the coordinates.

The Position class

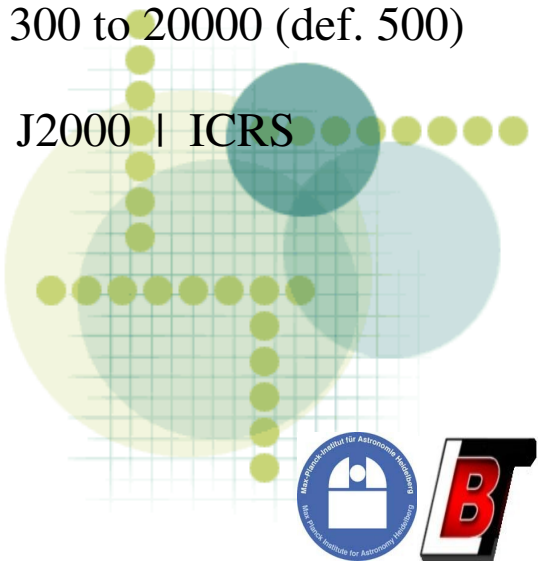
- Position arguments:

- **`equinoxType equinox`** * defines the value of the equinox for purpose of precession
- **`double epoch`** * Represents the value used in conjunction with proper motion values to reference the coordinates to the equinox
- `ProperMotionType *propmotion` (optional) * specifies the potential proper motion of the celestial object described by the position
- `MagnitudeType *magnitude` (optional) * specifies the apparent magnitude and filter of the celestial object
- `unsigned int wavelength` (optional) * computes the object's atmospheric refraction correction

The Position class

Units and ranges

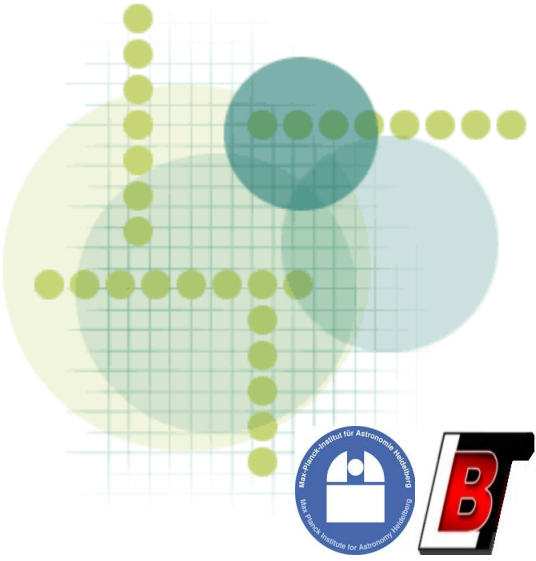
Parameters	Coord. System (coordType)	Unit	Range
double coord1	COORD_RADEC_SKY	radians	0.0 to 2PI
	COORD_ALTAZ	radians	0.0 to PI/2
double coord2	COORD_RADEC_SKY	radians	-1.0 to 2PI
	COORD_ALTAZ	radians	-PI/2 to 5PI/2
Unsigned int wavelenght	-	nanometer	300 to 20000 (def. 500)
equinoxType equinox	-	-	J2000 | ICRS



PresetTelescope

- This command slew the telescope into position in order to begin an observation cycle.

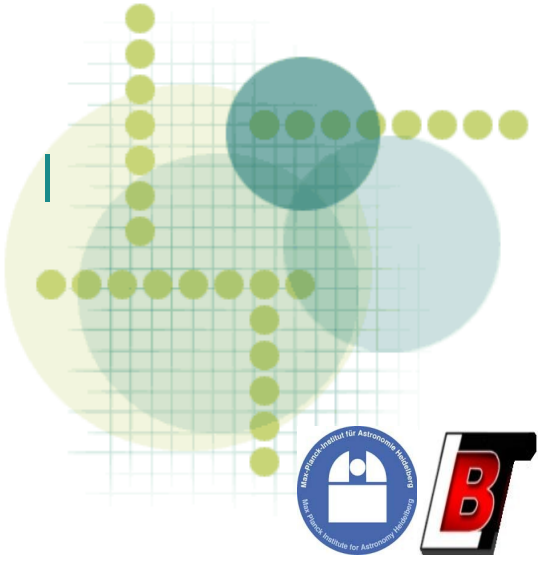
```
PresetTelescope (  double ROTANGLE,
                   rotatorType ROTATORMODE,
                   Position* TARGET,
                   [Position** GUIDESTARS],
                   modeType MODE,
                   const char* SIDE,
                   [Offset* OFFSET],
                   [Hotspot* HOTSPOT],
                   [bool WRAPFLAG] )
```



PresetTelescope

- `double ROTANGLE`
  - Purpose: to specify the value for the initial rotator angle.
  - Unit: radian
  - Range or possible values: `-2PI to 2PI`
  - Default value: `0`
- **`rotatorType ROTATORMODE`**
  - Purpose: to specify the rotator mode of the instrument.
  - Possible values:

    `ROTATOR_PAR | ROTATOR_PSTN | ROTATOR_NATIVE | ROTATOR_GRAV | ROTATOR_IDLE`
  - Default value: -



PresetTelescope

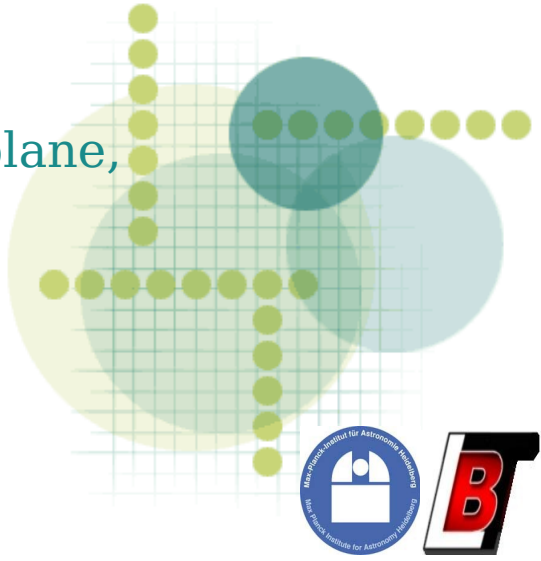
- `Position * TARGET`
  - Purpose: To specify all characteristics of the target and its location according to the coordinate system chosen.
- `Position ** GUIDESTARS (optional)`
  - Purpose: To specify all characteristics of the guide stars and their locations according to the coordinate system chosen.
- **`modeType MODE`**
  - Purpose: to specify the operating mode of the telescope.
  - Possible values:

```
MODE_STATIC | MODE_TRACK | MODE_GUIDE |
MODE_ACTIVE | MODE_ADAPTIVE| MODE_INTERFEROMETRIC
```

PresetTelescope

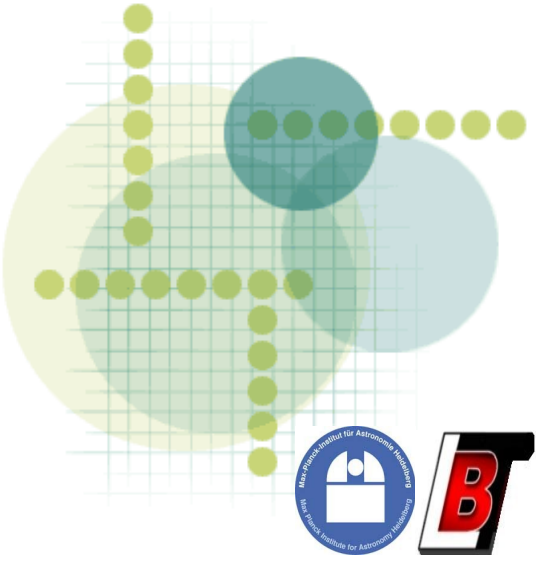
- `const char * SIDE`
  - Purpose: to specify the side of the telescope
  - Range or possible values: `SIDE_LEFT | SIDE_RIGHT | SIDE_BOTH`
- **`Offset * OFFSET (optional)`**
  - Purpose: to specify the offset for the target in RA and DEC, ALT and AZ, or SFP coordinates.
  - Default value: NULL
- **`Hotspot * HOTSPOT (optional)`**
  - Purpose: to specify the reference position in the focal plane,

    by default, the center of the focal plane.
  - Default value: NULL



PresetTelescope

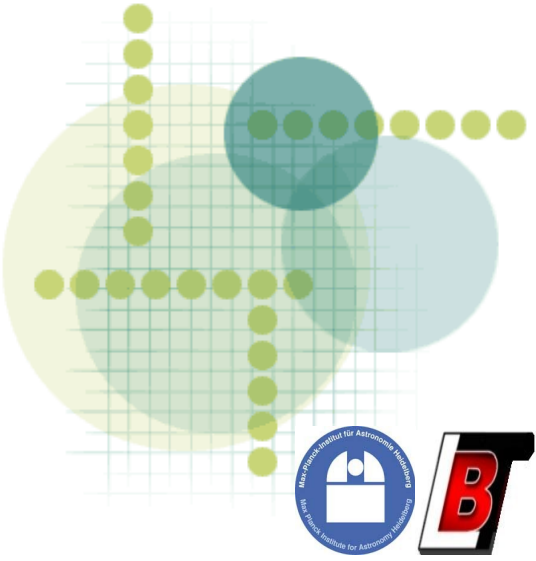
- **`bool WRAPERFLAG (optional)`**
  - Purpose: This is a "maximize-time-on-target" flag, where `true` means to choose the path that selects the cable wrap that will provide the longest possible observing time on the object. `false` means to move from the present position to a new object by the shortest path possible.
  - Range or possible values: `true | false`
  - Default value: `false`



OffsetPointing

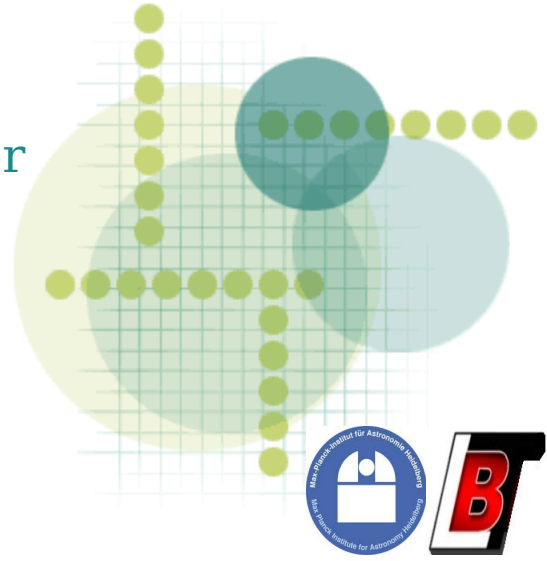
- OffsetPointing moves the telescope a small distance, setting the value of the telescope pointing coordinates to the new position.
- It uses the existing target information and the setup declared in the last `PresetTelescope`.

```
OffsetPointing (   double ROTANGLE,
                   Offset * OFFSET,
                   opeType OPE,
                   bool NEWPOSITION,
                   moveType MOVETYPE,
                   const char * SIDE  )
```



OffsetPointing

- `double ROTANGLE`
  - Purpose: to specify the value in radians of the rotation angle.
  - Unit: radians
  - Range: `-2PI to 2PI`
- **`Offset * OFFSET`**
  - Purpose: to specify the offset from the position specified by the `MOVETYPE` argument.
- **`moveType MOVETYPE`**
  - Purpose: to determine if the movements are relatives or absolutes.
  - Possible values: `MV_REL | MV_ABS`
  - Default value: `MV_REL`



OffsetPointing

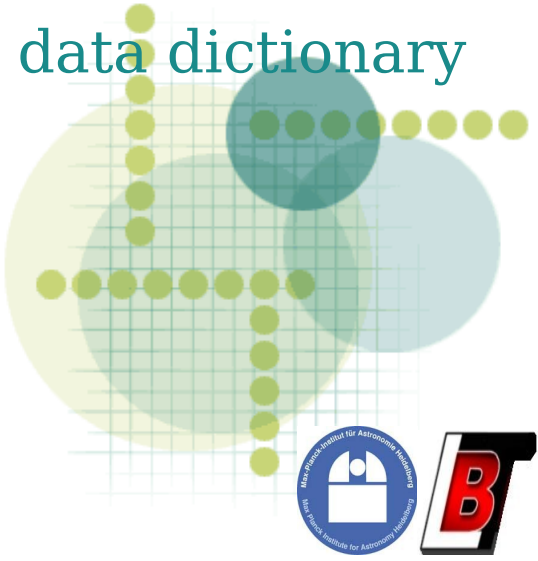
- **`OpeType OPE`**
  - Purpose: to specify the optical element to be moved.
  - Possible values:

    `MOUNT | M1 | M2 | M3 | HEXAPOD | DEFAULT`
  - Default value: `DEFAULT`, which allows the pointing kernel to choose the action which should be taken based upon its internal logic.
- **`bool NEWPOSITION`**
  - Purpose: to determine if the target position should be changed or not.
  - Possible values: `true | false`, where true means to move the guide stage but do not change targetRA and DEC; false means to update RA and DEC (dither).
  - Default value: `false`
- `Const char * SIDE`

GetMultiParameter

- This command is used to read a block of entries from the data dictionary in one shot.
- Argument: a `MultiDDEntry` object, which is the list of parameters to be read by the command from the data dictionary.
- This object must be populated with valid data dictionary entries, using the internal method `PushEntry(entry_name);`
- It returns a list with the values of the requested data dictionary entries.

```
GetMultiParameter( MultiDDEntry ENTRIES )
```



SetMultiParameter

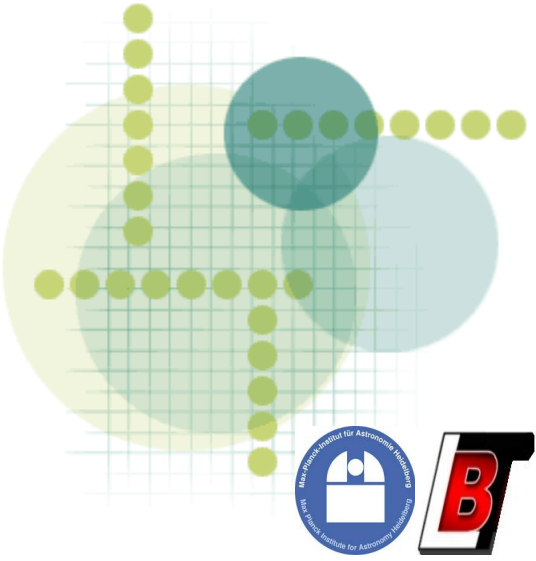
- `SetMultiParameter` sets the values of the specified data dictionary entries on the TCS in one shot
- The instrument only has permission to modify its own predefined entries.
- Argument: `MultiDDEntry` object, which must be populated with string pairs, the local data dictionary name, and the value (`PushEntry` and `PushValue`).
- The CSQ subsystem will generate the fully qualified data dictionary name as "`csq.<InstrumentID>.entry_name`".

```
SetMultiParameter( MultiDDEntry ENTRIES )
```

Get/SetMultiParameter

- Example:

```
...
MultiDDEntry DDEntries;
DDEntries.PushEntry("pmc.side[0].elevationAngle");
DDEntries.PushEntry("pmc.side[1].elevationAngle");
aResult = anIIF->GetMultiParameter(DDEntries);
...
DDEntries.Clear();
DDEntries.PushEntry("side[0].cooler" , "ON");
DDEntries.PushEntry("side[1].cooler" , "OFF");
aResult = anIIF->SetMultiParameter(DDEntries);
...
```



Target acquisition - Alignment

- **MoveXY** (double XMOTION, double YMOTION,
  opeType OPE, const char * SIDE)
  - moves an OPE in X or Y direction (micrometers), relative to the current position.
  - In closed-loop mode with w/W, this may require offsetting the relevant stage/s as well to maintain lock on the specified reference star.
  - Possible OPE: `M1 | M2`

- **MoveFocus** (double ABSPOS, **opeType OPE**,
  const char * SIDE)
  - MoveFocus moves an optical element to a new absolute position z to adjust or to define a new focus position.
  - Any focus move in closed-loop mode must be accompanied by the corresponding offset of the w/W stage along the focus direction.
  - Possible OPE: `M1 | M2 | M3(piston) | M1M2`

Target acquisition - Alignment

- **StepFocus** (double RELPOS, **opeType OPE**,
  const char * SIDE)
  - moves the respective focus position, by moving the OPE a given distance in the direction of of the telescope's Z axis
  - Possible OPE: `M1 | M2 | M3 | M1M2 (scale-preserving focus)`

- **TipTilt** (double XROTATION, double YROTATION,
  opeType OPE, const char * SIDE)
  - The TipTilt command moves an OPE in tip or tilt direction, relative to the current position
  - For OPE `M3`, `XROTATION` is Tip, and `YROTATION` is Tilt, which are defined local to the M3 mirror
  - Positive tip will move the beam up
  - Positive tilt will move the beam toward the front of the telescope, regardless of side

Target acquisition - Alignment

- **Move** (double X, double Y, double Z,
  double RX, double RY, double RZ,
  int D_FLAG, moveType MOVE_TYPE,
  opeType OPE, double TIME,
  const char * SIDE)
  - X, Y, and Z represent the naked focal plane movements. For OPE M3 (M3 piston), X and Y are ignored.
  - RX, RY, and RZ represent the the naked focal plane rotation. For OPE M3, RX is M3 Tip, RY is M3 Tilt.
  - MOVE_TYPE will determine if they are absolute or relative values.
  - Possible OPE: M1 | M2 | M3 | M1M2 | M1M3 | M2M3 | M1M2M3 | DEFAULT
  - D_FLAG is a 6 bits flag with a bit for each of the preceding 6 variables. Bit 0 enables X, bit 1 enables Y, bit 2 enables Z, and so on.
  - TIME: the lookahead time (sec) for the collimation correction

Target acquisition - Alignment

- **RotateCommon** (double X, double Y, double Z,
  double ANGLE, double DIRECTION,
  const char * SIDE)
  - Rotates M1 and M2 around a common point. The movement is relative. Initial positions for the mirror are depending on the focal station; must be defined in the collimation model.
  - X, Y, and Z represent the position of the point the mirrors rotate around. The coordinate zero is TBD.
- **RotateZ** (double ANGLE, moveType MOVETYPE,
  const char * SIDE)
  - Rotates M3 to adjust the incoming beam angle for the instrument.
  - A relative rotation is incremental: it adds to the current position.
  - An absolute rotation is with respect to the focal station position maintained by the OSS. An absolute rotation of zero will go back to the default focal station position.

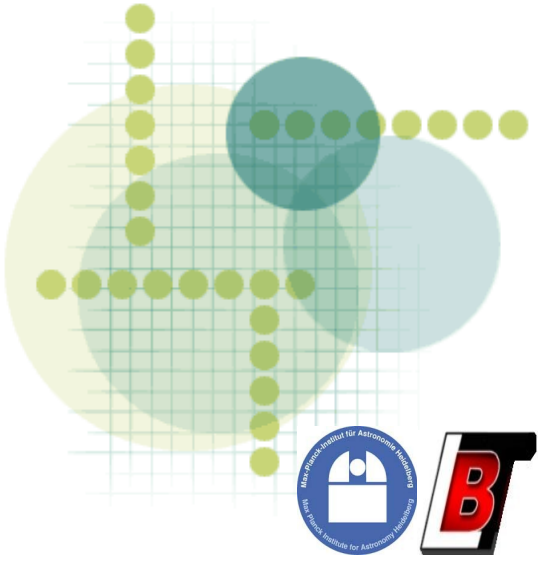
Target acquisition - Alignment

- **TelescopeMove** (double MX, double MY, double MZ,
  moveType MOVETYPE, const char * SIDE)
  - moves the entire optics on a single side as a rigid body
  - MX, MY, and MZ represent the translation values in X, Y, and Z

- **TelescopeRotate** (rotcenterType ROTCENTER,
  double RX, double RY, double RZ,
  moveType MOVETYPE,
  const char * SIDE)
  - Rotates the entire optics on a single side as a rigid body
  - RX, RY, and RZ represent the rotation angle in X, Y, and Z
  - ROTCENTER: M1 | M2 | M3 | FS_PRIME | FS_DIRECTGREGORIAN | ROT_CENTER_POS

Target acquisition - Alignment

- **`TelescopeScale`** `( double SCALE , const char * SIDE )`
  - Adjusts the overall plate scale of the telescope side as delivered to the instrument. The plate scale will be changed and the side will remain in focus.
  - SCALE ranges: `-2.5E-4 to 2.5E-4`
  - SCALE unit: TBD.



AOS

- The Adaptive Optics Subsystem (AOS) provides all the functions needed for interaction between the LBT Adaptive Optics system and the rest of the telescope, including instruments.

- TCS-IIF provides a set of commands, that correspond exactly to the AOS commands, in order to handle this subsystem.

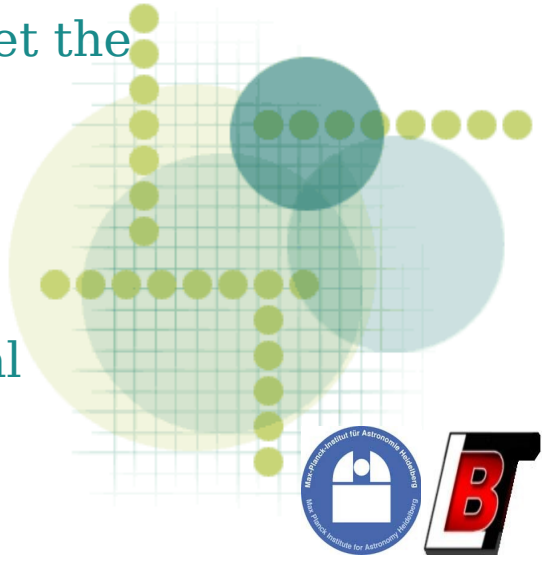
- At the moment, they are only prototypes. TCS and AOS teams need to work on the details

AO commands

- `AOPreset()`
  - is issued in a AOS observation service status in order to prepare the AO system for an observation in adaptive mode.
- `AOAcquireRef()`
  - issued after a `AOPreset`, 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.
- `AORefine()`
  - is used to support the ICE-AO operating mode. It maybe used to request the AOS to modify the value of some loop parameter before closing the loop.

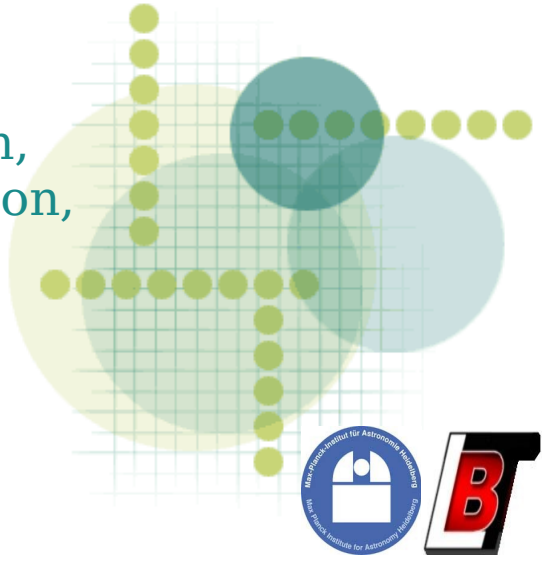
AO commands

- `AOStar()`
  - This command is used to request the closing of the AO loop.
- `AOOffsetXY()`
  - is issued in AOS observation service status in order to offset the pointing of the AOS.
  - It is meaningful only in closed loop mode.
- `AOOffsetZ()`
  - Used in AOS observation service status in order to offset the focus of the AOS.
  - It is meaningful only in closed loop mode.
- `AOCorrectModes()`
  - used in AOS observation service status to apply a modal correction on the mirror shape.



AO commands

- `AOStop()`
  - used to stop the current operation. After this command any setting defined by a previous `AOPreset` is canceled.
- `AOPause()`
  - This command is issued to temporarily suspend the current AO operation.
- `AOResume()`
  - This command resumes suspended operation after a `AOPause`.
- `AOUserPanic()`
  - This command is issued whenever some TCS subsystem, including an instrument, detects any dangerous condition, and decides to perform a fast shutdown.



Rotator commands

- `RotSetRotator ( bool ENABLE, const char * SIDE )`
  - This command is issued to enable or disable a rotator.
  - “Enable” means to turn the rotator on and make it ready to respond to commands.
  - This command is specifically designed to be used by an instrument that is not the “authorized” instrument. However, an authorized instrument can invoke this command as necessary.
- `RotAdjustPosition ( double DELTA_ROTANGLE, rotatorType ROTATORMODE, const char * SIDE )`
  - Allows fine adjustments to the current rotator angle.
  - It would normally be issued while the rotator is tracking. In that case it makes an offset adjustment to the polynomials being generated by PCS.
  - Issued when the rotator is HOLDING, it will move the rotator by the specified amount, resuming holding at that new position.

Rotator commands

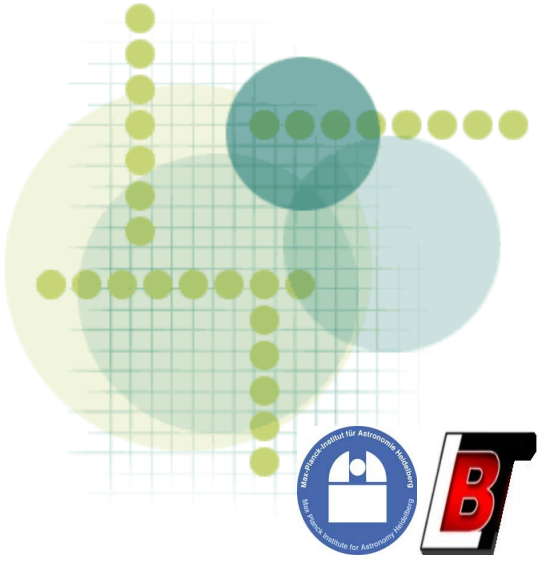
- `RotTrack ( const char * SIDE )`
  - Makes rotator begin tracking according to the trajectory it is currently receiving from the PCS.
  - It will in general, need to do a slew to the target position and then start tracking.
- `RotHold ( const char * SIDE )`
  - If the rotator is tracking or slewing, makes it stop moving and hold position at the point it was at when it received the hold command.
  - If the rotator is already holding position, this command has no effect.

Rotator commands

- `RotMaximizeTime ( const char * SIDE )`
  - provides some control over the use of the rotator's cable wrap.
  - If they are not in the wrap which maximizes observing time on the object, one or both will do a "slew-to-track" to acquire the same object in the other end of their cable wrap.
  - This command will either do nothing or it will slew the rotator and/or AZ axis 360 degrees.
- `RotServicePosition ( double ANGLE, rotatorType ROTMODE, const char * SIDE )`
  - Makes the rotator move to the specified angle in the specified coordinate frame and hold at that position.

Rotator commands

- `RotNextPosition ( double RA, double DEC, double LIMIT, const char * SIDE )`
  - It has no affect on telescope or rotator motion.
  - The “time-to-limit” values for the “next position” in reflective memory will be computed using these parameters. It also returns the current AZ and EL for the specified object.



Guiding related commands

- `PresetGuiding ( Position ** GUIDESTARS, const char * SIDE )`
  - This command is issued to start the guiding.
  - The guide stars in the list will be tried in the order as they were provided.
  - The first one that is usable will be the one the GCS will use for the guiding and possible WF sensing.
- `UpdateGuidestar ( Position ** GUIDESTARS, const char * SIDE )`
  - This command is used to update the list of guide stars for the specified side.
  - In the current version of GCS it is required to issue this command before the start of the acquisition in order to have an effect.
  - This restriction might change or become obsolete with future versions of GCS .

Guiding related commands

- `StopGuiding ( )`
  - This command stops the current guiding operation.
- `StartGuiding ( )`
  - This command is used to start again the guiding loop, using the existing information and the setup declared in the last `PresetGuiding` command provided for the given telescope side.
- `PauseGuiding ( )`
  - This command is issued to temporarily suspend the current guiding operation.
- `ResumeGuiding ( )`
  - This command resumes suspended operation after a `PauseGuiding`.