**SUN/257.1**

Starlink Project
Starlink User Note 257.1

D.S. Berry

3rd July 2006

# ATL
# A Library of AST Utility Routines
# Version 1.0
# Programmer's Manual

## Abstract

ATL provides high level utility functions for handling WCS and other AST-related tasks.

# Contents

## 1 Introduction

This library contains routines that use the AST library (SUN/211) to perform various higher-level utility tasks.

## A Routine Descriptions

## ATL_ADDWCSAXIS
## Add one or more axes to an NDFs WCS FrameSet

**Description:**

This routine adds one or more new axes to all the Frames in an NDF WCS FrameSet. Frames that are known to be NDF-special (e.g. GRID, AXIS, PIXEL and FRACTION) are expanded to include a number of extra appropriate axes equal to the Nin attribute of the supplied Mapping. all other Frames in the FrameSet are replaced by CmpFrames holding the original Frame and the supplied Frame. These new axes are connected to the new GRID axes using the supplied Mapping.

**Invocation:**

```
CALL ATL_ADDWCSAXIS( WCS, MAP, FRM, LBND, UBND, STATUS )
```

**Arguments:**

**WCS = INTEGER (Given)**
A pointer to a FrameSet that is to be used as the WCS FrameSet in an NDF. This imposes the restriction that the base Frame must have Domain GRID.

**MAP = INTEGER (Given)**
A pointer to a Mapping. The forward transformation should transform the new GRID axes into the new WCS axes.

**FRM = INTEGER (Given)**
A pointer to a Frame defining the new WCS axes.

**LBND() = INTEGER (Given)**
An array holding the lower pixel index bounds on the new axes. The length of this array should beq aual to the Nin attribute of the MAP Mapping.

**UBND() = INTEGER (Given)**
An array holding the upper pixel index bounds on the new axes. The length of this array should beq aual to the Nin attribute of the MAP Mapping.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- This routine is just a wrapper around the C function atlAddWcsAxis.
- The new axes are appended to the end of the existing axes, so the axis indices associated with the new axes will extend from " nold+1" to " nold+nnew" , where " nold" is the number of axes in the original Frame, and " nnew" is the number of new axes.
- An error will be reported if the Nout attribute of " map" is different to the Naxes attribute of " frm" .

## ATL_AXTRM
## Trim axes from the current Frame of a FrameSet

**Description:**

This routine ensures that the number of axes in the current Frame of the supplied FrameSet is the same as the number in the base Frame. If this is not the case on entry, one or more new Frames with the required number of axes are created and added into the FrameSet, one of which becomes the new current Frame. The only case in which more than one new Frame is added is if the current Frame has too many axes, and the FrameSet contains more than one " ROI" Frame (that is, Frames which are Regions and which have a Domain name beginning with " ROI" ). If the FrameSet contains zero or one ROI Frame, then only a single new Frame is added into the FrameSet.

If the original current Frame has too few axes, the new Frame is a copy of the original current Frame with extra simple axes added to the end. These extra axes are supplied a value of AST__BAD by the Mapping which connects the original current Frame to the new current Frame.

If the original current Frame has too many axes, one or more new Frames will be created by picking the specified axes from the original current Frame. Each of these Frames is added into the FrameSet using a Mapping which has a forward transformation which simply drops the values for the unselected axes. The inverse transformation (from new to old Frame) attempts to assign usable values for the dropped axes if possible. If this is not possible, then AST__BAD is assigned to the dropped axes.

Two methods are used for finding suitable values to assign to dropped axes. The first is only possible if the value for a dropped axis can be determined uniquely from the value of one of the retained axes. This may be the case for instance in a situation where (RA,wavelength) axes were selected from the (RA,Dec,Wavelength) axes describing a 2D longslit spectrum. The missing Dec value can probably be determined from the RA value because the relationship between RA and Dec is determined by the position and orientation of the slit on the sky.

If it is not possible to determine the value for a dropped axis in this way, then a search is made for Frames that are Regions having a Domain name beginning with " ROI" . If any are found, then a new Frame is added into the FrameSet for each ROI Region found, connected to the original current Frame via a PermMap. The values to be assigned to the dropped axes by the inverse PermMap transformation are determined by transforming the bounding box of the corresponding ROI Region into the original current Frame. The assigned axis values are the mean values of the transformed bounding box on each dropped axis. The Domain name of the corresponding ROI Region is stored in the Ident attribute of each new Frame so that later code can identify the corresponding ROI Region, and is also appended to the end of the Frame' s Domain. The new Frame corresponding to the first ROI Region found in the FrameSet is left as the current Frame on exit.

**Invocation:**

```
CALL ATL_AXTRM( IWCS, AXES, LBND, UBND, WORK, STATUS )
```

**Arguments:**

**IWCS = INTEGER (Given)**

The FrameSet to use. A new current Frame may be added to the FrameSet by this routine.

**AXES( ∗ ) = INTEGER (Given)**

The one-based indices of the axes to be retained in the event of there being too many axes in the original current Frame of IWCS. The number of values in the array should be equal to the number of axes in the base Frame of IWCS (i.e the number of pixel axes).

**LBND( ∗ ) = INTEGER (Given)**

The lower pixel index bounds of the NDF from which the FrameSet was obtained. The number of values in the array should be equal to the number of axes in the base Frame of IWCS (i.e the number of pixel axes).

**UBND( $*$ ) = INTEGER (Given)**
The upper pixel index bounds of the NDF from which the FrameSet was obtained. The number of values in the array should be equal to the number of axes in the base Frame of IWCS (i.e the number of pixel axes).

**WORK( $*$ ) = INTEGER (Given)**
Work space. It's length should be at least twice as large as the largest pixel dimension implied by LBND and UBND.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_CHRSPLITRE
## Extract sub-strings matching a specified regular expression

**Description:**

This routine compares the supplied string with the supplied regular expression. If they match, each section of the test string that corresponds to a parenthesised sub-string in the regular expression is copied and stored in the returned GRP group.

**Invocation:**

```
CALL ATL_CHRSPLITRE( STR, REGEXP, MATCHEND, IGRP, STATUS )
```

**Arguments:**

**STR = CHARACTER $*$ ( $*$ ) (Given)**
Pointer to the string to be split.

**REGEXP = CHARACTER $*$ ( $*$ ) (Given)**
The regular expression. See " Template Syntax:" in the astChrSub prologue. Note, this function differs from astChrSub in that any equals signs (=) in the regular expression are treated literally.

**MATCHEND = INTEGER (Returned)**
The index of the character that follows the last character within the supplied test string (STR) that matched any parenthesises sub-section of " regexp" . A value of 0 is returned if no matches were found.

**IGRP = INTEGER (Given and Returned)**
An identifier for an existing GRP group, or GRP__NOID. If GRP__NOID is supplied a new empty group is created and its identifier returned. On exit, the group is extended by appending to it a copy of each sub-string extracted from the supplied string.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- This routine is just a wrapper around the C function atlChrSplitRE.
- If a parenthesised sub-string in the regular expression is matched by more than one sub-string within the test string, then only the first is returned. To return multiple matches, the regular expression should include multiple copies of the parenthesised sub-string (for instance, separated by " .+?" if the intervening string is immaterial).

## ATL_CPPLA
## Copy attributes from one Plot to another

**Description:**

This routine copies all public attribute values from one AST Plot to another AST Plot. The attributes copied are those that affect the visual appearance of the Plot.

**Invocation:**

```
CALL ATL_CPPLA( IPLOT1, IPLOT2, FIXATE, STATUS )
```

**Arguments:**

**IPLOT1 = INTEGER (Given)**
The source Plot.

**IPLOT2 = INTEGER (Given)**
The destination Plot.

**FIXATE = LOGICAL (Given)**
If .FALSE., then attribute values are only set in IPLOT2 if they have been assigned an explicit value (i.e. are not defaulted) in IPLOT1. If .TRUE., then values are set explicitly in IPLOT2 whether they are default values or not.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_CREAT
## Write an AST Object to a text file or NDF specified using an environment parameter

**Description:**

Write an AST Object to a text file or NDF specified using an environment parameter.

**Invocation:**

```
CALL ATL_CREAT( FPARAM, IAST, STATUS )
```

**Arguments:**

**FPARAM = CHARACTER * ( * ) (Given)**

The parameter name. If the supplied string contains a colon, then the parameter name is taken to be the string following the colon. The string before the colon indicates the format required for the output text file:

" AST:" - AST_SHOW format " STCS:" - STCS format " MOC-JSON:" - MOC JSON format " MOC" - MOC " string" format " XML:" - AST XML format " FITS-xxx:" - FITS, using the specified encoding " NATIVE:" - FITS, using NATIVE encoding

The default (i.e. used if the string does not contain a colon) is " AST" . Attribute values for the Channel (of whatever class) can be specified using the environment variable ATOOLS_CHATT_OUT.

**IAST = INTEGER (Given)**

The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**

The global status.

## ATL_CUTPL
## Create a Plot covering a sub-region of another Plot

**Description:**

This routine creates a new Plot with the same attributes as a supplied Plot, but covering a sub-region within the world coordinate system and graphics viewport.

**Invocation:**

```
CALL ATL_CUTPL( IPLOT1, IFRM, DLBND, DUBND, IPLOT2, STATUS )
```

**Arguments:**

**IPLOT1 = INTEGER (Given)**
The source Plot.

**IFRM = INTEGER (Given)**
Index of the Frame within IPLOT1 in which the bounds are supplied.

**DLBND( $*$ ) = DOUBLE PRECISION (Given)**
The axis values at the lower left corner of the region to be covered by the new Plot. The number of axis values supplied should equal the number of axes in the Frame identified by IFRM.

**DUBND( $*$ ) = DOUBLE PRECISION (Given)**
The axis values at the upper right corner of the region to be covered by the new Plot. The number of axis values supplied should equal the number of axes in the Frame identified by IFRM.

**IPLOT2 = INTEGER (Returned)**
The new Plot.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_FINDSKY
## Locate any sky axes within a Frame

**Description:**

This routine searches the supplied Frame (which may be a CmpFrame) for a SkyFrame. If found, it returns a pointer to the SkyFrame, together with the indices (within the supplied Frame) of the longitude and latitude axes.

**Invocation:**

```
CALL ATL_FINDSKY( FRAME, SKYFRAME, LATAX, LONAX, STATUS )
```

**Arguments:**

**FRAME = INTEGER (Given)**
The Frame to be searched.

**SKYFRAME = INTEGER (Returned)**
A pointer to the SkyFrame contained within FRAME, if any. If no SkyFrame is found, AST__NULL is returned.

**LATAX = INTEGER (Returned)**
The index (one-based) of the celestial latitude axis in FRAME. Returned equal to zero if no SkyFrame is found.

**LONAX = INTEGER (Returned)**
The index (one-based) of the celestial lonitude axis in FRAME. Returned equal to zero if no SkyFrame is found.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_FSPEC
## Locate a SpecFrame within a CmpFrame

**Description:**

This routine searches the supplied CmpFrame for an axis that is a SpecFrame. It returns the axis index of the SpecFrame within the CmpFrame, and also returns a pointer to the SpecFrame itself.

No error is reported if the CmpFrame does not contain a SpecFrame. If the CmpFrame contains more than one SpecFrame, the first (i.e. the lowest index) is returned.

**Invocation:**

```
CALL ATL_FSPEC( FRM, SPAX, SPFRM, STATUS )
```

**Arguments:**

**FRM= INTEGER (Given)**
The CmpFrame pointer.

**SPAX = INTEGER (Returned)**
The index of the spectral axis within the CmpFrame. Returned equal to zero if no spectral axis is found.

**SPFRM = INTEGER (Returned)**
A pointer to the SpecFrame. Returned equal to AST__NULL if no spectral axis is found.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_GETPIXELPARAMS
## Find typical values for " FITS-like" parameters describing a FrameSet

**Description:**

This function finds values that resemble the the FITS keywords CRVAL1/2/3.., CRPIX1/2/3..., CRDELT1/2/3... and CROTA2, on the assumption that the base Frame in the supplied FrameSet describe GRID coords (i.e. FITS pixel coords), and the current Frame describe the required WCS. It is not restricted to 2D FrameSets.

If the FrameSet can be written to a FitsChan successfully using FITS-WCS encoding, the the resulting keyword values are returned. Otherwise, the values are estimated by transforming closely spaced pixel positions along each axis. If the current Frame contains a SkyFrame, and the SkyFrame has a defined reference position, then this position specifies the returned CRVAL values. Otherwise, the reference position is assumed to be at the central pixel.

**Invocation:**

```
CALL ATL_GETPIXELPARAMS( FSET, DIMS, DEGS, CRPIX, CRVAL, CDELT, CROTA, STATUS )
```

**Arguments:**

**FSET = INTEGER (Given)**
The FrameSet.

**DIMS(∗) = INTEGER (Given)**
An array supplied holding the number of pixels along each edge of the pixel array. The number of elements in this array should match the number of axes in the base Frame of FSET.

**DEGS = LOGICAL (Given)**
If .TRUE., then the CRVAL, CDELT and CROTA values for sky axes are returned in units of degrees. Otherwise they are returned in radians.

**CRPIX(∗) = DOUBLE PRECISION (Returned)**
An array returned holding the position of the reference pixel in the base Frame of FSET. The number of elements in this array should match the number of axes in the base Frame of FSET.

**CRVAL(∗) = DOUBLE PRECISION (Returned)**
An array returned holding the position of the reference pixel in the current Frame of FSET. The number of elements in this array should match the number of axes in the current Frame of FSET.

**CDELT(∗) = DOUBLE PRECISION (Returned)**
An array returned holding the geodesic distance along each edge of the reference pixel, measured within the current Frame of FSET. The number of elements in this array should match the number of axes in the base Frame of FSET.

**CROTA = DOUBLE PRECISION (Returned)**
The angle from north in the current frame of FSET to the second spatial pixel axis, measured positive through east. This will be returned set to AST__BAD if the current frame of FSET does not contain a SkyFrame.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_GTGRP
## Obtain lines of text from a parameter, and store them in a GRP group

**Description:**

Currently this routine expects the parameter to be associated with:

1 - a text file (the returned group contains the lines of the file). 2 - a FITS file (the returned group contains the FITS headers).

In future it may be possible to add other ways of using the parameter (i.e. by associating it with objects other than text files).

**Invocation:**

```
CALL ATL_GTGRP( PARAM, IGRP, STATUS )
```

**Arguments:**

**PARAM = CHARACTER $*$ ( $*$ ) (Given)**
The parameter name.

**IGRP = INTEGER (Returned)**
The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_KY2HD
## Converts an AST KeyMap into an HDS structure

**Description:**

This routine copies the contents of an AST KeyMap into a supplied HDS structure.

**Invocation:**

```
CALL ATL_KY2HD( KEYMAP, LOC, STATUS )
```

**Arguments:**

**KEYMAP = INTEGER (Given)**
The AST KeyMap identifier.

**LOC = CHARACTER $*$ (DAT__SZLOC) (Given)**
A locator for the HDS object into which the KeyMap contents are to be copied.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_KYCHK
## Reports an error if a given key is not found in a KeyMap

**Description:**

This routine checks a supplied KeyMap for a supplied Key and reports a supplied error if the key is not found.

**Invocation:**

```
CALL ATL_KYCHK( KEYMAP, KEY, ERRMSG, STATUS )
```

**Arguments:**

**KEYMAP = INTEGER (Given)**

Pointer to the AST KeyMap.

**KEY = CHARACTER * ( * ) (Given)**

The key to check.

**ERRMSG = CHARACTER * ( * ) (Given)**

The error message to report if the key is not found. This may include references to the MSG token " ^K" which will hold the supplied key name.

**STATUS = INTEGER (Given and Returned)**

The global status.

## ATL_MATCHREGION
## Ensure the axes in a Region match those in a Frame

**Description:**

This routine checks for matching axes in a supplied Region and Frame. If possible, a new region is created containing a set of axes that correspond in number and type (but not necessarily in specific attributes) to those in the supplied Frame. This means that AST_CONVERT should be able to find a Mapping between the supplied Frame and the returned Region. Note, the order of the axes in the returned Region may not match those in the Frame, but AST_CONVERT will be able to identify any required re-ordering.

If it is not possible to find a matching Region (for instance, if there are no axes in common between the supplied Region and Frame), an error is reported.

**Invocation:**

```
CALL ATL_MATCHREGION( REGION, FRAME, NEWREG, STATUS )
```

**Arguments:**

**REGION = INTEGER (Given)**
An AST pointer for the Region to be modified.

**FRAME = INTEGER (Given)**
An AST pointer for a Frame to be matched.

**NEWREG = INTEGER (Returned)**
An AST pointer to the returned Region.

**STATUS = INTEGER (Given and Returned )**
The global status.

## ATL_MGFTS
## Merge two FITS headers

**Description:**

This routine merges two FITS headers, each supplied in an AST FitsChan, in one of several different ways. The resulting merged list of headers is returned in a new FitsChan.

**Invocation:**

```
CALL ATL_MGFTS( METHOD, FC1, FC2, FC3, STATUS )
```

**Arguments:**

**METHOD = INTEGER (Given)**

Indicates how the two FITS headers should be merged:

1 - Concatenation. Store the contents of FC1 in the returned FitsChan, and then append the contents of FC2 to the end of the returned FitsChan. No checks are made for multiple occurences of the same keyword.

2 - Union (with priority given to FC2): For every header in FC1, see if FC2 contains the same keyword. If it does not, copy the FC1 header to the returned FitsChan. Then append the contents of FC2 to the end of the returned FitsChan.

3 - Overlap: For every header in FC1, see if FC2 contains the same keyword. If it does, and if the keyword value is the same in both FitsChans, copy the FC1 header to the returned FitsChan.

4 - Union (with priority given to FC1): Copy FC1 to the output, then for every header in FC2 append it to the end of the returned FitsChan unless the card already exists. This is similar to method 2 except that cards from FC2 are dropped instead of cards from FC1.

**FC1 = INTEGER (Given)**

Pointer to the first FitsChan.

**FC2 = INTEGER (Given)**

Pointer to the second FitsChan.

**FC3 = INTEGER (Returned)**

Pointer to the returned FitsChan.

**STATUS = INTEGER (Given and Returned)**

The global status.

**Notes:**

- The contents of FC1 and FC2 are unchanged on exit.
- For METHOD 3 (overlap), floating point values are compared by formatting into a string (using the accuracy specified by the FitsDigits attributes of the two supplied FitsChans) and then comparing the formatted strings for exact equality.
- Method 4 exists to allow a new header to be appended whilst retaining the primary order of the cards from the first header.

## ATL_MKLUT
## Create a Mapping to connect two 1D array of values

**Description:**

This routine creates a 1D Mapping which translates an X into a Y value on the basis of supplied tables of corresponding X and Y. This is like an AST LutMap except that the LutMap class requires Y to be tabulated at equal X intervals, whereas this routine allows Y to be tabulated at arbitrary X intervals.

**Invocation:**

```
CALL ATL_MKLUT( IX, IY, NPNT, NVAR, FRM, TABLE, MAP, STATUS )
```

**Arguments:**

**IX = INTEGER (Given)**
The index of the X values within the TABLE array.

**IY = INTEGER (Given)**
The index of the Y values within the TABLE array.

**NPNT = INTEGER (Given)**
The number of values supplied for each variable in the TABLE array.

**NVAR = INTEGER (Given)**
The number of variables described in the table. This will be at least 2 (for X and Y) but may be more.

**FRM = INTEGER (Given)**
If not AST__NULL, then this should be an AST pointer to a Frame with NVAR axes which will be used to normalise the axis values before creating the LutMap. No normalisation occurs if a value of AST__NULL is supplied.

**TABLE( NPNT, NVAR ) = DOUBLE PRECISION (Given and Returned)**
The table containing corresponding X and Y values. The table can also contain values for other variables, which will be ignored. These will be normalised on exit using the AST Frame supplied by FRM.

**MAP = INTEGER (Returned)**
An AST pointer to the returned Mapping, or AST__NULL if no Mapping could be created.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- It is only possible to create the Mapping if the tabluated X values are monotonic increasing or decreasing.
- The returned Mapping will have an inverse Transformation only if Y increases or decreases monotonically with X.

## ATL_NOTIF
## Print a message to the screen if ATOOLS_VERBOSE is set

**Description:**
Print a message to the screen if ATOOLS_VERBOSE is set.

**Invocation:**

```
CALL ATL_NOTIF( MSG, STATUS )
```

**Arguments:**

**MSG = CHARACTER * ( * ) (Given)**
The message.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_PLROI
## Create a set of Plots associated with each ROI in a given Plot

**Description:**

This routine searches the supplied Plot for ROI Frames (see ATL_AXTRM). For each ROI Frame found, it creates a new Plot that covers just the region of graphics coords occupied by the ROI. These new Plots are returned in an AST KeyMap.

**Invocation:**

```
CALL ATL_PLROI( IPLOT, RPLOTS, STATUS )
```

**Arguments:**

**IPLOT = INTEGER (Given)**
The supplied Plot to search for ROI Frames.

**RPLOTS = INTEGER (Returned)**
An AST KeyMap holding the Plots associated with the ROI Frames. The key used to identify each Plot within the KeyMap is the Domain name of the corresponding ROI Frame.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_PTFTI
## Store a keyword value in a FitsChan, replacing any existing value

**Description:**

This routine stores a value for a FITS keyword in a FitsChan. If the keyword already has a value in the FitsChan, the existing value is replaced with the new value. Otherwise, the new keyword is added to the end of the FitsChan. On exit, the current Card in the FitsChan is the card following the new keyword value (or end-of-file if the new card is the last one in the FitsChan).

**Invocation:**

```
CALL ATL_PTFTI( THIS, NAME, VALUE, COMMNT, STATUS )
```

**Arguments:**

**THIS = INTEGER (Given)**
Pointer to the FitsChan to use.

**NAME = CHARACTER $*$ ( $*$ ) (Given)**
The FITS keyword name. This may be a complete FITS header card, in which case the keyword to use is extracted from it. No more than 80 characters are read from this string.

**VALUE = INTEGER (Given)**
The new keyword value. If this is VAL__BADI, then an undefined value will be stored in the FitsChan.

**COMMNT = CHARACTER $*$ ( $*$ ) (Given)**
A new comment for the keyword. If this is blank, any comment in the NAME string is used. If the NAME string contains no comment, any existing comment for the keyword in the FitsChan is retained.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- This routine is not processed using GENERIC because the names of the required AST routines do not use standard data type codes.

## ATL_PTFTL
## Store a keyword value in a FitsChan, replacing any existing value

**Description:**

This routine stores a value for a FITS keyword in a FitsChan. If the keyword already has a value in the FitsChan, the existing value is replaced with the new value. Otherwise, the new keyword is added to the end of the FitsChan. On exit, the current Card in the FitsChan is the card following the new keyword value (or end-of-file if the new card is the last one in the FitsChan).

**Invocation:**

```
CALL ATL_PTFTL( THIS, NAME, VALUE, COMMNT, STATUS )
```

**Arguments:**

**THIS = INTEGER (Given)**
Pointer to the FitsChan to use.

**NAME = CHARACTER $*$ ( $*$ ) (Given)**
The FITS keyword name. This may be a complete FITS header card, in which case the keyword to use is extracted from it. No more than 80 characters are read from this string.

**VALUE = LOGICAL (Given)**
The new keyword value.

**COMMNT = CHARACTER $*$ ( $*$ ) (Given)**
A new comment for the keyword. If this is blank, any comment in the NAME string is used. If the NAME string contains no comment, any existing comment for the keyword in the FitsChan is retained.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- This routine is not processed using GENERIC because the names of the required AST routines do not use standard data type codes.

## ATL_PTFTR
## Store a keyword value in a FitsChan, replacing any existing value

**Description:**

This routine stores a value for a FITS keyword in a FitsChan. If the keyword already has a value in the FitsChan, the existing value is replaced with the new value. Otherwise, the new keyword is added to the end of the FitsChan. On exit, the current Card in the FitsChan is the card following the new keyword value (or end-of-file if the new card is the last one in the FitsChan).

**Invocation:**

```
CALL ATL_PTFTR( THIS, NAME, VALUE, COMMNT, STATUS )
```

**Arguments:**

**THIS = INTEGER (Given)**
Pointer to the FitsChan to use.

**NAME = CHARACTER * ( * ) (Given)**
The FITS keyword name. This may be a complete FITS header card, in which case the keyword to use is extracted from it. No more than 80 characters are read from this string.

**VALUE = REAL (Given)**
The new keyword value. If this is VAL__BADR, then an undefined value will be stored in the FitsChan.

**COMMNT = CHARACTER * ( * ) (Given)**
A new comment for the keyword. If this is blank, any comment in the NAME string is used. If the NAME string contains no comment, any existing comment for the keyword in the FitsChan is retained.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- This routine is not processed using GENERIC because the names of the required AST routines do not use standard data type codes.

## ATL_PTFTS
## Store a keyword value in a FitsChan, replacing any existing value

**Description:**

This routine stores a value for a FITS keyword in a FitsChan. If the keyword already has a value in the FitsChan, the existing value is replaced with the new value. Otherwise, the new keyword is added to the end of the FitsChan. On exit, the current Card in the FitsChan is the card following the new keyword value (or end-of-file if the new card is the last one in the FitsChan).

**Invocation:**

```
CALL ATL_PTFTS( THIS, NAME, VALUE, COMMNT, STATUS )
```

**Arguments:**

**THIS = INTEGER (Given)**
Pointer to the FitsChan to use.

**NAME = CHARACTER $*$ ( $*$ ) (Given)**
The FITS keyword name. This may be a complete FITS header card, in which case the keyword to use is extracted from it. No more than 80 characters are read from this string.

**VALUE = CHARACTER $*$ ( $*$ ) (Given)**
The new keyword value. If this is ATL__BADC, then an undefined value will be stored in the FitsChan. Note, the ATL__BADC string is defined in include file ATL_PAR, together with the integer constant ATL_SZBADC, which is equal to the length of the ATL__BADC string.

**COMMNT = CHARACTER $*$ ( $*$ ) (Given)**
A new comment for the keyword. If this is blank, any comment in the NAME string is used. If the NAME string contains no comment, any existing comment for the keyword in the FitsChan is retained.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- This routine is not processed using GENERIC because the names of the required AST routines do not use standard data type codes.

## ATL_PXDUP
## Ensure the number of WCS axes is no less than the number of pixel axes

**Description:**

This routine ensures that the number of axes in the current Frame (WCS Frame) of a FrameSet is at least equal to the number of axes in the base Frame (PIXEL or GRID Frame). If the initial number of current Frame axes is too small, extra axes are added to the current Frame by duplicating selected pixel axes.

**Invocation:**

```
CALL ATL_PXDUP( IWCS, POS, STATUS )
```

**Arguments:**

**IWCS = INTEGER (Given)**

The supplied FrameSet.

**POS( $*$ ) = DOUBLE PRECISION (Given)**

The base Frame coords of a position which has good current Frame coords.

**STATUS = INTEGER (Given and Returned)**

The global status.

## ATL_RDCH
## Read an AST Object from a GRP group using a Channel

**Description:**

Read an AST Object from a GRP group using a Channel. The Channel can be configured using a set of attribute settings specified in the environment variable ATOOLS_CHATT_IN.

**Invocation:**

```
CALL ATL_RDCH( IGRP, IAST, STATUS )
```

**Arguments:**

**IGRP = INTEGER (Given)**
An identifier for the group holding the text.

**IAST = INTEGER (Returned)**
The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- If the group contains the dump of a Channel (of any class), then the Object returned via IAST will be the Channel itself. The exception to this is that if the " Begin " line at the start of the dump ends with the string " (Read)" , then the returned IAST Object will be the Object read from the Channel, rather than the Channel itself. For instance, if the group contains the dump of a FitsChan, and the first line of the dump is " Begin FitsChan(Read)" , then the returned IAST object will be the Object read from the FitsChan, rather than the FitsChan itself. This facility is only available for top level objects (e.g. FitsChans contained within FitsChans cannot be read in this way).

## ATL_RDFCH
## Read an AST Object from a GRP group using a FitsChan

**Description:**

Read an AST Object from a GRP group using a FitsChan. The FitsChan can be configured using a set of attribute settings specified in the environment variable ATOOLS_CHATT_IN.

**Invocation:**

```
CALL ATL_RDFCH( IGRP, IAST, STATUS )
```

**Arguments:**

**IGRP = INTEGER (Given)**
An identifier for the group holding the text.

**IAST = INTEGER (Returned)**
The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_RDGRP
## Read an AST Object from a GRP group

**Description:**

Read an AST Object from a GRP group. The text in the group can be either an AST Object dump, a set of FITS headers, or an STC-S description.

**Invocation:**

```
CALL ATL_RDGRP( IGRP, IAST, STATUS )
```

**Arguments:**

**IGRP = INTEGER (Given)**
An identifier for the group holding the text.

**IAST = INTEGER (Returned)**
The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- If the group contains the AST dump of a Channel (of any class), then the Object returned via IAST will be the Channel itself. The exception to this is that if the " Begin " line at the start of the dump ends with the string " (Read)" , then the returned IAST Object will be the Object read from the Channel, rather than the Channel itself. For instance, if the group contains the AST dump of a FitsChan, and the first line of the dump is " Begin FitsChan(Read)" , then the returned IAST object will be the Object read from the FitsChan, rather than the FitsChan itself. This facility is only available for top level objects (e.g. FitsChans contained within FitsChans cannot be read in this way).

## ATL_RDMOC
## Read an AST Object from a GRP group using an MocChan

**Description:**

Read an AST Object from a GRP group using an MocChan.The MocChan can be configured using a set of attribute settings specified in the environment variable ATOOLS_CHATT_IN.

**Invocation:**

```
CALL ATL_RDMOC( IGRP, IAST, STATUS )
```

**Arguments:**

**IGRP = INTEGER (Given)**
An identifier for the group holding the text.

**IAST = INTEGER (Returned)**
The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_RDSTCS
## Read an AST Object from a GRP group using an StcsChan

**Description:**

Read an AST Object from a GRP group using an StcsChan.The StcsChan can be configured using a set of attribute settings specified in the environment variable ATOOLS_CHATT_IN.

**Invocation:**

```
CALL ATL_RDSTCS( IGRP, IAST, STATUS )
```

**Arguments:**

**IGRP = INTEGER (Given)**
An identifier for the group holding the text.

**IAST = INTEGER (Returned)**
The AST Object, or AST__NULL.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_RM
## Remove a file

**Description:**

This subroutine calls the " PSX_REMOVE" RTL function to remove a specified file. No error occurs if the file cannot be removed for any reason.

**Invocation:**

```
CALL ATL_RM( FILE, STATUS )
```

**Arguments:**

**FILE = CHARACTER $*$ ( $*$ ) (Given)**

The path to the file.

**STATUS = INTEGER (Given and Returned)**

The inherited global status.

## ATL_RMBLFT
## Remove contiguous blanks from FITS header

**Description:**

This routine removes contiguous blank lines from the FITS header.

**Invocation:**

```
CALL ATL_RMBLFT( FC, STATUS )
```

**Arguments:**

**FC1 = INTEGER (Given)**

Pointer to the FitsChan to clean.

**STATUS = INTEGER (Given and Returned)**

The global status.

## ATL_TOLUT
## Approximate a supplied Mapping by one or more LutMaps

**Description:**

This routine creates a Mapping that uses one or more LutMaps to approximate the supplied Mapping. The supplied Mapping must have 1 input but can have up to ATL__MXDIM outputs. One LutMap will be created for each output and combined in parallel in the output Mapping. The range of input VALUE over which the approximation is to be valid is specified, together with the input step size for the LutMaps.

**Invocation:**

```
CALL ATL_TOLUT( INMAP, XLO, XHI, DX, OPTS, OUTMAP, STATUS )
```

**Arguments:**

**INMAP = INTEGER (Given)**
The Mapping to be approximated. Must have only 1 input, and up to ATL__MXDIM outputs.

**XLO = DOUBLE PRECISION (Given)**
The lowest value of the INMAP input value for which the returned Mapping will be used.

**XHI = DOUBLE PRECISION (Given)**
The highest value of the INMAP input value for which the returned Mapping will be used.

**DX = DOUBLE PRECISION (Given)**
The increment in INMAP input value to be used when creating the LutMaps.

**OPTS = CHARACTER * ( * ) (Given)**
Options to pass to the LutMap constructor.

**OUTMAP = INTEGER (Returned)**
An AST pointer to the returned Mapping, or AST__NULL if no Mapping could be created. This will have the same number of inputs and outputs as INMAP.

**STATUS = INTEGER (Given and Returned)**
The global status.

## ATL_TTLPL
## Display a Plot Title without using AST_GRID

**Description:**

This routine display the Plot Title at the top of the area covered by the Plot, but does not draw anything else (e.g. axes, tick marks, borders, labels, etc). It does not need the inverse transformation from current to base Frame to be defined in the Plot.

**Invocation:**

```
CALL ATL_TTLPL( IPLOT, STATUS )
```

**Arguments:**

**IPLOT = INTEGER (Given)**

The Plot.

**STATUS = INTEGER (Given and Returned)**

The global status value.

## ATL_WCSPX
## Create a WCS FrameSet from a SPECX file

**Description:**

This returns a pointer to a FrameSet describing the WCS information in a SPECX file. The current Frame is a 3D Frame with RA on axis 1, DEC on axis 2, and frequency on axis 3. The base Frame is a 3D GRID Frame. The parameters defining the axes are read from two supplied AST KeyMaps, which should contain values for various items read from a SPECX and SPECX_MAP extensions in a SPECX map (the observatory location are provided separately by the caller).

**Invocation:**

```
CALL ATL_WCSPX( KM1, KM2, CRPIX, OBSLON, OBSLAT, IWCS, STATUS )
```

**Arguments:**

**KM1 = INTEGER (Given)**

Pointer to an AST KeyMap holding items read from the SPECX extension in the required SPECX map file. The key for each entry is identicial to the name of the item in the SPECX extension. It should contain entries with the following keys (the data type with which each entry is accessed is also shown):

" JFREST(1)" - _INTEGER " RA_DEC(1)" - _DOUBLE " RA_DEC(2)" - _DOUBLE " DPOS(1)" - _DOUBLE " DPOS(2)" - _DOUBLE " IDATE" - _CHAR " ITIME" - _CHAR " LSRFLG" - _INTEGER " V_SETL(4)" - _DOUBLE " JFCEN(1)" - _INTEGER " JFINC(1)" - _INTEGER " IFFREQ(1)" - _DOUBLE

In addition, the KeyMap may contain an item " CENTRECODE" (_INTEGER) that specifies the co-ordinate system to which the RA_DEC and DPOS values refer. It may take any of the following values:

1 : AZEL 4 : RD (geocentric apparent RA and Dec) 6 : RB (FK4 1950 RA and Dec) 7 : RJ (FK5 2000 RA and Dec) 8 : GA (galactic longitude and latitude)

An error is reported if any other value is supplied for CENTRECODE. If CENTRECODE is missing a value of 6 (FK4 B1950) is assumed.

**KM2 = INTEGER (Given)**

Pointer to an AST KeyMap holding items read from the SPECX_MAP extension in the required SPECX map file. The value AST__NULL should be provided if the SPECX file does not have a SPECX_MAP extension. If supplied, it should contain entries with the following keys:

" CELLSIZE(1)" - _DOUBLE " CELLSIZE(2)" - _DOUBLE " POSANGLE" - _DOUBLE

In addition, the KeyMap may contain an item " CELLCODE" (_INTEGER) that specifies the co-ordinate system to which the CELLSIZE and POSANGLE values refer. It may take any of the values listed for CENTRECODE in the " KM1" argument description above. If CELLCODE is missing a value of 6 (FK4 B1950) is assumed.

**CRPIX( 3 ) = DOUBLE PRECISION (Given)**

The pixel co-ordinates at the reference point. The spatial position of the reference point is given by RA_DEC(1) and RA_DEC(2) offset by the DPOS(1) and DPOS(2) values (all these are in the KM1 KeyMap). The spectral reference value is given by the JFCEN item in KM1.

**OBSLON = DOUBLE PRECISION (Given)**

The geodetic longitude of the observatory. Radians, positive east.

**OBSLAT = DOUBLE PRECISION (Given)**

The geodetic latitude of the observatory. Radians, positive north.

**IWCS = INTEGER (Returned)**
The returned FrameSet.

**STATUS = INTEGER (Given and Returned)**
The global status.

**Notes:**

- Various assumptions are made about the meaning of several items in the SPECX extensions. These are described in the code comments.
- Double Sideband is always assumed

## atlAddWcsAxis
## Add one or more axes to an NDFs WCS FrameSet

**Description:**

This function adds one or more new axes to all the Frames in an NDF WCS FrameSet. Frames that are known to be NDF-special (e.g. GRID, AXIS, PIXEL and FRACTION) are expanded to include a number of extra appropriate axes equal to the Nin attribute of the supplied Mapping. all other Frames in the FrameSet are replaced by CmpFrames holding the original Frame and the supplied Frame. These new axes are connected to the new GRID axes using the supplied Mapping.

**Invocation:**

```
void atlAddWcsAxis( AstFrameSet *wcs, AstMapping *map, AstFrame *frm, int *lbnd, int
*ubnd, int *status )
```

**Arguments:**

**wcs**
A pointer to a FrameSet that is to be used as the WCS FrameSet in an NDF. This imposes the restriction that the base Frame must have Domain GRID.

**map**
A pointer to a Mapping. The forward transformation should transform the new GRID axes into the new WCS axes.

**frm** A pointer to a Frame defining the new WCS axes.

**lbnd**
An array holding the lower pixel index bounds on the new axes. If a NULL pointer is supplied, a value of 1 is assumed for all the new axes.

**ubnd**
An array holding the upper pixel index bounds on the new axes. If a NULL pointer is supplied, any FRACTION Frame in the supplied FrameSet is removed.

**status**
Pointer to the global status variable.

**Notes:**

- The new axes are appended to the end of the existing axes, so the axis indices associated with the new axes will extend from " nold+1" to " nold+nnew" , where " nold" is the number of axes in the original Frame, and " nnew" is the number of new axes.
- An error will be reported if the Nout attribute of " map" is different to the Naxes attribute of " frm" .

## atlDumpFits
## Write the contents of a FitsChan to a text file

**Description:**

This function creates a new text file containing the contents of the supplied FitsChan as a set of FITS header cards. The name of the text file is obtained via the evironment using a specified parameter.

**Invocation:**

```
void atlDumpFits( const char *param, AstFitsChan *fc, int *status )
```

**Arguments:**

**param**
The parameter name.

**fc**
A pointer to the FitsChan.

**status**
Pointer to the global status variable.

## atlFrameSetSplit
## Extract axes from a supplied FrameSet to create a new FrameSet

**Description:**

This function searches the current Frame of the supplied FrameSet for axes that have a specified Domain. If any are found, and if they correspond to a distinct subset of axes in the base Frame of the supplied FrameSet (i.e. they are independent of the other axes), a new FrameSet is created and returned in which the current Frame contains the requested axes from the current Frame of the supplied FrameSet, and the base Frame contains the corresponding axes from the base Frame of the supplied FrameSet. If possible, any other Frames in the supplied FrameSet are also split and added to the returned FrameSet.

If the search is unsuccessful, or if the required current Frame axes are not independent of the other axes, then each of the other Frames in the FrameSet is searched in the same way (excluding the base Frame). If no suitable Frame can be found, a NULL pointer is returned but no error is reported.

**Invocation:**

```
AstFrameSet *atlFrameSetSplit( AstFrameSet *fset, const char *domain, int **bax, int
**cax, int *status )
```

**Arguments:**

**fset** The FrameSet to be split.

**domain**
The Domain value for the required current Frame axes. This can be a space-separated list of Domains, in which case each Domain will be used in turn until one is found which allows the supplied FrameSet to be split succesfully (any remaining Domain values will be ignored).

**bax** If not NULL, this should be the address of a pointer in which to return a pointer to an array holding the one-based indices of the base frame axes that were included in the returned FrameSet. The array should be freed using astFree when no longer needed. The length of the array will equal the number of base Frame axes in the returned FrameSet. A NULL pointer will be returned if the FrameSet could not be split.

**cax** If not NULL, this should be the address of a pointer in which to return a pointer to an array holding the one-based indices of the current frame axes that were included in the returned FrameSet. The array should be freed using astFree when no longer needed. The length of the array will equal the number of current Frame axes in the returned FrameSet. A NULL pointer will be returned if the FrameSet could not be split.

**status**
The global status.

**Returned Value:**

**A pointer to a new FrameSet, or NULL if no axes with the required**

**Domain could be found, or if the required axes do not correspond**

**to a distinct set of base Frame axes, or if an error occurs.**

## atlGetParam
## Create a vector character string entry in an AST KeyMap from a list of fixed length strings

**Description:**

This function obtains a value for a named environment parameter and stores it in the supplied KeyMap, using the parameter name as the key.

**Invocation:**

```
void atlGetParam( const char *param, AstKeyMap *keymap, int *status )
```

**Arguments:**

**param**

The parameter name.

**keymap**

A pointer to an existing KeyMap.

**status**

Pointer to the global status variable.

**Notes:**

- An error will be reported if the parameter value obtained from the environment is a vector with more than 100 elements.
- An error will be reported if any individual element in the parameter value obtained from the environment requires more than 255 when represented as a string.

## atlGetPixelParams
## Find typical values for " FITS-like" parameters describing a FrameSet

**Description:**

This function finds values that resemble the the FITS keywords CRVAL1/2/3.., CRPIX1/2/3..., CRDELT1/2/3... and CROTA2, on the assumption that the base Frame in the supplied FrameSet describe GRID coords (i.e. FITS pixel coords), and the current Frame describe the required WCS. It is not restricted to 2D FrameSets.

If the FrameSet can be written to a FitsChan successfully using FITS-WCS encoding, the the resulting keyword values are returned. Otherwise, the values are estimated by transforming closely spaced pixel positions along each axis. If the current Frame contains a SkyFrame, and the SkyFrame has a defined reference position, then this position specifies the returned CRVAL values. Otherwise, the reference position is assumed to be at the central pixel.

**Invocation:**

```
void atlGetPixelParams( AstFrameSet *fset, int *dims, int degs, double *crpix, double
*crval, double *cdelt, double *crota, int *status )
```

**Arguments:**

**fset** The FrameSet.

**dims**
Pointer to an array supplied holding the number of pixels along each edge of the pixel array. The number of elements in this array should match the number of axes in the base Frame of " fset" .

**degs**
If non-zero, then the crval, cdelt and crota values for sky axes are returned in units of degrees. Otherwise they are returned in radians.

**crpix**
Pointer to an array returned holding the position of the reference pixel in the base Frame of " fset" . The number of elements in this array should match the number of axes in the base Frame of " fset" .

**crval**
Pointer to an array returned holding the position of the reference pixel in the current Frame of " fset" . The number of elements in this array should match the number of axes in the current Frame of " fset" .

**cdelt**
Pointer to an array returned holding the geodesic distance along each edge of the reference pixel, measured within the current Frame of " fset" . The number of elements in this array should match the number of axes in the base Frame of " fset" .

**crota**
Pointer to a double in which to return the angle from north in the current frame of " fset" to the second spatial pixel axis, measured positive through east. This will be returned set to AST__BAD if the current frame of " fset" does not contain a SkyFrame.

**status**
The global status.

## atlMapGet1C
## Retrieve a vector of strings from an AST KeyMap entry as a list of fixed length strings

**Description:**

This function retrieves the null-terminated strings from a vector element in a KeyMap, and concatenates them into a list of fixed length strings, each padded with spaces.

**Invocation:**

```
int atlMapGet1C( AstKeyMap *this, const char *key, int bufsize, int len, int *nval,
char *buf, int *status )
```

**Arguments:**

**this**
A pointer to the KeyMap.

**key** The key for the entry.

**bufsize**
The length of the " buf" array.

**len** The required size ofr each fixed length string.

**nval**
Address of an int in which to return the number of fixed length strings returned in " buf" . This will be less than the number of elements in the KeyMap entry if the supplied buffer is not large enough to hold all the strings in the entry.

**buf** A pointer to a buffer in which to return the concatenated, fixed length strings.

**status**
Pointer to the global status variable.

**Returned Value:**

**Non-zero if an entry with the given key was found in the KeyMap,**

**and zero otherwise.**

## atlMapPut1C
## Create a vector character string entry in an AST KeyMap from a list of fixed length strings

**Description:**

This function splits up a supplied character array into a set of equal length sub-strings, null terminates them, and stores them as a character vector in a KeyMap. See also atlMapGet1S.

**Invocation:**

```
void atlMapPut1C( AstKeyMap *this, const char *key, const char *value, int len, int
size, const char *comment, int *status );
```

**Arguments:**

**this**
A pointer to an existing KeyMap.

**key** The key for the new entry.

**value**
A character array containing the concatenated fixed length strings. The length of this array should be at least " size*len" .

**len** The length of each fixed length string.

**size**
The number of fixed length strings in " value" .

**status**
Pointer to the global status variable.

## atlMatchRegion
## Ensure the axes in a Region match those in a Frame

**Description:**

This function checks for matching axes in a supplied Region and Frame. If possible, a new region is created containing a set of axes that correspond in number and type (but not necessarily in specific attributes) to those in the supplied Frame. This means that astConvert should be able to find a Mapping between the supplied Frame and the returned Region. Note, the order of the axes in the returned Region may not match those in the Frame, but astConvert will be able to identify any required re-ordering.

If it is not possible to find a matching Region (for instance, if there are no axes in common between the supplied Region and Frame), an error is reported.

**Invocation:**

```
AstRegion *atlMatchRegion( AstRegion *region, AstFrame *frm, int *status )
```

**Arguments:**

**region**
An AST pointer for the Region to be modified.

**frm** An AST pointer for a Frame to be matched.

**status**
The global status.

**Returned Value:**

**An AST pointer for a Region that matches Frame, or NULL if**

**an error occurs.**

**Notes:**

- If " frm" is a FrameSet, the current Frame is checked first. If no match is found, each other Frame is checked in turn, finishing with the base Frame. If any of these Frames match, it is left as the current Frame in the FrameSet on exit.

## atlPairAxes
## Find corresponding axes in a pair of Frames or FrameSets

**Description:**

For each axis in the base Frame of " From" , this function finds the index of the most closely aligned axis in the base Frame of " to" , (or the current Frame of " from" if " to" is not supplied), and returns these indices.

**Invocation:**

```
void atlPairAxes( AstFrameSet *from, AstFrameSet *to, double *p, const char *domainlist,
int *axes, int *status )
```

**Arguments:**

**from**
An AST pointer to the first FrameSet.

**to**
An AST pointer to the second FrameSet. An error is reported if it is not possible to align the two FrameSets using astConvert. If NULL, then the returned axis indices are the indices of the corresponding current axes in " from" .

**p**
The axis values of a point within the base Frame of " from" at which the pairing is to be determined. None of the supplied axis values should be zero.

**domainlist**
A list of domain names that define the prefered alignment Frames. This list is used by astConvert to align the two FrameSets. Only used if " to" is not NULL.

**axes**
The length of this array should be equal to the number of base Frame axes in " from" . Each returned value will be the one-based index of the corresponding axis in the base Frame of " to" (or the current Frame of " from" if " to" is not supplied), or zero if no corresponding axis can be found.

**status**
The global status.

## atlReadFile
## Reads an AST Object from a text file

**Description:**

This function creates an AST Object by reading the contents of a given text file. The file should contain a dump of an AST Object such as produced by the " atlShow" function.

**Invocation:**

```
AstObject *atlReadFile( const char *fname, const char *options, int *status )
```

**Arguments:**

**fname**

The file name.

**options**

Optional attribute settings for the Channel used to read the file.

**status**

Pointer to the global status variable.

**Returned Value:**

**A pointer to the Object read form the file, or NULL if no Object**

**could be read. A NULL pointer is also returned if an error occurs.**

## atlReadTable
## Create an AST Table from a text file

**Description:**

This function creates an AST Table by reading the contents of a given text file. The text file format matches that of TOPCAT's "ASCII" format, except that any comment lines before the first row of column values that are of the form "# name = value" or "! name = value" are used to create Table parameters. Here "name" is a contiguous block of alphanumeric characters, and "value" represents all characters following the equals sign, up to the end of the line, excluding leading and trailing white space. If value is a scalar integer or double, it is stored as such in the Table. Otherwise, it is stored as a string.

In addition, any row that consists just of two or more minus signs, with no leading spaces, is taken to mark the end of the catalogue. Any subsequent lines in the file are assumed to form a whole new table that is read in exactly the same way as the first. This second Table is stored as a parameter of the first Table using the key "SubTable".

**Invocation:**

```
AstTable *atlReadTable( const char *fname, int *status )
```

**Arguments:**

**fname**

The file name.

**status**

Pointer to the global status variable.

**Returned Value:**

**A pointer to the Table read from the file, or NULL if an error occurs.**

**Notes:**

- All columns in the returned Table will hold scalar values.

## atlShow
## Dumps an AST Object to a text file

**Description:**

This function dumps the supplied AST Object to a new text file with the given name. It may be read back from the file using astReadFile.

**Invocation:**

```
void atlShow( AstObject *this, const char *fname, const hcar *options, int *status
)
```

**Arguments:**

**this**

A pointer to the Object.

**fname**

The file name.

**options**

Optional attribute settings for the Channel used to create the dump.

**status**

Pointer to the global status variable.

## atlTableLutMap
## Create a LutMap from a column of an AST Table

**Description:**

This function creates a LutMap containing the values stored in a specified column of a Table.

**Invocation:**

```
AstLutMap *atlTablelutMap( AstTable *table, const char *column, int *status )
```

**Arguments:**

**table**

The Table.

**column**

The name of the column to use. The column must hold scalar numerical values.

**status**

Pointer to the global status variable.

**Returned Value:**

**A pointer to the LutMap. The input value corresponds to (one-based)**

**row number in the table, and the output value corresponds to the value**

**of the requested column.**