AXL
ASAS™ Interface for Microsoft® Excel®

Version 12.1

ANSYS, Inc.
Southpointe
275 Technology Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Revision Information

The information in this guide applies to all ANSYS, Inc. products released on or after this date, until superseded by a newer version of this guide. This guide replaces individual product installation guides from previous releases.

Copyright and Trademark Information

Disclaimer Notice

ANSYS, Inc. is certified to ISO 9001:2008

U.S. Government Rights

Third-Party Software

Published in the U.S.A.

AXL User Manual

Update Sheet for Version 12.1

November 2009

Modifications:

The following modifications have been incorporated:

Section	Page(s)	Update/Addition	Explanation

Table of Contents

1. Overview

1.1 Introduction

AXL is an interface from ASAS™ to Microsoft® Excel®. It is intended to facilitate the development of Microsoft Excel spreadsheets that require data from ASAS databases. The interface consists of a series of Excel functions that can be accessed within the spreadsheet to recover data directly from the ASAS database. A typical example might be an application that tabulates forces in a particular member for several load cases and then undertakes a post-processing exercise.

Key features are:

- Direct access to the ASAS database information using Excel Function calls
- Recovery of element forces using element number and load case
- Recovery of nodal displacements/reactions using node number and load case
- Lower level functionality for recovery of keyed data from the ASAS database
- Error trapping
- Simple installation
- Cross-platform interrogation of ASAS databases

1.2 Compatibility

AXL is available for Excel variants designed to run under Microsoft Windows.

The interface program does not require an ASAS dongle or security file to operate.

The interface has been designed to allow access to some non-PC versions of the ASAS database. At present databases created on SUN, HP and SG workstations are automatically identified by the interface functions and provide full extraction capabilities. Access to ASAS databases on other machine types will be added in due course.

2. Installation

2.1 Installing AXL

The notes below relate to Microsoft® Excel® 2007. Earlier versions may vary slightly from the suggested installation.

To use the AXL functions in Excel, it needs to know the location of the XLA file and to access the axl32.dll file:

axl32.dll: This file is located within the installed Ansys files (default location: C:\Program Files\Ansys Inc\<version>\asas\bin\<platform>). It must either be copied to a location on the system path or have the above folder added to the path. You may need additional permissions to perform either of these actions and hence we recommend that you consult your IT department.

axl32.xla: In Excel, select Office – Excel Options – Add-Ins; then browse to the location of the axl32.xla file; by default this is in the axl sub folder of the ASAS™ installation. Next, select Office – Excel Options – Add-Ins – Manage Excel Add-ins and ensure the Axl32 box is ticked. Ensure that these settings are saved before closing Excel.

If you are presented with options to update links when opening an existing spreadsheet, select "Don't Update" then copy the path preceding an AXL command, including apostrophes and the exclamation mark, and carry out a Find and Replace with no entry for the Replace to remove it. Save and close your file, and this message shouldn't be shown again when re-opening, unless the link is modified (this may occur, if you transfer the file to another machine and the path to the XLA file is different).

If you are shown this repeatedly, it is likely that the links are not being updated correctly, so check the path to the link in Edit – Links, ensure that only one axl32.xla file is installed on your machine, then update the location via Tools – Add-Ins.

Test the installation by entering "=AxlVersion()" into a cell to ensure that you have the correct version installed. If the result cannot be evaluated, then it is likely that the axl32.dll is not located on the system path.

3. Running AXL

3.1 Using AXL Functions

AXL is reasonably simple to use as follows:

- Start Excel® as usual
- If the AXL32.XLA worksheet file is not auto-loaded at startup, open the AXL32.XLA file (using the normal Excel File Open command)
- Use the axl.... functions in a standard Excel worksheet

When the AXL32.XLA file is loaded nothing appears on the screen. This is normal. If the file is not loaded, any worksheet formulae using axl.... functions will display as #REF!. If a function is used and #VALUE! is displayed, text has probably been entered for an argument which should be a number or vice versa. If #N/A appears, one of the arguments has probably been omitted.

AXL functions are included in formulae just like ordinary Excel functions (such as sine, sum, max, etc.). If you cannot remember the order or meaning of the arguments to the function, use Excel's function wizard - the AXL functions are listed in a category called ASAS-Excel.

A simple example is shown below:

```
=axldisplace("C:\ALPHA;PLAT;JACK",2020,401,"X")
```

This function returns the X displacement for node 2020 in load case 401 for the specified structure JACK, in project PLAT, stored in directory C:\ALPHA.

To avoid typing such long formulae, it is a good idea to use absolute and relative row and column references in Excel as much as possible. This is illustrated in the example below where the user requires a table of X, Y and Z displacements for a set of nodes for one loadcase on one model. The directory, project and model name are entered into cells A1, A2 and A3 (splitting these up makes changing to a different model easy). These items are then concatenated with semicolons into a single string in cell A4. The load case number is entered into cell B6. Axis names, X, Y and Z, are entered into cells B8, C8 and D8. Node numbers are entered into cells A9, A10, A11, A12, A13 and A14.

The displacement for cell B9 in the table is, as above:

```
=axldisplace("C:\ALPHA;PLAT;JACK",2020,401,"X")
```

which would be entered as a formula:

```
=axldisplace(A4,A9,B6,B8)
```

This is much shorter and can be entered by clicking on the appropriate cells. However, if this formula is copied to from cell B9 to cell C10, it becomes:

```
=axldisplace(B5,B10,C7,C9)
```

which is completely wrong! The required formula in cell C10 is:

```
=axldisplace(A4,A10,B6,C8)
```

Column and row numbers in a formula can be made absolute by prefixing them with dollar signs. Therefore the formula in cell B9 can be rewritten as:

```
=axldisplace($A$4,$A9,$B$6,B$8)
```

which when copied to C10 gives:

```
=axldisplace($A$4,$A10,$B$6,C$8)
```

which evaluates to:

```
=axldisplace("C:\ALPHA;PLAT;JACK",2030,401,"Y")
```

as required.

Use of named cells can also be useful when requiring invariant data access. By naming cell A4 ‘model’, use can be made of this name in the function cells and behaves as though giving an absolute address. Thus ‘model’ is synonymous with a4.

The formula in cell B9 becomes:

```
=axldisplace(model,$A9,$B$6,B$8)
```

The formula in cell B9 can be copied over the whole of the table B9:D14 to give the required table as follows:

	A:	B:	C:	D:
1:	C:\ALPHA			
2:	PLAT			
3:	JACK			
4:	=A1&";"&A2&";"&A3			
5:				
6:	Loadcase	401		
7:				
8:	Node	X	Y	Z
9:	2020	=axldisplace(model,$A9,$B$6,B$8)		
10:	2030			
11:	2040			
12:	3020			
13:	3030			
14:	3040			

4. AXL Functions

4.1 Functional Interface

The ASAS™ interface for Excel® consists of two main parts:

- A dynamic link library called axl32.dll
- An Excel® add-in macro sheet called axl32.xla

The dynamic link library (or dll) is a set of functions written in the C language. These functions understand enough about the ASAS file system to extract information from it. They do all the hard work.

The add-in macro sheet provides a set of spreadsheet functions that use the dynamic link library to access the ASAS™ file system. These spreadsheet functions can be used like any other functions in an Excel spreadsheet (or 'worksheet' in Microsoft®'s terminology). All the functions have names starting with the prefix *axl*...

The functions are designed to be relatively straightforward to use. All arguments are in terms of user numbers (user element number, user loadcase number, etc.) rather than ASAS internal numbers.

The AXL interface requires no special configuration, no start up macros, etc.

4.2 Function Description

There are two main classes of function:

- functions which retrieve information or results for a particular item from an ASAS model
- functions which retrieve a value from a record in an ASAS file

Functions in the first category are generally the most useful. The second category is intended for examining the contents of the ASAS file system, and assumes that the user knows what to expect on each record.

There are also a few miscellaneous utility functions that are described separately.

All the function information in this section can be obtained using the help button from the Function Wizard in Excel.

The list of functions below is complete at the time of writing, but more functions will be added as and when necessary.

Model Description

Many of the functions start with an argument called model in the descriptions below. This argument defines the ASAS project, the structure or component being investigated, and the directory in which the files are located. It is a string consisting of three parts, separated by semi-colons:

directory;project;structure/component tree

so, for example:

```
C:\USER;PROJ;PILE JACO
```

would specify a component with the assembled name JACO in structure PILE which is in an ASAS project called PROJ with all the files stored in a directory called \USER on drive C:. The best way of entering this data is to enter it into a single cell of the spreadsheet, then refer to this cell in each call of an AXL function. Use the $ symbol to make this an absolute cell reference, or name the cell to make it easier. The purpose of specifying the model in each call is to allow results from different models to be intermixed in one spreadsheet at will.

FUNCTIONS	DESCRIPTION
Functions to Retrieve Standard Information	
axlbeamaxis	Returns beam axis direction cosines
axlbeamforce	Returns the force or moment in a beam element at one end
axlbeamnsteps	Returns number of steps on a beam element
axlbeamoffset	Returns beam element rigid offset information
axlbeamprop	Returns beam element geometric properties
axlbeamsect	Returns section dimension for a beam element
axlbeamsectname	Returns section name for a beam element
axlbeamsecttype	Return section profile type for a beam element
axldisplace	Returns the displacement at a node for a specified load case
axlelementgeomn	Returns element geometric property number
axlelementnode	Returns the value of a node number on an element
axlelementresult	Returns element stress force or moment result
axlelementtype	Returns the ASAS element type
axlfreedomcode	Determines the internal ASAS freedom code corresponding to a named freedom
axlfx, axlqy, axlqz, axlmx, axlmy, axlmz	These six functions return the force or moment at one end of a beam element
axlgetmodel	Returns the component or structure name for the model from the master index
axlgotp	Returns a coordinate of the global overturning point used by ASAS in calculating moments in load resultants
axlgotpx, axlgotpy, axlgotpz	These three functions return x, y or z coordinate of the global overturning point used by ASAS in calculating moments in load resultants
axlgroupelements	Returns the user element number of the elements within the specified group
axlgroupnumber	Returns the user group number of the specified group
axlloadname	Returns the title of the specified loadcase number
axlloadnumber	Returns the user loadcase number of the specified internal loadcase
axlloadtimestep	Returns a time step/eigen number given a model, loadcase and subset
axlmodeltimestep	Returns the time step given a model and a step number
axlnelement	Returns the number of elements in the model
axlnload	Returns the number of loadcases on the model
axlnnode	Returns the number of nodes on the model
axlnode	Returns one of the global coordinates of a specified node
axlnodenumber	Returns the user node number
axlnodeskew	Returns the skew system number at a node
axlnodex, axlnodey, axlnodez	These three functions return the x, y and z coordinates, respectively, of a specified node
axlnskew	Returns the number of skew systems in the model
axlnumberincrements	Returns the number of time steps for a given model
axlnumberofgroupelements	Returns the user element number of the elements within the specified group
axlnumberofgroups	Returns the user group number of the specified group
axlnumbersubsets	Returns the number of subsets for a given loadcase
axlpileforce	Returns the force or moment in a pile at a specified depth
axlpilelength	Returns the length of a pile, calculated as the distance between its top and bottom nodes
axlprogversion	Returns the program name and the version of the run
axlreaction	Returns the reaction at a node for a specified loadcase
axlresultant	Returns the resultant force or moment for a specified loadcase
axlrundate	Returns date and time of the ASAS run
axlruntitle	Returns the title of the ASAS analysis
axlskew	Returns a component of the direction cosines for the specified skew system
axlsubsetnumber	Returns a user subset number given a model, user loadcase and system subset number
axlunits	Returns the units associated with the analysis
axluserelement	Returns the user element number

Functions to Retrieve Response Information	
axlbeamtransientforce	Returns the transient response force or moment in a beam element at one end
axltransientdisplace	Returns the displacement/velocity/acceleration for a node in a transient response analysis
axltransientntimesteps	Returns number of time steps selected for reporting in a transient response analysis
axltransienttimesteps	Returns the time associated with a given step in a transient response analysis
Functions to Retrieve Specific Keyed Items	
axli	Retrieves a single integer value from any file of the specified model, at any position of any record
axlr	Same as axli except that the value returned is real. The position on the record is also calculated in terms of real numbers
axls	Same as axli except that the integer is converted to a text string of four or fewer characters, such as an element name or a component name
axlleni	Returns the length of the record, counter in terms of ASAS integers. Returns zero if the record does not exist
axllenr	Same as axlleni, except that the record length is counted in terms of reals. This does not actually guarantee that the record contains real numbers
axllast	Returns the record number of the last record on an ASAS file
axlruninf	Returns an item from the RUNINF common block
Results Database Functions	
axlgetelement	Returns the requested element result
axlgetelementex	Returns the requested element result, extended for additional result availability
axlgetelementflt	Returns the requested element result, filtered based on the given information
axlgetequation	Returns the requested equation result
axlgetglobal	Returns the requested global result
axlgetnodal	Returns the requested nodal result
Miscellaneous Functions	
axlaxis	Converts an axis or force name to a number
axlerror	Returns a text string describing an error code returned by an AXL function
axlversion	Returns a text string identifying the version of the add-in macro sheet axl32.xla and the associated dynamic link library axl32.dll which are currently being used. This is mainly intended so that support personnel can identify possible problems due to the use of superseded software

4.3 Functions to Retrieve Standard Information

The following functions can be used to retrieve standard items of information from an ASAS database. No understanding of the internal structure of the database is required to make use of these functions.

axlbeamaxis

Purpose: Returns the direction cosine components for beam element local axes.

Syntax: *axlbeamaxis (model, element, axisname)*

Notes: *axisname* corresponds to the required axis direction cosine and may be one of the following:

mnemonic	Description
XX	Direction cosines
XY	for local
XZ	X axis
YX	Direction cosines
YY	for local
YZ	Y axis
ZX	Direction cosines
ZY	for local
ZZ	Z axis

The axis information returned relates to the final member orientation i.e. taking account of any rigid offsets.

Example: `axlbeamaxis(Jacket, 1001, "Zy")` returns the y component of the local Z axis for element 1001 in the model defined by the named cell Jacket.

axlbeamforce

Purpose: Returns the force or moment in a beam element at one end.

Syntax: *axlbeamforce (model, element, node, loadcase, force)*

Notes: *node* may be either one of the nodes on the beam or -1 to indicate end 1 of the beam, or -2 to indicate end 2. force may be given by name (“FX”, “QY”, “QZ”, “MX”, “MY” or “MZ”) or by a number (1 for axial force, 2 for shear in the local Y’ direction of the element, 3 for shear in the local Z’ direction, 4 for torque, 5 for moment about the local Y’ axis, 6 for moment about the local Z’ axis). It may be more convenient to use the shorter functions *axlfx*, *axlqy*, *axlqz*, *axlmx*, *axlmy*, and *axlmz* described below.

Results for non-beam elements may be obtained with the *axlelementresult* function. Results for piles from a SPLINTER analysis may be obtained with the *axlpileforce* function.

Example: `axlbeamforce(Jacket, 2050, -1, 10, "Fx")` returns the axial force at end 1 of element 2050 for loadcase 10 in the model defined by the named cell Jacket.

axlbeamnsteps

Purpose: Returns the number of steps on a beam element.

Syntax: *axlbeamnsteps (model, element)*

Notes: This function can only be used with the engineering beam elements BM2D, BM3D, TUBE and GRIL. Unstepped beams return a value of 0.

Example: `axlbeamnsteps (jacket, 1001)` returns the number of steps associated with element 1001 for the model defined by the named cell Jacket.

axlbeamoffset

Purpose: Returns a beam element rigid offset information.

Syntax: *axlbeamoffset (model, element, end, freedomname)*

Notes: *end* corresponds to which end of the beam the offset is required. Use -1 for end 1 or -2 for end 2. *freedomname* may be either X, Y, or Z representing the offset along the associated local axis of the beam.

If an offset has not been defined a value of zero will be returned.

Example: `axlbeamoffset (jacket, 1001, -2, "x")` returns the axial offset at end 2 of element 1001 in the model defined by the named cell Jacket.

axlbeamprop

Purpose: Returns the flexural properties for a beam element, such as length, area, inertia, etc.

Syntax: *axlbeamprop (model, element, step, property)*

Notes: *step* corresponds to the step number on the beam. For unstepped beams 0 or 1 may be used. The section information at either end of the beam may be obtained using a step number of -1 (for end 1) or -2 (for end 2). The *property* may be one of the following:

mnemonic	Description	Applicable to [2]
L	Element or step length [1]	GRIL, BM2D, BEAM, TUBE
A	Area	GRIL, BM2D, BM3D, BEAM, TUBE
IY	Inertia about Y axis	GRIL, BM3D, BEAM, TUBE
IZ	Inertia about Z axis	BM2D, BM3D, BEAM, TUBE
J	Torsion constant	GRIL, BM3D, BEAM, TUBE
AY	Shear area for Y axis	BM2D, BM3D, TUBE
AZ	Shear area for Z axis	GRIL, BM3D, TUBE

[1] For unstepped beams the length returned is the total physical length of the element (after the application of offsets). For stepped beams the length returned is for the requested step.

[2] Where properties are requested that are not applicable to a particular beam type, a value of zero will be returned.

Example: `axlbeamprop (jacket, 1001, -2, "a")` returns the cross sectional area at end 2 of element 1001 in the model defined by the named cell Jacket.

axlbeamsect

Purpose: Returns a section dimension for a beam element.

Syntax: *axlbeamsect (model, element, step, dimensionname)*

Notes: *step* corresponds to the step number on the beam. For unstepped beams 0 or 1 may be used. The section information at either end of the beam may be obtained using a step number of -1 (for end 1) or -2 (for end 2). *dimensionname* may be one of the following:

mnemonic	Description	Applicable to [2]
D	diameter	TUB
T	thickness	TUB
D	depth	WF, RHS, BOX, TEE, ANGL, CHAN, PRE
B	breadth	WF, RHS, BOX, TEE, ANGL, CHAN, PRE
TW	web thickness	WF, RHS[1], BOX[2], TEE, ANGL[1], CHAN
TF	flange thickness	WF, RHS[1], BOX[3], TEE, ANGL[1], CHAN
R	fillet radius	WF, RHS, CHAN, ANGL, TEE

If section data has not been defined, an error code will be returned. For TUBE elements, however, the dimension information can still be retrieved using this function.

[1] For RHS and ANGL sections, only one thickness is defined. Both TW and TF are set to this value.

[2] For BOX sections TW corresponds to the thickness of the side plates.

[3] For BOX sections TF corresponds to the thickness of the top and bottom plates.

Example: `axlbeamsect (jacket, 1001, -2, "t")` returns the thickness at end 2 of tubular element 1001 in the model defined by the named cell Jacket.

axlbeamsectname

Purpose: Returns the section name associated with a beam element.

Syntax: *axlbeamsectname (model, element, step)*

Notes: *step* corresponds to the step number on the beam. For unstepped beams 0 or 1 may be used. The section information at either end of the beam may be obtained using a step number of -1 (for end 1) or -2 (for end 2).

If section data has not been defined an error code will be returned.

The section name recovered may be up to 12 characters long.

Example: `axlbeamsectname (jacket, 1001, -2)` returns the section name associated with end 2 of element 1001 in the model defined by the named cell Jacket.

axlbeamsecttype

Purpose: Returns the section profile type for a beam element.

Syntax: *axlbeamsecttype (model, element, step)*

Notes: *step* corresponds to the step number on the beam. For unstepped beams 0 or 1 may be used. The section information at either end of the beam may be obtained using a step number of -1 (for end 1) or -2 (for end 2).

If section data has not been defined an error code will be returned.

Profiles returned will be one of the following:

mnemonic	Description
WF	Wide Flange
TUB	Tube
BOX	Fabricated Box
RHS	Rolled Hollow Section
TEE	Tee
ANGL	Angle Section
CHAN	Channel
PRI	Prismatic

Example: `axlbeamsecttype (jacket, 1001, -2)` returns the section type at end 2 of element 1001 in the model defined by the named cell Jacket.

axldisplace

Purpose: Returns the displacement at a node for a specified load case.

Syntax: *axldisplace (model, node, loadcase, freedom)*

Notes: The displacement may be in any of the applicable ASAS freedoms. This may be expressed as a name (e.g. “X”, “RX”) or as a number (1, 2, ...). If the node is skewed, the displacements are in the skewed directions.

Example: `axldisplace(Jacket, 1010, 10, "x")` returns the x displacement at node 1010 for loadcase 10 in the model defined by the named cell Jacket.

axlelementgeomn

Purpose: Returns the geometric property number associated with an element.

Syntax: *axlelementgeomn (model, element)*

Notes: Information cannot be retrieved from the database for specific geometric property numbers, but this function may be useful for checking data input cross references.

Example: `axlelementgeomn(Jacket, 1001)` returns the geometric property number for element 1001 in the model defined by the named cell Jacket.

axlelementnode

Purpose: Returns the value of a node number on an element. Nodes are returned in the same order as they are defined in the ASAS data.

Syntax: *axlelementnode (model, element, index)*

Notes: *index* is the position of the node in the element definition, as given in the original ASAS data.

Example: `axlelementnode(Jacket, 2050, 2)` returns the number of the second node on element 2050 in the model defined by the named cell Jacket.

axlelementtype

Purpose: Returns the ASAS™ element type.

Syntax: *axlelementtype (model, element)*

Notes: This function returns the four character name by which an element type is identified within the ASAS system e.g TUBE, BM3D, QUM4, etc.

Example: `axlelementtype(Jacket, 1001)` returns the ASAS element type for user element 1001 in the model defined by the named cell Jacket.

axlelementresult

Purpose: Returns one force, stress, etc. result for any type of element.

Syntax: *axlelementresult (model, element, loadcase, result)*

Notes: All the results for one load case on an element are stored as a list of numbers by ASAS. It is necessary to know the order of the results and the node numbering of the element to extract the required result. The *axlbeamforce* (or *axlfx* etc.) functions should be used for retrieving results from the standard three-dimensional beam elements.

Example: `axlelementresult(vessel, 2050, 10, 13)` returns the σ_{xx} stress on the third node of a GCS8 element number 2050 for loadcase 10 in the model defined by the named cell vessel.

axlfreedomcode

Purpose: Determines the internal ASAS™ freedom code corresponding to a named freedom. Returns zero if the freedom name is invalid. This function is generally not required as most AXL functions can convert freedom names to freedom codes when required.

Syntax: *axlfreedomcode (model, freedomname)*

Notes: If freedom X, Y, Z, RX, RY or RZ is specified, these are converted to codes 1, 2, 3, 4, 5 or 6 without accessing the ASAS™ files. Hence, this is one of the few AXL functions which may return a valid answer when an invalid model is specified.

Example: `axlfreedomcode(Jacket, "x")` returns the freedom code 1

axlfx, axlqy, axlqz, axlmx, axlmy, axlmz

Purpose: These six functions return the force or moment at one end of a beam element.

Syntax: *axlfx (model, element, node, loadcase)*

axlqy (model, element, node, loadcase)
axlqz (model, element, node, loadcase)
axlmx (model, element, node, loadcase)
axlmy (model, element, node, loadcase)
axlmz (model, element, node, loadcase)

Notes: *node* may be either one of the nodes numbers on the beam or -1 to indicate end 1 of the beam, or -2 to indicate end 2. These functions are similar to *axlbeamforce*, described above.

Example: `axlmx(Jacket, 2050, -2, 10)` returns the torsion moment at end 2 of element 2050 for loadcase 10 in the model defined by the named cell Jacket.

axlgetmodel

Purpose: Returns a component or structure name stored in the project file by reference to an index from 1 to the number of models in the project.

Syntax: *axlgetmodel (Directory, Project, index)*

Notes: *Directory* is the full path name of the directory in which the project file is stored, eg:

```
"C:\CENTURY"
```

Project is the four character identifier of the ASAS project. *index* is the position in the project file of the structure or component name being retrieved. The index should be a number between 1 and the number of components/structures in the project. An entry for a component does not imply that results exist for any instance of the component.

Example: `axlgetmodel("C:\CENTURY", "PROJ", 2)` returns the font character name of the structure or component stored in entry 2 of the project PROJ residing in the directory C:\CENTURY.

axlgotp

Purpose: Returns a coordinate of the global overturning point used by ASAS in calculating moments in load resultants.

Syntax: *axlgotp (model, axis)*

Notes: *axis* may be specified by name (“X”, “Y” or “Z”) or by number (1, 2 or 3).

Example: `axlgotp(Jacket, "x")` returns the x coordinate of the global overturning point for the model defined by the named cell Jacket.

axlgotpx, axlgotpy, axlgotpz

Purpose: These three functions return x, y or z coordinate of the global overturning point used by ASAS in calculating moments in load resultants.

Syntax: *axlgotpx (model)*

Notes: These functions may be used in place of calls to *axlgotp*.

Example: `axlgotpx(Jacket)` returns the same information as `axlgotp(Jacket, "x")` above.

axlgroupelements

Purpose: Returns the user number of an elements within a group.

Syntax: *axlgroupelements (model,group,element)*

Notes: Group should be the user group number and element the element index (1,2,3…). A zero value for group indicates the unspecified elements.

Example: `axlgroupelements(jacket, 4, 2)` returns the user number of the second element within group 4 associated with the model defined by the named cell jacket.

axlgroupnumber

Purpose: Returns the user number of a group.

Syntax: *axlgroupnumber (model,group)*

Notes: Group should be the group index number. A zero value for the user group number indicates the group which consists of the unspecified elements.

Example: `axlgroupnumber(jacket, 3)` returns the user number of the third group within associated with the model defined by the named cell jacket.

axlloadname

Purpose: Returns the title of the specified loadcase number.

Syntax: *axlloadname (model, loadcase)*

Notes: *loadcase* is the user defined loadcase number.

Example: `axlloadname(Jacket, 10)` returns the title of loadcase 10 in the model defined by the named cell Jacket.

axlloadnumber

Purpose: Returns the user loadcase number of the specified internal loadcase.

Syntax: *axlloadnumber (model, internalloadcase)*

Notes: ASAS internally numbers its loadcases 1, 2, 3, ... in the order in which they are defined. No other AXL functions require the user to know the internal load case number. The purpose of this function is to allow users to generate a list of all the load cases on the model by creating a table with one column containing the sequence 1, 2, 3, ... and the next column using this function to determine the ASAS load case number. See also *axlnload* and *axlresultant* below.

Example: `axlloadnumber(Jacket, 2)` returns the user loadcase number corresponding to the second loadcase for the model defined by the named cell Jacket.

axlloadtimestep

Purpose: Returns a time step/eigen number given a model, loadcase and subset. For an ASAS (L) or ASAS (Non-linear) frequency analysis, the value is the frequency f (Hz). For ASAS (Non-linear) buckling or spectral runs, the value is λ, the eigenvalue.

Syntax: *axlloadtimestep (model, load, step)*

Notes: *load is the* loadcase number of the results from a transient or eigenvalue analysis where step is the required subset number or step.

Example: `axlloadtimestep(Jacket, 115, 3)` returns the time or eigenvalue of the 3rd subset or step of loadcase number 115 for the model defined by Jacket.

axlmodeltimestep

Purpose: Returns a time step given a model and step number.

Syntax: *axlmodeltimestep (model, step)*

Notes: On occasion a model may produce load cases which represent different time steps of the solution or similarly one loadcase and many subsets, this function allows the retrieval of the time for each step when. This function should only be used when a model contains only one transient or eigenvalue solution.

Example: `axlmodeltimestep(Rig, 53)` returns the time or eigenvalue of the 53rd subset or step of the model defined by Rig.

axlnelement

Purpose: Returns the number of elements in the model.

Syntax: *axlnelement (model)*

Notes: This function will exclude any components that have been defined in the model.

Example: `axlnelement(Jacket)` returns the number of elements in the model defined by the named cell Jacket.

axlnload

Purpose: Returns the number of loadcases on the model.

Syntax: *axlnload (model)*

Notes: May be useful used in conjunction with *axlloadnumber* above to generate a list of user loadcase numbers.

Example: `axlnload(Jacket)` returns the number of loadcases in the model defined by the named cell Jacket.

axlnnode

Purpose: Returns the number of nodes on the model.

Syntax: *axlnnode (model)*

Notes: Nodes are counted on the basis of generating equations in the stiffness matrix, so nodes which are only used as guide points to define local axes or skew systems are not included.

Example: `axlnnode(Jacket)` returns the number of structural nodes in the model defined by the named cell Jacket.

axlnode

Purpose: Returns one of the global coordinates of a specified node.

Syntax: *axlnode (model, nodenumber, axis)*

Notes: *axis* may be specified by name (“X”, “Y” or “Z”) or by number (1, 2 or 3).

Example: `axlnode(Jacket, 1010, "y")` returns the y global coordinate for node 1010 in the model defined by the named cell Jacket.

axlnodenumber

Purpose: Returns the user node number by reference to an index in increasing node number order.

Syntax: *axlnodenumber (model, index)*

Notes: *index* alludes to the reference to a given node within a list of ascending user node numbers. *index* should be between 1 and the number of nodes in the model, see *axlnnode*.

Example: `axlnodenumber(Jacket, 216)` returns the node number at position 216 of the sorted node number list for the model defined by the named cell Jacket.

axlnodeskew

Purpose: Returns the skew system number at a node.

Syntax: *axlnodeskew (model, node)*

Notes: If the node is not skewed (or does not exist) returns zero. The direction cosines of the skew system may be obtained with the *axlskew* function.

Example: `axlnodeskew(Jacket, 1020)` returns the skew number associated with node number 1020 in the model defined by the named cell Jacket.

axlnodex, axlnodey, axlnodez

Purpose: These three functions return the x, y and z coordinates, respectively, of a specified node.

Syntax: *axlnodex (model, nodenumber)*
axlnodey (model, nodenumber)
axlnodez (model, nodenumber)

Notes: These functions may be used in place of calls to *axlnode* (*model, nodenumber, axis*).

Example: `axlnodey(Jacket, 1010)` returns the y global coordinate for node 1010 in the model defined by the named cell Jacket.

axlnumberincrements

Purpose: Returns the number of time steps within a model.

Syntax: *axlnumberincrements (model)*

Notes: On occasion a model may produce load cases which represent different time steps of the solution or similarly one loadcase and many subsets, this function allows the retrieval of the number of time steps with the model.

Example: `axlnumberincrements(Jacket)` returns the number of time steps or eigenvalues within the model defined by Rig.

axlnumberofgroups

Purpose: Returns the number of groups that a structure contains.

Syntax: *axlnumberofgroups (model)*

Notes: The number returned includes the group containing the unspecified elements.

Example: `axlnumberofgroups(jacket)` returns the number of groups associated with the model defined by the named cell jacket.

axlnumberofgroupelements

Purpose: Returns the number of elements within a group.

Syntax: *axlnumberofgroupelements (model,group)*

Notes: Group should be the user group number. A zero value for group indicates the unspecified elements.

Example: `axlnumberofgroupelements(jacket, 4)` returns the number of elements within group 4 associated with the model defined by the named cell jacket.

axlnskew

Purpose: Returns the number of skew systems in the model.

Syntax: *axlnskew (model)*

Notes: The direction cosines for each skew system may be retrieved with the *axlskew* function.

Example: `axlnskew(Jacket)` returns the number of skew systems in the model defined by the named cell Jacket.

axlnumbersubsets

Purpose: Returns the number of subsets which constitute a loadcase.

Syntax: *axlnumbersubsets (mode,loadcasel)*

Notes: The function returns the number of subsets within a loadcase of a model, these subsets may indicate different time steps or eigenvalues from RESPONSE or ASAS (Non-linear).

Example: `axlnumbersubsets(Pile, 4)` returns the number of subsets contained within the forth load of the model defined by `Pile`.

axlpileforce

Notes: Returns the force or moment in a pile at a specified depth.

Syntax: *axlpileforce (model, element, depth, loadcase, force)*

Notes: *force* may be given by name (“FX”, “QY”, “QZ”, “MX”, “MY” or “MZ”) or by a number (1 for axial force, 2 for shear in the local Y’ direction of the element, 3 for shear in the local Z’ direction, 4 for torque, 5 for moment about the local Y’ axis, 6 for moment about the local Z’ axis).

depth is the distance downwards along the pile from its top node (not the mudline, unless coincident). If this does not correspond to a sub-element division on the pile, the nearest results will be returned, without interpolation.

SPLINTER results recovery requires a SAVE 7 FILES 13, 15, 17, 29, 32, 35, 56 in the SPLINTER analysis.

Example: `axlpileforce(Pilemodel, 80, 12.0, 10, ”my”)` returns the y bending moment at a depth of 12m from the top of pile number 80 for loadcase 10 in the model defined by the named cell Pilemodel.

axlpilelength

Purpose: Returns the length of a pile, calculated as the distance between its top and bottom nodes.

Syntax: *axlpilelength (model, element)*

Notes: SPLINTER results recovery requires a SAVE 7 FILES 13, 15, 17, 29, 32, 35, 56 in the SPLINTER analysis.

Example: `axlpilength(Pilemodel, 80)` returns the length of pile number 80 in the model defined by the named cell Pilemodel.

axlprogversion

Purpose: Returns the program name and version information associated with a particular analysis.

Syntax: *axlprogversion (model)*

Notes: This function returns a description in the form “ASAS 12.00.01.0”.

Example: `axlprogversion(Jacket)` returns the program information for the model defined by the named cell Jacket.

axlreaction

Purpose: Returns the reaction at a node for a specified loadcase.

Syntax: *axlreaction (model, node, loadcase, freedom)*

Notes: The reaction may be for any of the applicable ASAS freedoms. This may be expressed as a name (e.g. “X”, “RX”) or as a number (1, 2, ...). If the node is skewed, the reactions are in the skewed directions.

Example: `axlreaction(Jacket, 1010, 10, "x")` returns the reaction in the x direction at node 1010 for loadcase 10 in the model defined by the named cell Jacket.

axlresultant

Purpose: Returns the total resultant force or moment for a specified load case.

Syntax: *axlresultant (model, loadcase, axis)*

Notes: The total load resultant is that calculated by ASAS during the data check. Hence it will be inaccurate if the resultant load calculated during the data check is inaccurate. The axis may be specified by name (“X”, “RY”, etc.), by force (“FX”, “MY”, etc.) or by number (1 to 6).

Example: `axlresultant(Jacket, 1010, "x")` returns the load resultant in the global x direction for loadcase 1010 in the model defined by the named cell Jacket.

axlrundate

Purpose: Returns the date and time for a given analysis.

Syntax: *axlrundate (model)*

Notes: This function returns a 17 character text string containing the date and time in the form “08.41 1-JUL-97”.

Example: `axlrundate(Jacket)` returns the date and time information for the model defined by the named cell Jacket.

axlruntitle

Purpose: Returns the title used for a given analysis.

Syntax: *axlruntitle (model)*

Notes: This function returns an 81 character text string containing the title in the form “Example 2.1 Extreme Wave Analysis”.

Example: `axlruntitle(Jacket)` returns the title for the model defined by the named cell Jacket.

axlskew

Purpose: Returns a component of the direction cosines for the specified skew system.

Syntax: *axlskew (model skewsystem, index)*

Notes: *index* selects which component of the direction cosines for the skew system is returned, where 1 = X'X, 2 = X'Y, 3 = X'Z, 4 = Y'X, 5 = Y'Y, 6 = Y'Z, 7 = Z'X, 8 = Z'Y, 9 = Z'Z. The final three values are not stored by ASAS™, but are calculated from the stored values by this function.

Example: `axlskew(Jacket, 12, 4)` returns the x component of the direction cosine defining the y axis of the skew system number 12 in the model defined by the named cell Jacket.

axlsubsetnumber

Purpose: Returns the user number of a subset.

Syntax: *axlsubsetnumber (model,loadcase,subset)*

Notes: The function returns the user number of a subset denoted by its model, load and system subset number .

Example: `axlsubsetnumber(Jacket, 6, 2)` returns the user subset number of the second subset contained within the sixth load of the model defined by Jacket.

axlunits

Purpose: Returns the units associated with a particular analysis.

Syntax: *axlunits (model, unitname)*

Notes: *unitname* refers to the particular unit type required and may be one of the following:

FORCE
LENGTH
ROTATION
MASS
TEMPERATURE
TIME

Abbreviations are permitted e.g. F, FOR, FORCE will all retrieve the force unit.

If units were not used in the analysis an appropriate error message is returned.

The function returns a 13 character unit name.

Example: `axlunits(Jacket, "length")` returns the length unit the model defined by the named cell Jacket.

axluserelement

Purpose: Returns the user element number by reference to an index in increasing user element order.

Syntax: *axluserelement (model, index)*

Notes: *index* alludes to the reference to a given element within a list of ascending user element numbers. *index* should be between 1 and the number of elements in the model, see *axlnelement*. Components are excluded from this list.

Example: `axluserelement(Jacket, 115)` returns the user element number at position 115 of the sorted user element list for the model defined by the named cell Jacket.

4.4 Functions to Retrieve Response Information

The following functions can be used to retrieve information from a database created by RESPONSE. It should be noted that to access this information the following SAVE commands must be included in the RESPONSE data file:

```
SAVE LOCO FILES
SAVE 2 FILE 3 10
```

axltransientntimesteps

Purpose: Returns the number of timesteps for a given model.

Syntax: *axltransientntimesteps (model)*

Notes: The number of time steps returned corresponds to the times given as part of the RESULTS Output TIME command, it does not refer to the times at which the transient analysis has been undertaken.

Example: `axltransientntimesteps(Jacket)` returns the number of timesteps for the named cell Jacket.

axltransienttimesteps

Purpose: Returns the time associated with a given time step.

Syntax: *axltransienttimesteps (model, timestep)*

Notes: refers to a given time within a list of ascending results time steps. The time is returned in seconds.

Example: `axltransienttimesteps(Jacket,2)` returns the time of the second recorded timestep for the named cell Jacket.

axltransientdisplace

Purpose: Returns the displacement, velocity or acceleration associated with a given time step and node.

Syntax: *axltransientdisplace (model, node, timestep,freedomname,resulttype)*

Notes: refers to a given time within a list of ascending results time steps. The freedom name is expressed as a name (e.g. “X”, “RX”). The *resulttype* is either “displacement”, “velocity” or “acceleration”.

Example: `axltransientdisplace(Rig,2,1,"X","velocity")` returns the velocity of the second node of the model Rig at the first time step.

axlbeamtransientforce

Purpose: Returns a beam force or moment for a given time step, element and node.

Syntax: *axlbeamtransientforce(model,element,node,timestep,forcename)*

Notes: refers to a given time within a list of ascending results time steps. *node* can be either one of the nodes on the beam or -1 to indicate end 1 of the beam, or -2 to indicate end 2. The force name is expressed as a name (e.g. “FX”, “MX”) or by a number (1 for axial force, 2 for shear in the local Y direction of the element, 3 for shear in the local Z direction, 4 for torque, 5 for moment about the local Y axis, 6 for moment about the local Z axis).

Example: `axlbeamtransientforce(Jacket,5,-1,3”FX”)` returns the force in the x direction at the end 1 of the beam for load case number 5 of the model named Jacket.

4.5 Functions to Retrieve Specific Keyed Items of Data from an ASAS Database

The following functions can be used to extract any data from any file of the ASAS file system. To use them effectively, you must have an understanding of the internal structure of the ASAS file system and detailed information about the contents of the relevant files. This information is contained in Volume 1 of the ASAS Programmer's Manual.

axli

Purpose: Retrieves a single integer value from any file of the specified model, at any position of any record.

Syntax: *axli (model, filenumber, recordnumber, position)*

Notes: Extreme care must be exercised when extracting information from mixed integer and real records if the ASAS files are stored on a remote computer running a different operating system to the host machine. The user is referred to the ASAS Programmer's Manual for detailed descriptions of the ASAS file system.

Example: `axli(Jacket, 13, 5, 7)` returns the 7th integer from record 5 of file 13 for the model defined in the named cell Jacket. This corresponds to the user element number for system element 3 (=(5+1)/2 since there are two records per element).

axlr

Purpose: Same as *axli* except that the value returned is real. The position on the record is also calculated in terms of real numbers.

Syntax: *axlr (model, filenumber, recordnumber, position)*

Notes: Extreme care must be exercised when extracting information from mixed integer and character records if the ASAS files are stored on a remote computer running a different operating system to the host machine. The user is referred to the ASAS Programmer's Manual for detailed descriptions of the ASAS file system.

Example: `axlr(Jacket, 13, 6, 2)` returns the 2nd real from record 6 of file 13 for the model defined in the named cell Jacket. This corresponds to the y coordinate of the first node for system element 3 (=6/2, since there are two records per element).

axls

Purpose: Same as *axli* except that the integer is converted to a text string of 4 or fewer characters, such as an element name or a component name.

Syntax: *axls(model, filenumber, recordnumber, position)*

Notes: Extreme care must be exercised when extracting information from mixed integer and real records if the ASAS files are stored on a remote computer running a different operating system to the host machine. The user is referred to the ASAS Programmer's Manual for detailed descriptions of the ASAS file system.

Example: `axls(Jacket, 13, 5, 7)` returns the character string stored at the 7th position of record 5 of file 13 for the model defined in the named cell Jacket. For a component this

corresponds to the assembled name for the component stored in position 3 of the element list (=(5+1)/2, since there are two records per element or component).

axlleni

Purpose: Returns the length of the record, counted in terms of ASAS integers. Returns zero if the record does not exist.

Syntax: *axlleni (model, filenumber, recordnumber)*

Notes: For real records use *axllenr*. If mixed integer and real records are addressed, the number of integers is predefined (and is given in the ASAS Programmer's Manual). The number of reals can then be determined by the following formula:

$$\text{numberReals} = \frac{\text{Integer Length of Record} - \text{Number Integers}}{\text{Int To Real}}$$

where:

Integer Length of Record	is returned from *axlleni*
Number Integers	is obtained from the Programmer's Manual
Int to Real	is the number of integer values in a real. For MOST machines this is 2 (the exception being the Cray, where it is 1)

Example: `axlleni(Jacket, 35, 13)` returns 120, the length of record 13 of file 35 for the model defined in the named cell Jacket.

axllenr

Purpose: Same as *axlleni*, except that the record length is counted in terms of reals. This does not actually guarantee that the record contains real numbers.

Syntax: *axllenr (model, filenumber, recordnumber)*

Notes: For integer or mixed integer/real records use *axlleni*.

Example: `axllenr(Jacket, 5, 2)` returns the length (in number of reals) of record 2 of file 5. This is the geometric properties data.

axllast

Purpose: Returns the record number of the last record on an ASAS file.

Syntax: *axllast (model, filenumber)*

Notes: This can be useful in determining how much information there is to retrieve from the database.

Example: `axllast(Jacket, 35)` returns the number of the last record on file 35, the IADMIN file, for the model defined in the named cell Jacket.

axlruninf

Purpose: Returns an item from the RUNINF common block (stored on record 13 of file 35, the IADMIN file).

Syntax: *axlruninf (model, item)*

Notes: The RUNINF common block stores many of the run time counters, such as number of elements, loadcases, etc. Some of these have explicit functions, such as *axlnelement*, but other values can only be obtained using *axlruninf*. The user is referred to the ASAS Programmer's Manual for detailed descriptions of the ASAS file system. Item can either be the position within RUNINF, or may refer to the name of the variable, as defined in the ASAS Programmer's Manual. Names may be given in upper or lower case.

Example:
```
axlruninf("Jacket", "NELEM")
```
or
```
axlruninf("Jacket", 11)
```
both return item number 11 in the /RUNINF/ common block, which is called NELEM and is the number of elements in the model.

4.6 Results Database Functions

With the introduction of the Unified Results Storage Database for ASAS Version 14, additional functions have been implemented with AXL to facilitate the retrieval of any data within the database. These functions require the model, load, subset, element and node number, as input and the result type and component must be specified. The valid result type and components are specified in the ASAS Database User Manual, and depend upon what programs have been run. The result type and component cannot be more than 20 and 8 characters long respectively. The 'model' information comprises the path, job and structure names, concatenated into a single string, separated by semicolons, for example C:\Models;Job1;STRA There are four general functions which return element, equation, global and nodal results, all of which read exclusively from the physical 45 file. Therefore within the ASAS (Linear), ASAS (Non-linear), etc. data file the option RESU must be specified.

axlgetelement

Purpose: Returns the requested element result.

Syntax: *axlgetelement (model, element, load, subset, result type, result component, surface, node)*

Notes: The result type and result component indicate which information will be retrieved. For example the result type may be "stress" and the result component "sxx", "syy", "szz" etc.. The result type and component can be entered as upper or lower case. An element may have more than one surface or node and therefore the surface number and node must be entered in this function. Zero should be entered for subset if the load contains no subsets.

In instances where beam type elements have results at section positions along their length, the node field is used to identify the section positions. Care should be taken if section steps are used, as two results will be generated at these positions, one on either side of the step. For example, if you have a 10m long tube, with a section change at 2m from one end, and section results are requested at 10% intervals, you will have twelve result positions (two at the ends, nine at intermediate points, one at the step).

Example: `axlgetelement(Jacket, 35, 2, 10, "force/moment", "x",1,3)` returns the force in the x direction at the third node on the first surface of element 35. The results are retrieved for the tenth subset of loadcase 2 within the model "Jacket".

axlgetelementex

Purpose: Returns the requested element result, extended for additional result availability.

Syntax: *axlgetelementex (model, element, load, subset, result type, result component, surface, node, result position)*

Notes: This is an extended version of axlgetelement, all input is the same as that except for the additional result position. This is used for results where there are multiple values of a result, for example, STRESS in a time history based FATJACK analysis with rainflow counting.

Example: `axlgetelementex(C:\;ABCD;WXYZ, 72, 11, 1, "RANGE HISTOGRAM", "STRESS",1,2,10)` returns the 10th STRESS in the RANGE HISTOGRAM values at the second node on the first surface of element 72. The results are retrieved for the 11 loadcase, 1st subset within the project ABCD, structure WXYZ, located in C:\.

axlgetelementflt

Purpose: Returns the requested element result, filtered based on the given information.

Syntax: *axlgetelementflt (model, element, load, subset, result type, result component, surface, node, result position, filter type, return type)*

Notes: This is an extended version of axlgetelement; however it allows for the filtering of a range of values to be obtained. Model, result type, result position and result component all function as per axlgetelementex. The filtering is controlled by the filter type and requested value, for which the following should be entered:

Filter type	Filtering algorithm
1	Maximum
2	Minimum
3	Absolute Maximum (i.e. furthest from zero)
4	Absolute Minimum (i.e. closest to zero)

Return type	Returned Result
1	Value of Result
2	Element Number at which the value can be found
3	Load Case / Set for which the value can be found
4	Load Subset for which the value can be found
5	Surface at which the value can be found
6	Local node at which the value can be found

Element, load, subset, surface and node should have either a particular value assigned to limit the searching to a particular set of results, or -1 entered to loop over each result available, e.g. entering a loadcase of -1 will mean that all loadcases will be searched according to the filter type and requested value.

Example: `axlgetelementflt(C:\;ABCD;WXYZ, -1, 100, 0, "API LRFDALLOED1 UC", "UC.AXIAL",1,-1,1,2,1)` returns the maximum unity check value for all positions on all elements for loadcase 100. The results are retrieved for the project ABCD, structure WXYZ, located in C:\.

axlgetequation

Purpose: Returns the requested equation result.

Syntax: *axlgetequation (model, node, load, subset, result type, result component)*

Notes: The result type and result component indicate which information will be retrieved. For example the result type may be "displacement" and the result component "x", "y", "z" etc.. The result type and component can be entered as upper or lower case. Zero should be entered for subset if the load contains no subsets.

Example: `axlgetequation(Rig, 2, 3, 0, "displacement", "y")` returns the displacement in the y direction at the second node of loadcase 3 within the model "Rig".

axlgetglobal

Purpose: Returns the requested global result.

Syntax: *axlgetglobal (model, load, subset, result type, result component)*

Notes: Global results are specific to the load case and/or subset and not to a certain element or node. The result type and result component indicate which information will be retrieved. For example the result type may be "reaction sum" and the result component "x", "y", "z" etc.. The result type and component can be entered as upper or lower case. Zero should be entered for subset if the load contains no subsets.

Example: `axlgetglobal(Jacket, 10, 0, “prescribed reac sum”, “z”)` returns the prescribed reaction global sum in the z direction for loadcase 10 within the model “Jacket”.

axlgetnodal

Purpose: Returns the requested nodal result.

Syntax: *axlgetnodal (model, node, load, subset, result type, result component, freedom)*

Notes: Nodal results are generally concerned with the history of the displacement, velocity and acceleration of a node. The result type and result component indicate which information will be retrieved. For example the result type may be “history displacement” and the result component “x”, “y”, “z” or “time” etc.. The result type and component can be entered as upper or lower case. Zero should be entered for subset if the load contains no subsets. If “max displacement” etc. is entered as a result type this will retrieve the maximum displacement from all the load cases and therefore the load case should be entered as one and the subset zero. The freedom input indicates which freedom “x”, “y”, “z” etc to retrieve when using “max displacement” etc.

Example: `axlgetnodal(Rig, 12, 1, 0, “max acceleration”, “time”, 2)` returns the time at which the maximum acceleration occurred for freedom number 2 when considering all loadcases within the results set.

4.7 Miscellaneous Functions

There are also a few miscellaneous utility functions which do not access a particular ASAS model:

axlaxis

Purpose: Converts an axis or force name to a number.

Syntax: *axlaxis (axis)*

Notes: This function is mainly intended for use by other AXL functions. Given a string it attempts to convert it to an axis number where 1 is the X axis, 2 is Y and 3 is Z, 4 is rotation about the X axis, 5 is rotation about Y, 6 is rotation about Z. For example, “y”, “Y”, “FY” and “qy” are all returned as 2. If the axis is not recognised, zero is returned. It does not access ASAS files.

Example: `axlaxis(“Y”)` returns 2.

axlerror

Purpose: Returns a text string describing an error code returned by an AXL function.

Syntax: *axlerror (error)*

Notes: The description may be useful in determining the cause of a problem. See the section about error handling in this document.

Example: axlerror (302)
or:
axlerror (“axl:302”)
will return the description: “AXL: invalid element number”. This function should only be used when another AXL function in a cell displays a message of the form axl:nnn where nnn is an error number. For example, if cell C22 is showing axl:302, then typing:
=axlerror(C22)
in an empty cell will show a description of the error, as above.

axlversion

Purpose: Returns a text string identifying the version of the add-in macro sheet axl.xla and the associated dynamic link library axl.dll which are currently being used. This is mainly intended so that support personnel can identify possible problems due to the use of out-of-date software.

Syntax: *axlversion ( )*

Notes: This returns a string something like:

“axl.xla version 12.00.01.0, axl.dll version 12.00.01.0”

Note that the function does not require an argument.

Example: Not required.

4.8 Problem Solving

When using AXL functions, several different types of problem might arise. These fall into three main groups:

- errors in function names and arguments
- errors in values given to a function
- errors in accessing the ASAS™ files

Some problems may cause Excel to display one of its standard error indicators. #NAME? usually means that you have mis-typed the name of the function, or that the axl32.xla add-in has not been loaded. #N/A usually means that too few arguments have been given to a function. #VALUE! usually means that the wrong type of arguments have been given (such as text where a number is expected) or a wrong value for an argument with a restricted range of valid values (such as "Q" or 8 for an axis name, where only "X", "Y", "Z", 1, 2 or 3 would be acceptable).

Other problems cause AXL to display one of its own error indications. These all take the form of a short text #AXL:nnnn where nnnn is a 1 to 4 digit number. A list of these errors is given in the appendix below. If it is not obvious why an AXL function is returning an error code, look it up in the appendix or use the axlerror function as described above to determine its meaning. Some of the descriptions are self-explanatory.

Some general rules for solving problems are:

- Check that the model files are readable by using the axlnelement function. If this cannot retrieve the number of elements, then either the model data is incorrect or there is something wrong with the ASAS files.
- If an error message is described as "unable to open ... file" this usually means that something in the first part of the function call is wrong: perhaps the directory name is wrong, or the project or structure name. Perhaps the files have been deleted! If in doubt, try testing the files with a simple function. Suppose cell C3 contains the directory;project;structure text for the ASAS files. Try entering this formula into an empty cell:

  =axlnload(C3)

  which will return the number of load cases for the structure (which may be zero if it is a master component without loading data). If it returns an error code, then the problem is definitely with the ASAS files. If it returns the number of load cases, the problem is elsewhere.
- If the error is of one of the standard Excel types apply the following checks:

  #NAME? - check that the function name is spelled correctly and that the axl32.xla add-in has been loaded

  #N/A - check that the correct number and type of arguments have been given to the function - use the Formula|Paste Function... command in Excel to check what the arguments mean

  #VALUE! - if the cell contains a formula which does a calculation, the AXL function may be returning an error code and Excel is signalling that it cannot perform arithmetic on text values; if the cell simply calls a function check that arguments which can only take a few possible values, such as axis names, force names, etc. have been correctly specified

  #REF! - this error is generally nothing to do with AXL

- If the ASAS analysis has been re-run since the spreadsheet was last used, make a null edit of the cell where the ASAS files are defined (simply put the cursor on the cell, press F2, enter) and Excel will re-calculate as if you had changed the cell (even though you know that you have not). This may clear up problems due to Excel displaying out-of-date results.

- Many errors in tables come from confusion about absolute and relative cell addresses, so check that a formula that you have copied does reference appropriate rows and columns for element, node, etc. numbers.

  The individual functions in the DLL are designed to be fairly robust so they will not cause Excel or Windows® to crash in the event of problems. They only read from ASAS files, so cannot corrupt any data. However it is not advisable to attempt to read from an ASAS backing file at the same time as an ASAS program is trying to write to it.

4.9 Restrictions

Excel sometimes returns spurious error codes from functional calls. All results should be checked by the user.

There are no facilities for writing information back to the ASAS files.

The DLL functions can be accessed directly from Excel using the CALL function. However, this is not recommended because errors in the parameters to CALL can easily lead to Excel or Windows crashing. Don't do it!

4.10 Performance

The AXL functions are designed for convenience, not speed. The functions in the DLL do include some local caching to reduce the amount of file access when a single model is being used. However, as they comply with the general Windows recommendation for not keeping files open between calls (which also means that they do not require initialisation or close down and should not cause system problems if the ASAS files are updated while Excel® is running), they do perform a lot of input from the ASAS file system to retrieve even small amounts of data.

The amount of physical disk input can be minimised by use of disk caching. This should be enabled when Windows is in use anyway.

Excel does have an unpleasant habit of performing a full recalculation when not necessary (especially when a row or column is inserted or deleted, or after moves or copies). If a worksheet has a table containing thousands of numbers retrieved by AXL, this can take some time. One way of avoiding the recalculation is to create the table using AXL functions and then replace the formulae with the results so the cells contain only numbers thereafter - which require no recalculation. To do this, select the main body of the table, then the copy command (Edit | Copy on the menu). Immediately after this give the paste special command specifying values only (Edit | Paste Special | Values | OK on the menu). It's a good idea to save a single copy of the original formula somewhere in case the table has to be reconstructed.

4.11 Model Updates

Like all spreadsheet programs, Excel tries to avoid doing unnecessary recalculation. If nothing has changed in a spreadsheet, it does not automatically recalculate it. Even the Excel command Options | Calculate Calc Now (F9) will not force a spreadsheet to recalculate if nothing has changed in the spreadsheet itself.

This can be a problem with ASAS results where models may be rerun: as far as Excel is concerned a function asking for (say) the axial force in member 123 at node 567 for load case 89 does not need recalculating if none of these numbers have changed - it does not take account of the fact that the data in files outside Excel may have changed. This consideration applies even when a spreadsheet is closed and re-opened or Excel restarted.

To force all the results to be reloaded, make a null edit of the model name (as used in the first argument to each AXL function) by clicking on the cell, then pressing F2 to edit the cell, then enter to accept the edit. Excel will recalculate the spreadsheet and update all the values.

5. Error Codes

The error codes on the left can be translated into the text on the right with the axlerror function, described above. Some of these error messages relate to problems within the AXL interface that cannot be corrected by the user. These may arise due to errors within the AXL code, corrupt ASAS™ files, or lack of resources such as free memory or file handles.

#AXL:1	invalid input arguments, misc errors
#AXL:2	inconsistent input arguments
#AXL:3	invalid option
#AXL:9	no ASAS project, etc specified
#AXL:101	problem with information in /RUNINF/ common block
#AXL:102	RUNINF on IADMIN is wrong length or unreadable
#AXL:103	invalid index into RUNINF common block
#AXL:104	Units not defined in analysis
#AXL:105	Unknown beam section type found
#AXL:106	Node number is out of available range
#AXL:107	step entered is too high
#AXL:108	step entered is too low
#AXL:109	Number of elements in a group is inconsistent
#AXL:110	result type not available for this node
#AXL:111	result type not available for this node/element
#AXL:200	unable to allocate memory
#AXL:201	unable to allocate memory
#AXL:202	unable to allocate memory
#AXL:203	unable to allocate memory
#AXL:204	unable to allocate memory
#AXL:205	unable to allocate memory
#AXL:206	unable to allocate memory
#AXL:207	unable to allocate memory
#AXL:208	unable to allocate memory
#AXL:209	unable to allocate memory
#AXL:210	unable to allocate memory
#AXL:211	unable to allocate memory
#AXL:212	unable to allocate memory
#AXL:213	unable to allocate memory
#AXL:214	unable to allocate memory
#AXL:215	unable to allocate memory
#AXL:216	unable to allocate memory
#AXL:217	unable to allocate memory
#AXL:218	unable to allocate memory
#AXL:219	unable to allocate memory
#AXL:220	unable to allocate memory
#AXL:221	unable to allocate memory
#AXL:222	unable to allocate memory
#AXL:223	unable to allocate memory
#AXL:224	unable to allocate memory
#AXL:225	unable to allocate memory
#AXL:226	unable to allocate memory
#AXL:227	unable to allocate memory
#AXL:228	unable to allocate memory
#AXL:229	unable to allocate memory
#AXL:230	unable to allocate memory
#AXL:231	unable to allocate memory
#AXL:232	unable to allocate memory

#AXL:233	unable to allocate memory
#AXL:234	unable to allocate memory
#AXL:235	unable to allocate memory
#AXL:299	unable to allocate memory
#AXL:300	invalid identifier
#AXL:301	invalid node number
#AXL:302	invalid element number
#AXL:303	invalid load case number
#AXL:304	invalid material number
#AXL:305	invalid geometric property number
#AXL:306	invalid freedom number
#AXL:307	invalid freedom name
#AXL:308	invalid skew system number
#AXL:309	invalid axis
#AXL:310	invalid force or moment
#AXL:311	invalid index into skew system
#AXL:312	inconsistent units information on backing files
#AXL:313	no units defined in analysis
#AXL:314	analysis run time information not available
#AXL:315	invalid unit name
#AXL:321	node does not occur on element
#AXL:322	freedom does not occur at node
#AXL:331	element is not a 3D beam
#AXL:332	index of node on element is out of range
#AXL:333	invalid distance along pile element
#AXL:341	cannot determine coordinates for node not connected to an element
#AXL:351	requested element is not a beam
#AXL:352	step number does not exist on requested element
#AXL:353	section information not defined for requested element
#AXL:354	requested result type incorrect
#AXL:355	cannot determine coordinates for node not connected to an element
#AXL:356	invalid time step requested
#AXL:357	result not available
#AXL:359	requested result component invalid
#AXL:369	load case or subset not found
#AXL:370	inconsistent number of element stress results
#AXL:371	invalid subset number
#AXL:372	no result type results available
#AXL:373	there are no elements in the structure
#AXL:374	group number is invalid
#AXL:375	surface number is invalid
#AXL:376	invalid node number
#AXL:377	result type is not a nodal result
#AXL:378	result type is not a global result
#AXL:379	result type is not a equation result
#AXL:380	result type is not a element result
#AXL:401	inconsistent offsets calculated for IBSST data
#AXL:402	inconsistent information in LEX file
#AXL:413	problem with data in element information (LEX) file
#AXL:417	problem with data in partitioning information (IFPART) file
#AXL:429	problem with data in displacement (ISFIL) file
#AXL:432	problem with data in stress (IBSST) file
#AXL:456	problem with data in pile element information (LEXSP) file
#AXL:501	cannot open coordinate (IFCOOR) file
#AXL:505	cannot open geometry (IFGEOM) file
#AXL:506	cannot open skew information (IFSKEW) file

#AXL:513	cannot open LEX file
#AXL:515	cannot open LV file
#AXL:517	cannot open IFPART file
#AXL:529	cannot open ISFIL file
#AXL:532	cannot open IBSST file
#AXL:535	cannot open IADMIN file
#AXL:556	cannot open pile element information (LEXSP) file
#AXL:569	cannot open results (IFRESL) file
#AXL:570	this function can only be used with a physical 45 file
#AXL:601	cannot read data from coordinate (IFCOOR) file
#AXL:606	cannot read data from IFSKEW file
#AXL:613	cannot read data from LEX file
#AXL:615	cannot read data from LV file
#AXL:617	cannot read data from IFPART file
#AXL:629	cannot read data from ISFIL file
#AXL:632	cannot read data from IBSST file
#AXL:635	cannot read data from IADMIN file
#AXL:656	cannot read data from pile element information (LEXSP) file
#AXL:669	cannot read data from results (IFRESL) file
#AXL:701	cannot open results steering (IFRESU) file
#AXL:702	cannot read data from results steering (IFRESU) file
#AXL:703	cannot open displacement/velocity/acceleration (IFOUTD) file
#AXL:704	cannot read data from displacement/velocity/acceleration (IFOUTD) file
#AXL:801	illegal project name
#AXL:802	cannot open project file
#AXL:803	cannot locate structure
#AXL:804	cannot locate component
#AXL:805	requested file does not exist in ASAS index
#AXL:806	requested file exists in ASAS index but has no records so
#AXL:807	physical file does not exist or is not readable
#AXL:808	unable to allocate memory to read indexes, etc
#AXL:809	defective master index
#AXL:810	defective sub-index
#AXL:811	defective unit information table
#AXL:812	cannot open any more ASAS files without first closing some
#AXL:813	cannot allocate more space for internal control table
#AXL:814	directory name too long
#AXL:815	cannot allocate space for structure tree
#AXL:816	defective tree in project file
#AXL:817	tree is for a master component, which is not acceptable
#AXL:818	assembled component has not been recovered
#AXL:821	invalid ASBI file handle
#AXL:822	file is not open
#AXL:823	record number out of range
#AXL:824	record does not exist
#AXL:825	error in file control information (ASAS or local)
#AXL:826	unable to allocate memory for temporary indexes
#AXL:827	could not set file pointer (‘fseek’ failed)
#AXL:828	could not read physical record
#AXL:829	could not read block list
#AXL:830	could not read record list
#AXL:831	offset is too large, nothing left in record
#AXL:832	unable to allocate memory for block list
#AXL:833	unable to allocate memory for sub-index
#AXL:834	unable to allocate memory for unit information table

#AXL:835	too many models in project file for buffer supplied
#AXL:836	internal error in integer to character conversion
#AXL:837	internal error in integer to character conversion
#AXL:838	cannot open the extension files
#AXL:839	Number of characters to read must be a multiple of sizeof(long)
#AXL:851	parameter must be a real value or array
#AXL:852	parameter must be a column vector
#AXL:853	unable to allocate memory
#AXL:854	attempt to return zero length string
#AXL:855	process interrupted
#AXL:856	illegal access to encoded string buffer
#AXL:901	program fault - cannot get coordinates of node on component
#AXL:902	invalid RHS partition number
#AXL:903	internal buffer length provided not big enough
#AXL:999	unspecified error