

ARC MACRO LANGUAGE (AML)

Introduction

AML Basics

- Directives

- Functions

- Variables

- Other Important Features

 - Line Continuation

 - Blank Lines

 - Comments

 - Operators

Decision Making and Flow of Control

- &if

- &select

- &goto

- &run

- &call, &routine

- &system

Loops

- Counted Loop

- While Loop

- Until Loop

- Nesting Loops

File Input/Output

- Opening a File

- Reading From a File

- Writing to a File

- Closing a File

- Deleting a File

- Checking if a File Exists

- Formatting Output

The SHOW Command

Debugging

INTRODUCTION

AML is the language through which you communicate in the ARC environment. An AML program might be composed entirely of commands from ARC or any of the subsystems. However, there are several other elements of AML that allow for the creation of more powerful and flexible applications.

AML is an interpreted language. Each line of an AML passes through the AML processor before it is executed by the current arc program. ARC never sees any of the elements of AML; it only sees the results of their interpretation.

AML BASICS

DIRECTIVES

1. Begin with an "&"
2. Directives instruct AML to perform a specific operation, the result of which is never passed directly to ARC.

3. Examples:

&type this is a test	&amlpath /gis/gisinfo/aml
&describe landuse	&workspace /data/cascade/hja
&sv cover = landuse	&run gopherdoc.aml

FUNCTIONS

1. The name of a function appears within square brackets "[]".
2. A function performs text substitution. When AML encounters the square brackets, it evaluates the function contained in the brackets and returns the value of the function.
3. Functions can return a number, character string, or Boolean value.

4. Examples:

```
&sv cover = [response "Enter name of coverage"]
&sv value = [calc %var1% / %var2% * 100.]
```

VARIABLES

1. A variable is a means of storing dynamic information.

2. Variables can be character strings (up to 1024 characters in length), integers, real numbers, or Boolean expressions (.true. or .false.).

3. A variable is referenced by percent signs (%). When percent signs are encountered by AML, they are recognized as a variable reference and the value of the variable is substituted for the variable reference. Generally, the only time a variable is not surrounded by percent signs is when it is being set with the "&setvar" directive (abbreviated "sv"). For example:

```
&sv cover = LANDUSE
&type %cover%
```

4. A variable name can be up to 32 characters in length. Any character but AML special characters can be used (%,&,-).

5. Variable names are not case sensitive, but the value stored by a variable is:

```
&sv string = HELLO
&sv STRING hello
```

6. AML variables are not explicitly declared as a particular type. Therefore a variable may be used as a character one time, then set to an integer value another time. AML provides a function to determine the type of value that a variable is holding. See the [type] function.

7. Variables can be concatenated:

```
&sv m = /users/marks
&sv n = /landuse
&sv dir = %m%%n%
```

8. Indexed variables can be created. An indexed variable is analogous to an array. Indexed variables are created by using a common name for the variable, then adding a subscript:

```
&sv cover.1 = landuse
&sv cover.2 = geology
&sv number = 3
&sv cover.%number% = streams
```

To access the value stored in an indexed variable, using another variable to represent the index number, the [value] function is used. [value] returns the contents of the given variable. For example:

```
&type [value cover.%number%]
```

9. Variables can be local or global. Local variables are known only by the AML program

in which they are set. Other AML programs do not have access to their values. Global variables can be accessed and modified by all AML programs run in the current ARC session. Global variables are commonly used to pass information from one AML program to another. To set a global variable, precede the variable name with a dot (.):

&sv .cover = landuse	global variable
&sv cover = landuse	local variable

10. AML has some built in reserved or system variables. Reserved variables contain information about coverages, grids, tins, graphic coordinates and the current AML program. They have names that are built from three characters followed by a "\$". The names of these variables are reserved; that is, you should not define variables that have identical names. See the &describe directive.

DSC\$	coverage information (#polys, lines, tics, units) dsc\$cover, dsc\$xmin, dsc\$xmax
GRD\$	grid info variables (number of rows, cols, cellsize) grd\$rows, grd\$ncols, grd\$ymin
TIN\$	tin information tin\$zmax, tin\$xmin, tin\$xmax
LAT\$	lattice information lat\$xmin, lat\$xmax, lat\$ymin
PNT\$	graphics info (x,y coordinate of mouse, mouse key) pnt\$x, pnt\$y, pnt\$key
AML\$	AML program and menu info (severity, menukey) aml\$message, aml\$sev

INFO also has a set of system variables that can be used in AML. Some of these are: \$recno, \$nosel, \$time (see Appendix D in the INFO User's Guide for a complete list).

OTHER IMPORTANT FEATURES

1. Line Continuation

Sometimes it is necessary to continue a command over more than one line. AML's continuation character is a "~". If you end a line with a "~", AML will interpret it and the next line as one.

2. Blank Lines

Blank lines between commands are ignored in AML. But some ARC commands require blank lines or a carriage return to terminate input. When a tilde is the only character on

a line, it is interpreted as a carriage return. For example:

```
reselect name cn 'CA'  
~  
n  
n
```

3. Comments

A forward slash followed by an asterisk (/*) is AML's comment symbol. Comments can be placed anywhere in AML programs. They can exist on a line by themselves or follow a statement that is to be executed.

4. Tabs

Never use tabs to indent lines within your AML program!! The AML processor does not know how to interpret a tab. Use spaces only for indentation.

5. Operators:

Arithmetic operators		Logical operators	
+	addition	&, and	and
-	subtraction	!, or	or
*	multiplication	^,not	not
/	division	xor	xor
**	exponentiation		
ln	logarithm		

Relational operators

=,eq	equal to
<,lt	less than
>,gt	greater than
<=,le	less than or equal to
>=,ge	greater than or equal to
^=,<>,ne	not equal to
cn	contains
nc	does not contain
in	is in the specified set
lk	is like the specified set

DECISION MAKING AND FLOW OF CONTROL

AML statements are executed in the order in which they appear unless an explicit change in the flow of control is specified. Below are the statements which can change the flow of control.

1. &if statement

`&if <logical expression> &then <statement/action>`

The result of the logical expression is a Boolean value of true or false. The `&then` statement is executed when the condition is true and bypassed when the condition is false.

```
&if %value% = 10 &then
    &type value of variable is 10

&describe %cover%
&if [exists %cover% -poly] and %dsc$qedit% &then
    build %cover% poly
```

When the value returned by a function is a Boolean value, no relation operator is needed. `&if` statements made up of two or more expressions (with `and`, `or`) evaluate as:

`&if %val_1% or %val_2% &then`

<code>%val_1%</code>	<code>%val_2%</code>	
true	or true	TRUE, &then statement executed
true	or false	TRUE, &then statement executed
false	or false	FALSE, &then statement NOT executed

`&if %val_1% and %val_2% &then`

<code>%val_1%</code>	<code>%val_2%</code>	
true	and true	TRUE, &then statement executed
true	and false	FALSE, &then statement NOT executed
false	and false	FALSE, &then statement NOT executed

An `&if` allows only one statement to be executed following `&then`. To execute more than one statement, follow the `&if` with an `&do` block.

```
&if [exists %cover% -poly] &then &do
    &describe %cover%
    &if %desc$qedit% &then
        build %cover% poly
&end
```

An `&else` statement can be added to the `&if` statement to handle the condition when the logical expression is false.

```
&if <logical expression> &then
    <statement>
&else
    <statement>
```

or

```
&if <logical expression> &then
    <statement>
&else &if <logical expression> &then
    <statement>
&else
    <statement>
```

```
&sv slope = [show select %cov% line 1 item slope]
```

```
&if %slope% <= 10 &then
    &sv count10 = %count10% + 1
&else if %slope% > 10 and %slope% <= 20 &then
    &sv count20 = %count20% + 1
&else if %slope% > 20 and %slope% <= 30 &then
    &sv count30 = %count30% + 1
&else
    &sv count40 = %count40% + 1
```

2. &select

&select evaluates an expression only once and determines the flow of control based on the value of the expression.

```
&select <expression>
    &when <value>
        <statement>
    &when <value>
        <statement>
    &otherwise
        <statement>
&end
```

```
&select [show editfeature]
    &when ARC
        &menu arc.menu
    &when LABEL
        &menu label.menu
    &when NODE
        &menu node.menu
    &otherwise
        &type NO menu for this edit feature
&end
```

3. &goto

An &goto directive sends control to the statement following the specified &label directive:

```
&if %value% = 9 &then &goto stop
.
.
&label stop
&sv status = [close -all]
&return
```

4. &run

The &run directive runs another AML program. The flow of control is transferred to that program. When the program completes, control is returned to the calling program.

```
&if %choice% cn 'UTM' &then
  &run utm.aml
```

5. &call, &routine

Within a program, independent operations are created through routines. Routines alter the order of execution by transferring control to a different statement in the program. The AML directives that manage routines are &call and &routine. &call transfers control to the statement following the &routine directive and &routine delimits the beginning of a routine block. A routine commonly contains a series of statements that must be performed frequently in several different places in a program. A routine must end with an &return statement. Control is transferred to the line immediately following the &call directive when the &return statement is encountered. Routines are placed at the end of an AML program.

5. &system

The &system directive transfers the flow of control to the operating system to perform operating system commands. A "CTRL-D" returns you to ARC. For example:

```
&system
df
ls -l
CTRL-D
```

```
&system regression %filename% %num%
```

LOOPS

A loop executes a series of statements continually until some exiting criteria is met. There are several different kinds of loops in AML. Each performs the same type of operation, but differs in the way they are implemented. Although the syntax is slightly different for each loop variant, every loop begins with the &do directive and ends with the

&end directive. The syntax for three types of loops is described below, but AML has many more variants.

1. Counted Loop

The counted loop is used in situations where a given operation must be performed a known number of times.

```
&do i = 1 &to 20 &by 2
  &type i = %i %
&end
```

2. While Loop

The while loop executes while a logical expression is true and terminates when the expression is false. If the first test of the expression is false, the loop won't execute at all.

```
arcplot
res %cover% poly class = 2
&sv num = [extract 1 [show select %cover% poly]]
&sv i = 1

&do &while %i% le %num%
  &sv area = [show select %cover% poly %i% item area]
  &sv perimeter = [show select %cover% poly %i% item perimeter]
  &sv ratio.%i% = %area% / %perimeter%
  &sv i = %i% + 1
&end
```

3. Until Loop

An until loop executes until a logical expression evaluates to true. As long as the expression is false, the loop continues to execute. The until loop evaluates the logical expression at the end of the loop, not the beginning like the while loop. Therefore, the loop is always executed at least once.

```
&do &until %pnt$key% = 9
  &getpoint &map &cursor
  &sv status = [write %file% [quote %pnt$x% %pnt$y%]]
&end
```

4. Nesting Loops

Execution begins in the outer loop. When the inner loop is reached, the inner loop executes until it passes its test for completion.

```
&do i = 1 &to 2
  &do j = 1 &to 3
    &sv x%i%%j% = [response [quote Enter value for X(%i%,%j%)]]
```

- 0: record read successfully
- 100: file unit not open for reading
- 101: error during read
- 102: end of file
- 103: unable to re-open file
- 104: string truncated to 1024 characters

If the record read from a file contains any blank spaces, the value returned by the [read] function will be enclosed within single quotes. If these quotes are not desired, use the [unquote] function to remove them:

```
&sv record [unquote [read %file% status]]
```

The maximum length of a record that can be returned by the [read] function is 1024 characters; 1022 if the record contains blanks. If the record is longer than this, it is truncated and the status variable is set accordingly.

Here's an example of how to read all the records in a file:

```
&sv record = [unquote [read %file% status]]
&do &while %status% ne 102
  &sv x = [extract 1 %record%]
  &sv y = [extract 2 %record%]
  &type x: %x%, y: %y%
  &sv record = [unquote [read %file% status]]
&end
```

3. Writing to a file

Before writing to a file, the file must be opened with the -write or -append access. Once opened, the [write] function can be used to write a single record to the end of the file. The [write] function has no status variable. Instead, the status of the write request is indicated by the value returned by the function:

```
[write <file unit> <record>]
```

- 0: record written successfully
- 100: file not open for writing
- 101: error during write
- 102: unable to re-open file

The record to be written to the file can be a text string, a value stored in an AML variable, or a mixture of both. If the record contains blanks, it must be enclosed in single quotes with the [quote] function.

```
&sv record = Text string stored in a variable
&sv status = [write %file% [quote %record%]]
```

```
&sv status = [write %file% 'This is a record']
&sv status = [write %file% [quote This record contains a %record%]]
```

4. Closing a file

The [close] function can be used to close an opened AML file unit.

```
[close <file unit number | -all>]
```

```
&sv status = [close %file%]
&sv status = [close -all]
```

5. Deleting a file

The [delete] function can be used to delete a system file, a directory, or an INFO file. A file must be closed before it can be deleted.

```
[delete <file> <-file | -directory | -workspace | -info>]
```

6. Checking if a file exists

Use the [exist] function to determine if a file, coverage, grid, network, library, workspace, etc., exists:

```
[exist <object> -type]
```

```
&sv exist = [exists data.dat -file]
```

6. Formatting Output

Use the [quote] and [format] functions for writing text and variables to files. Use the [quote] function if you need to write out a string containing blanks or more than one variable to a record.

```
&sv cover = landuse
&sv type = polygon
&sv status = [write %file% %cover%]
&sv status = [write %file% [quote Coverage: %cover%]]
```

This won't work because it contains blanks:

```
&sv status = [write %file% Coverage: %cov%]
```

The [quote] function does not give you much control over how a record is formatted. Use the [format] function if you want fields to be the same width or justified. The [format] function substitutes arguments into the specified format string.

```
[format <format_string> {argument ....argument}]
```

The format string must be contained within single quotes. Up to ten format variables can

be placed within the format string. A field width for each argument can also be specified by including a comma and the field width after the format variable number. For example:

```
&sv status = [write %file% [format 'ID: %1,-5% Area: %2,-9% km' ~
%id%, %area%]]
```

A negative field width causes the output numbers to be right justified, a positive value, left justified. Numeric arguments will be formatted according to the number of decimal places specified with the &format directive.

SHOW COMMAND

The show command is very powerful. It returns all kinds of useful information about a coverage or environment settings established in the current ARC module. It is the only way to extract the value of a coverage attribute in a selected record. See the SHOW command in arcplot and arcedit, as well as the [show] function in AML for the syntax. The syntax differs for each returned parameter.

```
res landuse $recno = 5
&sv area = [show select landuse poly 1 item area]
&sv extent [show mapextent]
```

DEBUGGING

Listed below are some debugging tips, and common errors.

1. Use &type statements freely to print the value of variables. When a program dies, the line number of the statement it was executing is printed. Use the &type directive to type to the screen the value of all variables involved in the bad statement before it is executed.
2. Use &lv (listvariables) to list the value of all variables defined in the current arc session.
3. Check for tabs. The error message issued when a tab is encountered says nothing about tabs. Check several lines before and after the line where the program dies.
4. If a certain block of code gets executed when it shouldn't, check for a missing &end, or verify that the code was enclosed within an &do block.
5. If you get stuck inside a loop, verify that you incremented the counter.
6. Every command that is contained in your AML file can also be typed at the command line in ARC. If you can't figure out what's going on, often its helpful to type a command or (several commands) interactively at the ARC command line.

7. Turn on a watch file to see what's going on.

8. Verify that there are records selected before you attempt to execute a block of code.

```

/*****
/* File: patchinfo.aml
/*
/* Purpose: Written for Bill McComb. Writes the following information
/* to the file veg.table for each vegetation polygon in the input
/* coverage:
/*
/* veg class, area, perimeter, average elevation, average slope,
/* ownership, meters of road within polygon, meters of stream
/* within polygon
/*
/* Inputs:
/* polygon coverage of vegetation
/* polygon coverage of vegetation unioned with elevation, slope and
/* ownership
/* polygon coverage of vegetation unioned with roads
/* polygon coverage of vegetation unioned with streams
/*
/* Programmer: Barbara Marks
/*
/* Date: October, 1994
*****/

```

```

rformat 2 /* set 2 decimal places for outputting real numbers

```

```

&sv cov0 = [response 'Enter name of veg coverage']
&sv cov1 = [response 'Enter name of veg/elev/slope/ownership coverage']
&sv cov2 = [response 'Enter name of veg/road coverage']
&sv cov3 = [response 'Enter name of veg/stream coverage']

```

```

/*
/* Open report file and write header to it.
/*

```

```

&sv status = [close -all]

```

```

&sv file = [open veg.table status -write]
&if %status% ne 0 &then &do
  &type ERROR! Can not open file veg.table, error: %status%
  &stop
&end

```

```

&sv status = [write %file% [format 'CLASS AREA PERIMETER ELEV SLOPE
ROADLEN STREAMLEN']]
&sv status = [write %file% [format '
(m) (m) (ha) (km) (m) (deg)
(m) (m)']]

```

```

arcplot

```

```

/*
/* Get the maximum "#" value of the veg polygons
/*

```

```

clearsel
res %cov1% poly $recno > 1
statistics %cov1% poly
  max %cov0%#
end
&sv max = [show statistic 1 1]

```

```

&sv i = 2 /* start at 2; 1 is the universal polygon

```

```

&do &while %i% le %max%
  clearsel
  res %cov1% poly %cov0%# = %i%
  &if [extract 1 [show select %cov1% poly]] eq 0 &then
    &goto continue
/*
/* Get the veg class, average elevation and slope, and area of this
/* polygon.
/*
  statistics %cov1% poly
    mean elev
    mean slope
    mean newclass
    sum area
  end
/*
/* Can't control precision of values returned from the SHOW function.
/* Therefore "&format 2" has no effect. So force integers to be printed
/* as reals with 2 decimal places so that columns in output file are
/* aligned.
/*
&sv ave_elev = [show statistic 1 1]
&if [type %ave_elev%] eq -1 &then
  &sv ave_elev = %ave_elev%.00

&sv ave_slope = [show statistic 2 1]
&if [type %ave_slope%] eq -1 &then
  &sv ave_slope = %ave_slope%.00

&sv veg = [show statistic 3 1]
&sv patch_area = [calc [show statistic 4 1] / 10000] /* area in km
/*
/* Need perimeter from veg coverage because other coverages have multiple
/* polygons per veg polygon (slope, elev, ownership, roads, streams)
/*
res %cov0% poly %cov0%# = %i%
&sv perimeter = [calc [show select %cov0% poly 1 item perimeter] / 1000]
/*
/* Get names and area of all the landowners in this veg polygon
/*
statistics %cov1% poly owner
  sum area
end
&sv nowners = [show cases]

&sv j = 1
&do &while %j% le %nowners%
  &sv own.%j% = [show case %j%]
  &sv area.%j% = [calc [show statistic 1 %j%] / 10000]
  &sv j = %j% + 1
&end
/*

```

```

-/* Get the length of any roads that traverse this veg polygon.
/*
res %cov2% line %cov0%# = %i%
&if [extract 1 [show select %cov2% line]] > 0 &then &do
  statistics %cov2% line
    sum length
  end
  &sv road_len = [show statistic 1 1]
&end
&else
  &sv road_len = 0.00

/*
/* Get the length of streams that traverse this veg polygon.
/*
res %cov3% line %cov0%# = %i%
&if [extract 1 [show select %cov3% line]] > 0 &then &do
  statistics %cov3% line
    sum length
  end
  &sv stream_len = [show statistic 1 1]
&end
&else
  &sv stream_len = 0.00

/*
/* Write info to file. Right justify columns.
/*
  &sv status = [write %file% [format '%1,-2% %2,-7% %3,-6% %4,-7% %5,-6%
%6,-9%:%7,-6% %8,-7% %9,-9%' ~
%veg% %patch_area% %perimeter% %ave_elev% %ave_slope% [value own.1] ~
[value area.1] %road_len% %stream_len%]]

/*
/* If this veg poly has multiple owners, write owner and area on successive
/* lines.
/*
&if %nowners% > 1 &then &do
  &sv j = 2
  &do &while %j% le %nowners%
    &sv status = [write %file% [format '
%2,-6%' [value own.%j%] [value area.%j%]]]
    &sv j = %j% + 1
  &end
&end

  &label continue
  &sv i = %i% + 1
&end

&sv stat = [close -all]

&return

```

CLASS	AREA (ha)	PERIMETER (km)	ELEV (m)	SLOPE (deg)	OWNER/AREA (ha)	ROADLEN (m)	STREAMLEN (m)
4	2.56	0.84	558.33	31.67	USFS: 2.56	0.00	0.00
4	41.59	3.03	335.65	34.07	USFS: 40.05	6.10	2011.94
					PRIVATE: 1.53		
4	0.12	0.26	525.00	20.00	USFS: 0.12	0.00	0.00
2	1.54	0.68	570.83	25.00	SIMPS: 1.41	0.00	0.00
					USFS: 0.12		
0	14.83	1.55	417.50	37.50	SIMPS: 13.11	0.00	732.09
					USFS: 1.72		