AutoLISP®: Kicking It Up a Notch

Robert Green – CAD-Manager.com

CP419-1

So you've learned enough about AutoLISP to start writing your own functions and hack through other people's programs. Great. Now let's take it to the next level. In this class, we'll work through example program syntax to rapidly move though advanced concepts like set manipulation, extracting entity properties, entity updating, grouping, conditional operators (COND, WHILE and IF), querying the drawing header (TBLSEARCH), and some of the more common Visual LISP® functions (VL and VLR) that you can use to write powerful programs for engineering, architectural, or civil environments. This course will make use of example problem statements so you can understand what the problem is, and then apply the new syntax you learn to solve the problem as we go. Make sure to bring the copy of the handout, a pencil, and put on your thinking cap as we take your AutoLISP skill set to the next level.

About the Speaker:

Add edited Speak Robert is head of the Robert Green Consulting Group, and a 15 year veteran speaker at Autodesk University. You have likely read his work in Cadalyst magazine, where he authors the "CAD Manager" column, or in his bi-monthly CAD Manager's Newsletter. He holds a degree in Mechanical Engineering from the Georgia Institute of Technology and gained his CAD skills from 24 years of AutoCAD®, MicroStation®, and MCAD® software usage. Since starting his own company in 1991, Robert has performed consulting and teaching duties for private clients, and throughout the U.S. and Canada.

r.green@CAD-Manager.com
Pre-Requisite Topics

This course is a continuation of “The AutoLISP Cheat Sheet for CAD Managers” which is course number CM228-1 at AutoCAD University 2010. If you have any questions about the pre-requisite topics below you can download the first course handout and read through it to get up to date.

You can also find my course guide at my web site:  www.cad-manager.com/au2010

For today’s course I assume the following knowledge:

• Using the ACADDOC.LSP file to load in programs
• Basic command line access concepts
• Basic command function declarations using DEFUN C:XXXX format
• Basic system variable functions – SETVAR and GETVAR
• Basic variable types – REAL, INTEGER and STRING
• Basic comparators - =, <, >, <=, >=
• Capturing information to variables – SETQ, GETDIST, GETINT and GETSTRING

Today’s Topics and Approach

This session will introduce the following AutoLISP topics:

• Conditional operators – IF, WHILE, COND, AND, OR
• Coordinate operators – CAR, CADR, CADDR and LIST
• Error handlers - *ERROR*
• Working with sets – Implied or SSGET

Along the way I’ll pass along hints for debugging your code and getting your program to work with minimal stress. To introduce the concepts I’ll use an approach that I call “pseudo code” where I state a problem in plain English, then write out the steps of the program (again in plain English) so everyone can understand what the program should do. Once the problem is stated we’ll look at the AutoLISP code that actually accomplishes the required task.

What is important is that you understand the CONCEPT of what the program is supposed to do, THEN you can worry about the specific code.

Resources

Copies of handouts and additional example files for this presentation can be found at my web site using this URL:  www.cad-manager.com/au2010  Feel free visit and check out the challenge exercises, sample code and expanded coverage offered.
Conditional operators
It all starts with injecting logic into your programs so they can actually “be smart” via branching and looping structures. In programming parlance we call these types of functions conditional operators.

In order for your programs to execute logical operations you need to use conditional operators like IF, WHILE and COND depending on the particular circumstances.

The IF conditional
Let’s kick things off with the IF command.

Basic format: (IF (TEST EXPRESSION)
  (TRUE CASE)
  (FALSE CASE)
)

As you can see the IF command evaluates a test to see if the test is TRUE or FALSE. The result of the IF command then goes two a true case and a false case. Therefore the IF command can be though of as a logical “fork in the road” since there are two distinct outcomes.

Pseudo code: A user is prompted for an integer variable called SPOKES
  If the variable has a value less than 20 then print “LESS THAN 20”
  If the variable has a value greater than/equal to 20 then print “MORE THAN 20”

Real code: (if (< spokes 20)
  (alert “LESS THAN 20”) ; true case
  (alert “MORE THAN 20”) ; false case
)

The COND conditional
COND is like a “super IF” command in that it allows you to evaluate a test case and take action but you may have any number of outcomes rather than the IF statements two outcomes. Let’s see the structure of the command:

Basic format: (COND
  ( (TEST EXPRESSION 1) (OUTCOME) )
( (TEST EXPRESSION 2) (OUTCOME) )

( (TEST EXPRESSION 3) (OUTCOME) )

)

Note that there are more parentheses to keep track of and that the structure is a little different than IF, but the basic premise is very much like IF.

**Pseudo code:** A user is prompted for an integer variable called **SPOKES**
If the variable has a value less than 10 then print “SMALL WHEEL” to the screen
If the variable is between 10 and 15 print “MEDIUM WHEEL”
If the variable is greater than 15 print “LARGE WHEEL”

**Real code:**

```
(setq spokes (getint "Enter number of spokes: "))
(cond
  ( (< spokes 10) (alert "SMALL WHEEL") )
  ( (and (>= spokes 10) (<= spokes 15)) (alert "MEDIUM WHEEL") )
  ( (> spokes 15) (alert "LARGE WHEEL") )
)
```

**Note:** See how the AND operator was used to evaluate a more complex test case? In the MEDIUM case the spokes had to be in the range between 10 and 15 so two evaluations were required.

**The WHILE conditional**
The main use of the WHILE conditional is to execute a looping structure to qualify user inputs.
Let’s see the structure of the command:

**Basic format:** *(OBTAIN VARIABLE)*

```
(WHILE (TEST EXPRESSION ON VARIABLE))
  (Statement 1)
  .
  (Statement N)
```


Let’s say we want to prompt the user for a number of spokes but the number of spokes needs to be no less than 20 for our purposes. How can we handle a case where the user gives us an incorrect number? Let’s use the WHILE conditional like this:

```lisp
(setq spokes 0)
(while (< spokes 20)
  (setq spokes (getint “Enter number of spokes: ”))
)
```

In this case we initialize the variable to less than 20 so we will enter the WHILE loop. Now if a user enters a value for spokes that is LESS than 20 they'll be stuck in the loop. If they enter a value of 20 or higher the test expression is true and the loop is exited. Simple enough.

**Conditional Example**

Let’s now write a program that uses conditional structures to achieve some complex logic. Here’s the pseudo code for the example:

- Prompt the user for a number of spokes
- Be sure they give you at least 8 spokes but not more than 24 spokes
- Write SMALL MEDIUM or LARGE to the screen based on the rules we used in the COND example

**Real Code**

First I’ll write the code that assures user input and enforces the maximum and minimum spokes cases. This looping structure will use a WHILE function along with a complex test case like this:

```lisp
(setq spokes 0)
(while (or (< spokes 8) (> spokes 24))
  (setq spokes (getint “Enter number of spokes [from 8 to 24]: “))
)
```

See how this will stick the user in the input loop if the spokes are too few or too many? Now the portion of the code that makes the small, medium or large designation:

```lisp
(cond
  ( (< spokes 10) (alert “SMALL WHEEL”) )
  ( (and (>= spokes 10) (<= spokes 15)) (alert “MEDIUM WHEEL”) )
  ( (> spokes 15) (alert “LARGE WHEEL”) )
)
```

**Points and List Manipulation**
At its core, AutoCAD is a program that manipulates graphical entities by using the association list you’ve now seen to control aspects like layer, color, etc. But from a geometric standpoint AutoCAD is all about the points. As an example any line has two controlling points while a circle has a centering point and a point on the circle that defines the circle’s radius.

Our mission now is to understand how to work with the points to construct geometry. First a few basics:

Obtain a point PT1 from a user like this:  \( \text{(setq pt1 (getpoint \"Select a point: \")} \)
Now get PT2 in the same way like this:  \( \text{(setq pt2 (getpoint \"Select a point: \")} \)

You now have two 3D points stored in the variables PT1 and PT2 right?

**CAR, CADR, CADDR**
To decompose points into their X, Y and Z coordinates you need to know how to use the CAR, CADR and CADDR functions. Let’s demonstrate:

The X value of PT1 would be:  \( \text{(car pt1)} \)
The Y value of PT1 would be:  \( \text{(cadr pt1)} \)
The Z value of PT1 would be:  \( \text{(caddr pt1)} \)

Why? Because that’s how somebody defined it! Just memorize these functions and we’ll move along.

**LIST**
Now let’s say that you’d like to draw a circle with an origin point that is at the X coordinate of PT1 and the Y and Z coordinate of PT2. Further, the radius of the circle should have a value of 11.5. Here’s how that code would look:

\( \text{(command \".circle\" (list (car pt1) (cadr pt2) (caddr pt2)) \"11.5\")} \)

The unexpected part here is the LIST statement, but the concept is easy considering the following:

- To build a point you must have an X, Y and Z coordinate.
- The CAR, CADR and CADDR functions allow you to obtain the X, Y and Z.
- The LIST function simply concatenates the coordinates together into one point
- The CIRCLE command takes the point as input

Once you start thinking LIST every time you access a point and follow the basic rules above it gets easy.
Expanded Example
Let’s now beef up our wheel creating program by adding a few more requirements. Here’s the pseudo code for the example:

- Prompt the user for a point for the wheel center
- For small wheels use spoke length of 8, medium gets 10 and large gets 12
- Draw a first spoke at the center point
- Array the first spoke with the appropriate number of spokes

Real Code
First I’ll write the code that assures user input and enforces the maximum and minimum spokes cases. This looping structure will use a WHILE function along with a complex test case like this:

```lisp
(setq spokes 0)
(while (or (< spokes 8) (> spokes 24))
  (setq spokes (getint "Enter number of spokes [from 8 to 24]: "))
)

(setq wheel_center (getpoint "Select center point of wheel: "))

(cond
  ; (<= spokes 10) (command ".line" wheel_center "@8<0" "")
  ; (and (>= spokes 10) (<= spokes 15)) (command ".line" wheel_center "@10<0" "")
  (>) (command ".line" wheel_center "@12<0" "")
)

(command "-array" "last" "p" wheel_center spokes "360" "y")
```

How did I know how to write the ARRAY logic? Here’s a few hints:

- Use the –ARRAY command so you interact with the command prompt not a dialog box
- Use the LAST object drawn to build the selection set
- Use the “” to close the selections
- Use the WHEEL_CENTER and SPOKES variables to pass user input to the command line
- Go through the command manually to get the sequencing

A Little more
Let's now change the program so it'll always draw the spokes on the layer WHEELS which has a color assignment of YELLOW like this:

```lisp
(setq spokes 0)
(while (or (< spokes 8) (> spokes 24))
  (setq spokes (getint "Enter number of spokes [from 8 to 24]: "))
)

(setq wheel_center (getpoint "Select center point of wheel: "))

(command "-layer" "make" "wheels" "color" "yellow" """)

(cond
  ((< spokes 10) (command ".line" wheel_center "@8<0" ""))
  ((and (>= spokes 10) (<= spokes 15)) (command ".line" wheel_center "@10<0" ""))
  ( (> spokes 15) (command ".line" wheel_center "@12<0" ""))
)

(command "-array" "last" "p" wheel_center spokes "360" "y")
```

Again, check out the –LAYER command sequencing manually and you’ll see where I got the order of operands.
Error Handling
While I’m sure you’ll never make a mistake in your programs (insert chuckle here) what about
the person who uses your programs? They might hit the ESC key at the wrong time, they might
provide input to the program that makes it crash, etc.

To make sure your program environment gets put back to the correct state after an error
AutoLISP gives you a way to define your own error functions. Here’s a quick primer on what
you need to know.

AutoCAD’s default error handling function is called *ERROR* and is defined like this:

(defun *error* (msg)
    (princ "error: ")
    (princ msg)
    (princ)
)

You can create your own error handler like this:

(defun *custom_error* (msg)
    (princ "Custom error: ")
    (princ msg)
    (princ)
)

But it isn’t quite that simple because you need to make sure to put the original error handler back in place.
Therefore your original program will store the old error handler and put the new error handler in place like
this:

(setq orig_error *error*) ; stores old error handler
(setq *error* *custom_error*) ; makes your error handler current

Now make sure that your error handler has a line in it that resets the error handler like this:

(defun *custom_error* (msg)
    (princ "Custom error: ")
    (princ msg)
    (setq *error* *orig_error*) ; This line reflects your program above
    (princ)
)

Let’s further say that your original program was changing layer names at some point in its execution.
Therefore you insert a line in your program to store the original layer like this:

(setq orig_error *error*)
(setq *error* *custom_error*)
(setq old_layer (getvar "clayer")) ; stores the original layer

Now make sure that your error handler reflects the layer variable like this:

(defun *custom_error* (msg)
Let me draw a few conclusions about error handlers:

- Writing your own error handlers requires you to keep track of all variables in your programs so you can be sure to set them back in your error handler.
- Error handlers require more thought, and more planning, but make your programming environment more stable and professional.

Finally, you can always test your error handlers by forcing your program to terminate abnormally by hitting the ESC key several times during execution.

**Accessing Entities**

Accessing entities is the only way I know of to see what’s “under the hood” of any given entity. You’ll also need this trick to manipulate entities and sets so let’s start simply by showing you how to obtain an entity association list and interpret the results.

If you define the following C:GE (short for get entity) function you can select any entity in the drawing and the function will print out the information about the entity on the command line.

```
(defun c:ge ()
  (setq ent_temp (entget (car (entsel "Select entity: "))))
  (print ent_temp)
  (princ)
)
```

Since there are several things going on in the first line of this program let me break the steps down for you. Remember you always work from the innermost parenthesis and solve your way back out:

**Action:** (entsel "Select entity: ") ; Requests a selection of a single entity

**Returns:** (<Entity name: 7efa0390> (39.5417 23.1991 0.0)) ; This is the entity and point picked

**Note:** What makes ENTSEL different is it remembers WHERE you picked and WHAT you picked which can sometimes come in handy.
**AutoLISP®: Kicking It Up a Notch**

**Action:** (car (entsel "Select entity: "))

**Returns:** <Entity name: 7efa0390>

**Action:** (entget (car (entsel "Select entity: ")))

**Returns:** 

```
((-1 . <Entity name: 7efa0390>) (0 . "LINE") (330 . <Entity name: 7ef9ecf8>)
 (5 . "14A") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine")
 (10 24.5076 13.113 0.0) (11 44.8963 27.0071 0.0) (210 0.0 1.0))
```

**Action:** (setq ent_temp (entget (car (entsel "Select entity: "))))

**Returns:** A variable called ENT_TEMP which can be used later in the routine when the variable is printed.

**Modifying Entities**

Now you can change the layer for this line from "0" to "TEMP" like this:

```
(setq ent_temp (subst (cons 8 "TEMP") (assoc 8 ent_temp) ent_temp))
(entmod ent_temp)
```

All we’re doing here is accessing the entity variable ENT_TEMP and replacing its layer assignment with the layer TEMP. By constructing a new list for the layer parameter (cons 8 "TEMP") and then substituting it back into the original entity you’re changing the layer but you don’t have to use the CHANGE command.
Building Entity Sets - SSGET

Let's say that you wanted to build a program that would find all the circles in your drawing, no matter what layer they were on and change them to a layer called CIRCLES with a bylayer color of BLUE.

First, let's draw a circle in AutoCAD and use the GE function from the previous session to get an idea of how the circle is constructed. Here's the resulting list from a circle entity:

```lisp
((-1 . <Entity name: 7efa0390>) (0 . "CIRCLE") (330 . <Entity name: 7ef9ecf8>) (5 . "14A") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 39.9536 25.0516 0.0) (40 . 8.1027) (210 0.0 0.0 1.0))
```

Note the pair (0 . "CIRCLE") which indicates entity type and (8 . "0") which indicates the layer. We know the circle has a BYLAYER color attribute because the entity list does not include a (62 . X) pair (where X would be the AutoCAD color number). I know this may all seem confusing at first, but if you use the GE function I've give you to explore various entities you'll start to see the patterns.

Note: You can also find out about entity list parameters in the Developer's Help section of the AutoCAD help system.

Now let's make the leap to building selection sets with SSGET by using an example. First we'll look at building a set which we'll store as a variable name called CIRCLE_SET that finds all circles in the drawing no matter what layer they are on:

```lisp
(setq circle_set (ssget "x" '((0 . "CIRCLE")) )
```

This is an example of selection set filtering which is denoted by the "x" in the format of the command. What comes next is a quoted list of all the associative pairs of data you'd like to look for in building the set. The fewer items in the quoted list, the more broad the search. Here's an example of finding only circles on layer GEOM:

```lisp
(setq circle_set (ssget "x" '((0 . "CIRCLE") (8 . "GEOM")) )
```

What would this example find:

```lisp
(setq circle_set (ssget "x" '((0 . "CIRCLE") (62 . 2)) )
```

This would find circles that are yellow. Remember the (62 . X) parameter I gave you earlier?
Where to Use This?

Anywhere you would normally require a selection set of objects like this:

```lisp
(command "change" circle_set "" ... change properties here ...)
```

or

```lisp
(command "erase" circle_set "")
```

Don’t make it too complicated, the SSGET function simply allows you to build sets via filtering just like you would in AutoCAD … just easier that’s all!

Loading Code

After you’ve written a program and debugged it you might like to save your code in its own LSP file for your convenience. Therefore the question becomes how can you load programs into the AutoCAD environment? Well, there are a few ways that I’ll cover briefly here.

Option 1 – Using Appload

You can always load an LSP file using AutoCAD’s APPLOAD command to navigate to the file and load it in. The only problem with apploading a program is that the results of the load will be lost the next time you exit AutoCAD unless you know the trick. To use APPLOAD simply type it in at the command line.

If you want to load your program EVERY TIME AutoCAD starts simply use the Startup Suite CONTENTS button to add the program name permanently as shown at right.

Using APPLOAD has its advantages in that it is graphically oriented and easy to navigate, but to really gain control over your programs and AutoCAD’s startup using the ACADDOC.LSP gives the best results so we’ll look at it next.

Option 2 – Load from ACADDOC.LSP
Since the ACADDOC.LSP is automatically loaded at AutoCAD’s startup you can simply insert a line into your ACADDOC.LSP file that loads the program remotely like this:

```
(load "c:\mydirectory\myprogram.lsp")
```

Where the MYDIRECTORY and MYPROGRAM parameters are simply replaced with the appropriate values. Note the `\` characters required for pathing!

**Centralizing Your Code**

In this case we wish to load an external AutoLISP file with some utilities in it (called UTILS.LSP) which resides on a network drive. The contents of the ADADDOC.LSP would look like this:

```
(if (findfile "x:\autolisp\utils1.lsp") ; findfile verifies that the file exists
  (load "x:\autolisp\utils1.lsp")
)
```

Now you’ve got a way to load files from a remote network location so the programs can be maintained in one place! This is the best way to load programs (especially if you’re a CAD manager) because you never have to visit the user’s machine to modify your utilities.

**Compile Your Code**

Let’s face it, if you put unprotected LSP code on your network somebody is going to hack into it and change it. Wouldn’t it be wiser to simply secure your code from tampering before this happens? You bet it would and the good news is the Visual LISP compiler allows you to do so easily.

The steps are simple:

- Know where the program you’re compiling is located (example: `C:\TEST\MYPROG.LSP`)
- Start the Visual LISP Integrated Development Environment by typing VLIDE at AutoCAD’s command prompt
- Compile your program using this syntax: `(vlisp-compile 'st "c:\test\myprog.lsp")`
The results of this compile will be that you now have C:\TEST\MYPROG.FAS as a compiled (or Fast Load) formatted file. This file can be loaded in via the APPLOAD function or using a load command like this:

(load "c:\test\myprog.fas")

Safe and secure! Now that’s better.

**VLIDE Environment (and Help)**

You can also write your code in the VLIDE window (see below) and get the benefit of a drop down function selector and parenthesis matching. Some programmers really like the VLIDE environment while others (myself included) prefer to use their own editing program. Either way is fine. It is simply a matter of preference.

If you do choose to dig into the VLIDE interface spend some time with the HELP utility provided with it.
While we're on the subject of help, use the Developer's Help section of the AutoCAD help system and you'll find tons of information on all the VLIDE functions and AutoLISP functions as well.

**Updated Materials and PowerPoints**

You can download the updated course guide (with any additional notes or corrections) for this presentation at my web site [www.CAD-Manager.com/au2010](http://www.CAD-Manager.com/au2010).

I will send you a PDF copy of the session PowerPoint presentation if you request it. Just send an email to me at [rgreen@cad-manager.com](mailto:rgreen@cad-manager.com) and be sure to put the course title in the subject line so I'll know which class you want.