AutoLISP® / Visual LISP® - Part 1

Robert Green
CAD-Manager.com
My background …

- ME degree from Georgia Tech
- AutoCAD user since 1985
- AutoLISP programmer since 1990
- Cadalyst columnist since 1998
- Self confessed computer geek
- Coming to you from Atlanta GA so …
Let’s learn some LISP y’all!
Course guide (password = LISP1)
Asking a question ...

Chat

Thank you for joining today's training! We will begin momentarily.
Some fundamentals

- LISP – LISt Processor
- Everything is in lists
Some history ...

- LISP is an old language
- It was open source (cheap)
- It made sense for AutoCAD because AutoCAD entities are lists
- It runs without compiling (simple)
- It made sense at the time (cheap) ...
Some advice …

- AutoLISP is still very valid!
- A HUGE base of LISP code exists
- LISP shareware is abundant
- LISP does many things very easily as compared to VB/VBA
- LISP is weak on dialog controls
- LISP is weak on graphical interfaces
- View it as an 80/20 language …
Key files and why they matter …

- LSP
- Load in order
- What to mess with
- What not to!
All those wacky file names …

- ACAD20XX.LSP (XX is version)
- ACAD.LSP
- ACAD20XXDOC.LSP (XX is version)
- ACADDOCC.LSP
- MENUNAME.MNL
- Help …
What do they do ...

- They load on startup of AutoCAD
- They load in a certain order (listed on previous slide)
- Some have code in them and some don’t (ACADDOCD.LSP and ACAD.LSP don’t as an example)
- They reside in the SUPPORT folder ...
So what should I do …

- Use ACADDOC.LSP to get started
- You create your own ACADDOC.LSP so you can’t really mess it up
- It loads with every new drawing
- Put it in the SUPPORT folder and start hacking away
- If you mess up too bad, just delete!
Syntax Basics

- Lists and Arguments
- Rules of AutoLISP
- Variables
- Functions
- Accessing the command line
- Special characters
- User input
Lists and Arguments

- \((+ 20 30)\)
  - Here the + is a FUNCTION and the two numbers are ARGUMENTS
- \((\text{command} \ "\text{line}\" \ "0,0\" \ "1,1\" \ "\")\)
  - Here COMMAND is the function, all others are ARGUMENTS
Some Rules

“For every ( there is an equal and opposite )” – Isaac Newton
Like this: (setq A 3.0)
Not like this: (setq A 3.0)) or (setq A 3.0)

Same goes for quote marks!
Like this: (setq name “Robert Green”)
Not like this: (setq name “Robert Green"

When formatting numbers always avoid “invalid dotted pairs”
Like this: (setq A 0.5)
Not like this: (setq A .5)
## The New Math

<table>
<thead>
<tr>
<th>Standard</th>
<th>AutoLISP</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 + 4</td>
<td>(+ 8 4)</td>
<td>12</td>
</tr>
<tr>
<td>8 + 2.3</td>
<td>(+ 8 2.3)</td>
<td>10.3</td>
</tr>
<tr>
<td>5.0 ÷ 4.0</td>
<td>(/ 5.0 4.0)</td>
<td>1.25</td>
</tr>
<tr>
<td>5 ÷ 4.0</td>
<td>(/ 5 4.0)</td>
<td>1.25</td>
</tr>
<tr>
<td>5.0 ÷ 4</td>
<td>(/ 5.0 4)</td>
<td>1.25</td>
</tr>
<tr>
<td>5 ÷ 4</td>
<td>(/ 5 4)</td>
<td>1</td>
</tr>
</tbody>
</table>
## The New Math (con’t)

<table>
<thead>
<tr>
<th>Standard</th>
<th>AutoLISP</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8 + 4) * 3</td>
<td>(* (+ 8 4) 3)</td>
<td>36</td>
</tr>
<tr>
<td>(3.0 * 5.0) ÷ 4.0</td>
<td>(/ (* 3.0 5.0) 4.0)</td>
<td>3.75</td>
</tr>
<tr>
<td>((4 + 4) * 3) ÷ 2</td>
<td>(/ (* (+ 4 4) 3) 2)</td>
<td>12</td>
</tr>
</tbody>
</table>
The Command Line

- How would you work normally?
- Draw a line between two user points
  - Type in `LINE` to start the line command
  - Click `POINT` for first point location
  - Click `POINT` for second point location
  - Type in `ENTER` to halt the command

- In AutoLISP: (command “line” pause pause “”)
- PAUSE instruction waits for the user
- The “” is equal to hitting the ENTER key.
- Why? Because that’s they way it is …
What’s Going On Here?

- (command “viewres” “y” “5000”)
- (command “-color” “BYLAYER”)
- (command “-linetype” “set” “BYLAYER” “”)
- (command “menu” “menuname.mnc”)
- (command “viewres” “y” pause)

- That’s not so bad … intuitive actually …
Variables and Data Types ...

- Real
- Integer
- String
- User input
Variables and data types

- SETQ (SET eQual)
- (setq VALUE 12.7) – real number
- (setq VALUE 1) – integer number
- (setq VALUE “Happy”) – string variable
- (setq VALUE 1.0) – ?
- (setq VIEWRES_VALUE 5000) – ?
Variables and Commands

- (setq VIEWRES_VALUE 5000)
  - (command “viewres” “y” viewres_value)
  - What would happen in this case?
- (setq VIEWRES_VALUE “Happy Bday!”)
  - (command “viewres” “y” viewres_value)
  - What would happen in this case?
- Variable TYPES and how they interact with commands are key ...
User Input

- (getdist "\nInput a distance: ")
- (getreal "\nInput a real number: ")
- (getint "\nInput number of items: ")
- (getstring "\nInput text: ")
- What’s the \n – a line feed
- Why?
- Just because …
User Input (reference handout)

(setq user_dist (getdist "Input a distance: ")
(setq user_real (getreal "Input a real number: ")
(setq user_int (getint "Input an integer number: ")
(setq user_string (getstring "Input text: ")
(setq wheel_dia (getdist "Enter wheel diameter: ")
(setq spokes (getint "Enter number of spokes: ")

Get the idea …
User functions

- Speed for the user
- Lower support for you
- A win-win scenario
- Let’s put everything we’ve learned into action to build some functions.
User Function Examples

(defun C:ZA ()
  (command "zoom" "a")
  (princ)
)

AutoLISP/Visual LISP Part 1
Robert Green
User Function Examples

(defun C:VR ()
  (command "viewres" "y" "5000")
)

(defun C:BL ()
  (command "-color" "BYLAYER")
  (command "-linetype" "set" "BYLAYER" "")
  (princ)
)
Fillet Zero Function

Fillet Zero
(defun c:fz ()
  (setvar "filletrad" 0.0)
  (command "\.fillet" pause pause pause)
  (princ)
)
Improved Fillet Zero

(defun c:fz ()
  (setq old_filletrad (getvar "filletrad"))
  (setqvar "filletrad" 0.0)
  (command ".fillet" pause pause pause)
  (setqvar "filletrad" old_filletrad)
  (princ)
)

AutoLISP/Visual LISP Part 1
Robert Green
Auto Purge Function

Auto Purge
(defun c:atp ()
  (command "-purge" "a" "*" "n" "qsavex")
  (princ))
More Bits and Bytes ...

- Undefine
- Dot form
- Redefine
- Alerts
- CMDECHO
Undefining …

- (command “.undefine” “LINE”)
- (command “.undefine” “TORUS”)
- Don’t want them messing with a command? Just undefine it …

Now you can SUBTRACT from the AutoCAD Command set in your ACADDOC.LSP file.
The DOT form …

- Invoke commands like this: .LINE
- Note the dot “.” character?
- This allows you to invoke a command whether it has been undefined or not!
- This is our little secret right ...
Redefining ...

- (command "r redefine" "LINE")
- (command "r redefine" "TORUS")
- Want to be sure that a command is active?
- Just redefine it ...

Now you can UNSUBTRACT from the AutoCAD Command set with ease.

AutoLISP/Visual LISP Part 1
Robert Green
Undefining revisited …

- What if your users find out about REDEFINE and start REDEFINING your UNDEFINES?
- Just undefine the redefine like this:
- `(command "\.undefine" "REDEFINE")`
- That’ll show ‘em …
Redefining …

- You can undefine a command and redefine it like this:

```lisp
(command "undefine" "TORUS")
(defun C:TORUS ()
  (alert "Don’t use that command!")
  (princ)
)
```

Now you do whatever you want!
What Does This Do?

(command "."undefined" "QSAVE")
(defun c:qsave ()
  (command "-purge" "b" "+" "n")
  (command ".qsave")
  (princ)
)
Alerting the user …

- You can send a message to the user like this:

(alert "Message goes here")
Command echo (CMDECHO)

Run in STEALTH mode like this:

(defun C:BL ()
  (setvar "cmdecho" 0)
  (command "-color" "BYLAYER")
  (command "-linetype" "set" "BYLAYER" "")
  (setvar "cmdecho" 1)
  (princ)
)

AutoLISP/Visual LISP Part 1
Robert Green
Conditional operators

These functions allow your program to “think” and “react” to user input.

Loops and branches form the basis for the conditional operators we’ll look at.
IF – it all starts here …

(IF (TEST EXPRESSION)
   (TRUE CASE)
   (FALSE CASE)
)
IF – 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” to the screen
- If the variable has a value greater than or equal to 20 then print “MORE THAN 20” to the screen ...
IF – Real Code

(setq spokes (getint "\nEnter number of spokes: "))
(if (< spokes 20)
  (alert "LESS THAN 20") ; true case
  (alert "MORE THAN 20") ; false case
)

Note how a decision is made using a branching logic structure …
COND – IF on steroids

(COND
  ( (TEST EXPRESSION 1) (OUTCOME) )
  ( (TEST EXPRESSION 2) (OUTCOME) )
  ( (TEST EXPRESSION 3) (OUTCOME) )
)

COND – 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” …
COND – 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"))
)

This is just like an IF but with more than two possible logic branches ...
WHILE – Looping logic

(OBTAIN VARIABLE)
(WHILE (TEST EXPRESSION ON VARIABLE)
   (Statement 1)
   ...
   (Statement N)
)
WHILE – Pseudo Code

- Prompt the user for a number of spokes but the number of spokes needs to be no less than 20 for our purposes.
- We need to make sure the user gives us a valid integer number greater than or equal to 20 ...
WHILE – Real Code

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

The WHILE function loops around until the test case is satisfied ...
Let’s Try a Problem

Using the concepts we’ve learned so far for branching and looping logic.

We’ll use the pseudo code approach to build our logic as we go.
Example Problem (Pseudo Code)

- 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

- Hint: Use WHILE to qualify the input, use COND to evaluate the cases …
Example Problem (Real Code)

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

(cond
  ((< spokes 10) (alert "SMALL WHEEL")
   (if (and (>= spokes 10) (<= spokes 15)) (alert "MEDIUM WHEEL")
    (if (> spokes 15) (alert "LARGE WHEEL")
     ))
)
Points and Lists

How do points work?

How can we decompose points and put them back together again?
Get some points like this ...

(setq pt1 (getpoint "\nSelect a point: "))
(setq pt2 (getpoint "\nSelect a point: "))

You now have two 3D points stored in the variables PT1 and PT2 ...
CAR, CADR and CADDR …

- Decompose points into their X, Y and Z coordinates like this:
  - The X value of PT1 would be: (car pt1)
  - The Y value of PT1 would be: (cadr pt1)
  - The Z value of PT1 would be: (caddr pt1)

Why? CAR and CDR concatenation

Just memorize these functions …
LIST: Pseudo Code …

- 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 (points supplied by the user).
- Further, the radius of the circle should have a value of 11.5 …
LIST: Real Code ...

(setq pt1 (getpoint "\nSelect a point: "))
(setq pt2 (getpoint "\nSelect a point: "))
(command "\.circle"
  (list (car pt1) (cadr pt2) (caddr pt2))
  "11.5"
)

Hint: LIST is simply a point in this case ...
Let’s Try A Couple More Problems

Let’s ramp up the complexity a bit using techniques we already know.

I’ll throw in some (command) line access to spice it up a bit.
Let’s Draw a Wheel: Pseudo Code

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 …
Let’s Draw a Wheel: Real Code

(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" ""))
   ((> spokes 15) (command ".line" wheel_center "@12<0" ""))
)
(command "-array" "last" "p" wheel_center spokes "360" "y")
The Array syntax?

- How did I know how to write the ARRAY logic?
- Here are 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 and write it down ...
A Better Wheel: Pseudo Code

How would you change the program to assign all the spokes to the layer SPOKES which has a color of YELLOW?
(defun c:wheel ()
(setq spokes 0)
(while (or (< spokes 8) (> spokes 24))
  (setq spokes (getint "\nEnter number of spokes [from 8 to 24]: ")
)
(setq wheel_center (getpoint "\nSelect center point of wheel: ")
(setq old_layer (getvar "clayer"))
(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")
(setvar "clayer" old_layer)
(princ)
)
Working with Sets

Working with sets of objects.

SSGET
Using SSGET (Pseudo Code)

Let’s say that you wanted to build a program that would find all the circles in your drawing.

First, let’s draw a circle in AutoCAD and use the GE function from the previous section to get an idea of how the circle is constructed.
The association list

Here’s the resulting list from a circle entity:

```
((-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.
Use SSGET to Filter

- Here’s an example of finding all circles:
  (setq circle_set (ssget "x" '((0 . "CIRCLE"))) )

Note the “dotted pair” (0 . "CIRCLE")
Use SSGET for multiple Filters

- Here’s an example of finding circles on layer GEOM:
  (setq circle_set (ssget "x" '((0 . "CIRCLE")
  (8 . "GEOM")))
)

Note: It’s all about using “dotted pairs” to stack up filtering criteria like entity type or layers …
What would this find?

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

Hint: Nobody has this stuff memorized, you can find information on the formats in AutoCAD’s “Developer’s Help” section or use the GE function to figure it out.
What would these statements do?

(setq circle_set (ssget "x" '((0 . "CIRCLE") (62 . 2)) ))
(command "erase" circle_set "")

Note: You can use the set with any AutoCAD command with the convenience of storing the sets as variables that you can create with filters.
What would these statements do?

(setq circle_set (ssget "x" '((0 . "CIRCLE") (62 . 2)) ))
(command "change" circle_set "" "p" "la" "temp" "")
Challenge:

Could you write a program that would ask the user for a layer name then change all objects found on that layer (assuming it exists) to the layer NEW that has a color of GREEN?

Pseudo code:
Prompt for layer name
Find objects on layer
Use –LAYER to create new layer
Use CHANGE to change objects
More Bits and Bytes

- Loading programs
- Centralized loading
- VLIDE environment
- Compiling
VLIDE environment ...

- You can write code in the VLIDE window like this:
Compile your code …

- Use the VLIDE environment like this:
  
  (vlisp-compile ‘st “c:\test\myprog.lsp”)

- You’ll get MYPROG.FAS as a result
Load compiled code ...

- Use a LOAD statement like this:

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

- Now your LSP code is secure!
- Be sure not to lose your LSP code though!
Questions?

AutoLISP® / Visual LISP® - Part 1

Robert Green
CAD-Manager.com