



Autodesk
University
2007

The AutoLISP® Crash Course

Robert Green – Robert Green Consulting Group

CP105-2

AutoLISP/Visual LISP is a powerful way to extend AutoCAD functionality, but many avoid using it because they think it's too hard to learn. This class starts at the beginning and rapidly moves you through concepts such as Lists, accessing the command line, storing and retrieving variables, writing your own command functions, working with simple sets, and controlling AutoCAD software's start-up to achieve greater standardization of your AutoCAD environment. If you want to tap the power of AutoLISP, fasten your AutoCAD seatbelt for this session.

About the Speaker:

Robert is head of the Robert Green Consulting Group and a 13-year veteran speaker at Autodesk University. You've likely read his work in *Cadalist* 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 21 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.

Web site: www.CAD-Manager.com



Origins and expectations

AutoLISP is a descendant of Common Lisp, which is actually a very old programming language that was conceptualized for artificial intelligence applications. It also happened to be open source and was therefore an ideal (and cost-free) way for AutoCAD to be extended with a programming language. Since AutoLISP doesn't have to be compiled, meaning you can run programs directly, it was even more ideal as almost nobody but hard-core coders even knew what a program compiler was in the mid-1980's.

AutoLISP is a very powerful language that can take years to learn and master. I've been working with it since 1990 and I still learn new things about AutoLISP on every project. Having said this, I'd like to say that there is a lot you can do with AutoLISP without having to be a programming jock if you approach things with a cookbook-style approach and use existing routines.

Key files, functions and variables

The first thing to understand is that AutoLISP has some key files and a key function that perform startup operations for you. The key files can contain custom AutoLISP code that you want loaded whenever AutoCAD starts up. There are also a few useful functions that AutoLISP has predefined to perform startup operations for you. I'll summarize everything here:

ACAD200?.LSP	Names like ACAD2005.LSP, ACAD2006.LSP, ACAD2006.LSP, ACAD2008.LSP, etc. This file loads first when AutoCAD starts up.
ACAD.LSP	This file loads second when AutoCAD starts up.
ACAD200?DOC.LSP	Names like ACAD2005DOC.LSP, ACAD2006DOC.LSP, ACAD2007DOC.LSP, ACAD2008DOC.LSP, etc. This file loads third when AutoCAD starts up.
ACADDOC.LSP	This file loads fourth when AutoCAD starts up.
MENUNAME.MNL	This file loads after the previous files but only when its menu counterpart (either an MNU or MNC file) is loaded into AutoCAD. By placing AutoLISP code in this file you can be assured that the code will be loaded into memory ONLY when the parent menu is in use. Note that the code is only loaded once (like the ACAD.LSP) rather than with each drawing session (like the ACADDOC.LSP file). Plan accordingly.
ACADLSPASDOC	This variable controls the loading of ACAD.LSP. ACAD.LSP file only loads once, when AutoCAD first starts, unless you set the variable ACADLSPASDOC to 1 rather than the default setting of 0 .



EXERCISE: Key Files, Functions and Variables

To verify that you can create an AutoLISP file that loads and executes, follow these steps:

- Open a text editor like Notepad
- Type in the following line in the Notepad session taking great care to match all the parentheses and quotes as I have

(prompt “The ACADDOC.LSP file has loaded.”)

- Now save the file with the name ACADDOC.LSP and place the file in the SUPPORT folder of your AutoCAD application software. Typical locations would be as follows depending on your AutoCAD version:

C:\Program Files\AutoCAD 2006\Support

C:\Program Files\AutoCAD 2007\Support

C:\Program Files\AutoCAD 2008\Support

- Start AutoCAD and check your command prompt history (use the F2 key) to be sure the message “**The ACADDOC.LSP file has loaded**” shows up.
- Start multiple drawing sessions in the same session of AutoCAD and verify that the same message is displayed at the command prompt history as each drawing starts.

Recommendation

The above methodology is recommended for adding basic AutoLISP syntax to your AutoCAD environment. By using the ACADDOC.LSP methodology you don't have to worry about the ACADLSPASDOC setting and since the file probably doesn't exist in your installation already, there's essentially zero chance you'll write over or destroy any existing customization.

Note 1: *Using ACADDOC.LSP in the SUPPORT directory yields maximum results, low complexity and far less chance of messing something up.*

Note 2: *Be sure to use a plain text editor like Notepad rather than a word processing program like Word. Notepad will strip out weird characters like backward quote marks that can cause your program to malfunction.*



Syntax basics

So now that you know which files to put your AutoLISP code in, let's establish the basics of AutoLISP syntax so you can start writing some code. The basic topics you need to understand are:

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

We'll examine these topics using a combination of lecture and viewing code examples. Please understand that given our time constraints I can't fully develop all the theory behind these commands. I'll have to ask you to trust me for the meantime, then research the language more thoroughly later when you have a working knowledge of the language. Let's get started!

Lists and arguments

Anything you do in AutoLISP is formatted as a list. Lists can have both ARGUMENTS and FUNCTIONS embedded within them so you can perform various activities. Here are a few examples:

(+ 20 30) Here + is the FUNCTION and the two numbers are ARGUMENTS

(command "line" "0,0" "1,1" "") Here COMMAND is the function, all others are ARGUMENTS

If you study more you'll start to see that any AutoCAD entity can be described as a complex list like this:

```
((-1 . <Entity name: 400b1580>) (0 . "LINE") (330 . <Entity name: 40094cf8>) (5 . "378") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "2") (100 . "AcDbLine") (10 198.441 78.786 0.0) (11 247.87 179.589 0.0) (210 0.0 0.0 1.0))
```

Though it isn't obvious, this list describes a line that is drawn on layer "2" with a start point and end point and a UCS vector of 0,0,1. Once you start to read lists, you're on your way!

Lists are simply a way to group information or to submit information, in the form of ARGUMENTS to a FUNCTION as we illustrated above. The AutoCAD entity lists allow you to perform actions on AutoCAD entities via more advanced programming techniques than we'll look at today. I just want you to have a glimpse of what's under AutoCAD's hood.

Rules of AutoLISP

“For every (there is an equal and opposite)” – Einstein circa 1939

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)** or **(setq name Robert Green”)**

When formatting numbers always avoid “invalid dotted pairs.”

Like this: **(setq A 0.5)**

Not like this: **(setq A .5)**

Arithmetic functions a little different than you’re used to. For example:

<u>Standard language</u>	<u>AutoLISP equivalent</u>	<u>Result</u>
8 + 4	(+ 8 4)	12
8 + 2.3	(+ 8 2.3)	12.3
5.0 ÷ 4.0	(/ 5.0 4.0)	S1.25
5 ÷ 4.0	(/ 5 4.0)	1.25
5.0 ÷ 4	(/ 5.0 4)	1.25
5 ÷ 4	(/ 5 4)	1 (Why? Integers!)

More complex math gets a little persnickety. Look at the following examples and remember that AutoLISP works from the inside out with respect to parentheses. For example:

<u>Standard language</u>	<u>AutoLISP equivalent</u>	<u>Result</u>
(8 + 4) * 3	(* (+ 8 4) 3)	36
(3.0 * 5.0) ÷ 4.0	(/ (* 3.0 5.0) 4.0)	3.75
((4 + 4) * 3) ÷ 2	(/ (* (+ 4 4) 3) 2)	12

I’ll talk more about data types in an upcoming session but for now take careful note that how you enter numbers (real or integer) has everything to do with how the numbers get processed.



Accessing the command line

To make AutoCAD and AutoLISP shake hands, the most basic skill set you'll need is accessing the command line. I alluded to the idea of a COMMAND function with arguments in the previous section but I didn't do any real practical examples. Let's have a look at several possible examples of "talking" to AutoCAD via the AutoLISP command line interface.

If you wanted to draw a line between two user-supplied points on the screen, you'd do the following:

- 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 you'd do it this way:

(command "line" pause pause "")

Note: The PAUSE instruction waits for the user and the "" is equal to hitting the ENTER key. Why? Because that's the way the good folks at Autodesk set it up!

How about a few more examples of command line access that you may want to actually use? Here are a few examples:

<u>AutoLISP function</u>	<u>Purpose</u>
(command "viewres" "y" "5000")	Sets view resolution for no jaggy circles
(command "-color" "BYLAYER")	Sets color to BYLAYER
(command "-linetype" "set" "BYLAYER" "")	Sets linetype to BYLAYER
(command "menu" "menuname.mnc")	Loads menu file of your choosing

Now let's look at an example with user input:

<u>AutoLISP function</u>	<u>Purpose</u>
(command "viewres" "y" pause)	Sets view resolution to a user value



Variables

While knowing about lists is one thing, doing something functional with lists depends on being able to store and pass variables within lists.

First things first: How do you store a variable? Let's say you wanted to store a variable called VALUE that has the number 12.7 associated with it. In order to set the variable we need to use the SETQ function (think SET eQual) like this:

```
(setq VALUE 12.7)
```

This particular variable would be a REAL NUMBER variable. Some other variable types are shown here for your reference:

```
(setq VALUE 1)                      This would be an INTEGER variable
```

```
(setq VALUE "Happy")              This would be an STRING variable
```

```
(setq VALUE 1)                      This would be an INTEGER variable
```

Now when you interact with AutoCAD you have to pass the variable in the command line. Use this case as an example:

```
(setq VIEWRES_VALUE 5000)  
(command "viewres" "y" viewres_value)
```

What would happen in this case?

```
(setq VIEWRES_VALUE "Happy Birthday!")  
(command "viewres" "y" viewres_value)
```

This wouldn't make much sense, would it? When you write a program you have to make sure that the variables you use match up with the AutoCAD commands the variables are "speaking" to. It makes sense to pass a numeric value to the VIEWRES command, but it doesn't make any sense to send it a string-type variable. Understanding this simple trick of variable typing makes things a lot easier.



User input

Let's say you wanted to write a program that would create a wheel with some spokes in it. Well, how many spokes should the wheel have? If you write a program with a known number of spokes, say 8, the program could then draw wheels with 8 spokes. Wouldn't it be better to write a program that draws wheels with a variable number of spokes? For that matter, how about writing a program where you specify the number of spokes and the diameter of the wheel?

In order to specify how many spokes are on the wheel and how big the diameter is, you'll need to ask the user to define those variables like this:

(getdist "\nInput a distance: ")	Prompts the user for a distance value
(getreal "\nInput a real number: ")	Prompts the user for a real number value
(getint "\nInput an integer number: ")	Prompts the user for an integer value
(getstring "\nInput text: ")	Prompts the user for a string/character value

Now let's associate the input the user gives us to a named variable like this:

```
(setq user_dist (getdist "\nInput a distance: "))  
(setq user_real (getreal "\nInput a real number: "))  
(setq user_int (getint "\nInput an integer number: "))  
(setq user_string (getstring "\nInput text: "))
```

Or, to revisit our spoke/wheel analogy, how about this:

```
(setq wheel_dia (getdist "\nEnter wheel diameter: "))  
(setq spokes (getint "\nEnter number of spokes: "))
```

User command functions

OK, we know how to collect information from the user and we know how to pass that input to AutoCAD's command line. But how can we do all this in a way that adds to AutoCAD's native set of commands? After all, wouldn't it be ideal if we could write AutoLISP routines that would trick AutoCAD into thinking it had new commands? If we do a good enough job, our CAD users won't even know we've programmed commands for them.

Let's have a few simple examples so you can see how it works:

The idea: We want to create a ZA command that does a zoom all.

The code:

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

Here's what the routine is doing on a line-by-line basis:

(defun C:ZA ()	Reserves a command called ZA with no inputs to the command which explains the () after the command name. Don't worry about why the () works as it does now; to get more into this would require much more detailed coverage.
(command “.zoom” “a”)	Engages the command line and passes the arguments .ZOOM and A for all to the command processor. Why the “.” in front of zoom, you ask? Because it guarantees that we'll use the native ZOOM command instead of any other version that may be redefined. More on this later.
(princ)	Clears the print console and suppresses AutoLISP from leaving a NIL result at the command line as the function terminates. Don't worry about why it works, it just does and your program will look a lot cleaner for using it.
)	Closes the function out. This actually balances the misbalanced paren count in the first line of the function. (You did notice the imbalance, right?)

Here are a few more I like to use:

```
(defun C:VR ()  
  (command “viewres” “y” “5000”)  
)  
  
(defun C:BL ()  
  (command “-color” “BYLAYER”)  
  (command “-linetype” “set” “BYLAYER” “”)  
  (princ)  
)
```



More user command functions

Let's look at some additional functions and try to figure out why they work and crank up the complexity just a little to keep things interesting. I'll comment on the code in comment form (that is, the code includes a ";" character which allows us to write comments in our programming). Here goes:

The idea: We want to create an FZ command that performs a fillet command but makes sure that the radius is set to 0.0 so we fillet sharp corners.

```
(defun c:fz () ; Define an FZ command
  (setvar "filletrad" 0.0) ; Set the fillet radius to 0.0 using SETVAR
  (command ".fillet" pause pause) ; Invoke FILLET and wait for two user inputs
  (princ) ; Clear the command line
)
```

We learned that we can set AutoCAD variables using the SETVAR function very easily. It turns out that you can set a variable using SETVAR and can read the value of any variable using, cleverly enough, GETVAR. Don't know which variable you need? Look in the AutoCAD help file under the command topic SETVAR and you'll find a list of all the variables and what they do!

The program will work but it would be better if we checked the value of the fillet radius first and then made sure we put it back the way we found it when our program is done running. Here's how:

```
(defun c:fz () ; Define an FZ command
  (setq old_filletrad (getvar "filletrad")) ; Store the old fillet radius value
  (setvar "filletrad" 0.0) ; Set the fillet radius to 0.0 using SETVAR
  (command ".fillet" pause pause) ; Invoke FILLET and wait for user inputs
  (setvar "filletrad" old_filletrad) ; Put the fillet radius back
  (princ) ; Clear the command line
)
```

This example illustrates reading a system variable value into an AutoLISP variable, then using that variable later in the program to AutoCAD's fillet radius to its previous value. That's it! Not so bad, really.

Auto Purge (ATP)

Here's a function I like to use daily. It does an autopurge and save (thus combining two commands into one) by keying in ATP:

```
(defun c:atp ()
  (command "-purge" "a" "*" "n" ".qsave")
  (princ)
)
```

Cool stuff to know

There are a few other little handy things to sprinkle into your AutoLISP sessions that I'll cover here and comment on one by one:

Comment “;” Lines

You may have noted my use of the semi-colon character in some examples. The semi-colon is the comment character in AutoLISP which allows you to embed notes in the code. I find commenting very helpful because several months from now when I'm working on a program I won't remember what I was doing way back when I wrote the program.

Anything that comes after the semi-colon is simply ignored until the next line of the program starts.

Dash “-“ Form Commands

Why did I use –PURGE instead of just PURGE in the ATP example?

I used the dash form of the PURGE command specifically because I didn't want the dialog form of the command invoking. While the dialog form of commands like PURGE and LAYER are cool for the user, an AutoLISP command can't reach out and grab the mouse to interact with the command, right?

So if a command has a dialog box, like LAYER for example, just try putting a dash “-“ in front of the command and see if it works!

I prefer using the dash form of commands because I can always go to AutoCAD's command prompt and type in the dash form of the command and see exactly how it will behave. I therefore make fewer mistakes writing my programs!

Extra credit: *If you're a glutton for punishment look in the AutoLISP function reference (find it under the AutoCAD help system) and look up the INITDIA function. This AutoLISP function provides a more detailed explanation of how dialogs are handled within AutoLISP functions.*

Dot “.“ Form Commands

In my FILLET programming example, why did I type the command as “.FILLET” rather than just “FILLET”?

I used the dot form of the command just in case some enterprising CAD manager had undefined the FILLET command previously. The dot form of the command allows you to call the command even if has been undefined. We'll talk more about undefining functions shortly.

Alerting Users

You can send a message to the user like this: **(alert “Message goes here”)**



Note: *The user must acknowledge the alert by clicking the OK button!*

Undefining commands

You can turn AutoCAD commands off like this:

```
(command “.undefine” “LINE”)
```

```
(command “.undefine” “TORUS”)
```

This way, if you don't want users messing with a command, you just shut it off. Place the code in your ACAD.LSP file and it'll have the equivalent effect of disabling AutoCAD's command vocabulary.

Undefining and alerting

You can undefine a command and alert the user like this:

```
(command “.undefine” “TORUS”)
```

```
(defun C:TORUS ()
```

```
  (alert “I don't want you using that command!”)
```

```
  (princ)
```

```
)
```

Redefine via AutoCAD

To bring a command back simply use the REDEFINE command at the AutoCAD command prompt like this:

```
Command: redefine
```

```
Enter command name: torus
```

Redefine via AutoLISP

Now take it another step and redefine the function with an alert to the user like this:

```
(command ".undefine" "TORUS")

(defun C:TORUS ()
  (alert "I don't want you using that command!")
  (princ)
)
```

This way the user will get the message!

Controlling Command Echo

Many times you may have a program that invokes a command, yet you don't want the user to see the command you've invoked. As an example, the BL function I wrote earlier could be made to run in STEALTH MODE by doing this:

```
(defun C:BL ()
  (setvar "cmdecho" 0) ; Turns the echo off
  (command "-color" "BYLAYER")
  (command "-linetype" "set" "BYLAYER" "")
  (setvar "cmdecho" 1) ; Turns the echo back on
  (princ)
)
```

Downloads Available

I've found CAD Depot to be a very useful source for public domain AutoLISP utilities you can use as inspiration for your own routines. You can find the AutoLISP area at:

<http://www.caddepot.com>

Besides AutoLISP routines, there's a lot of other nice material at CAD Depot worth some of your browsing time.



Autodesk
University
2007

The AutoLISP® Crash Course

Want the PowerPoint?

I'll be happy to send you a copy of the session PowerPoint presentation. Just send an email to me at rgreen@cad-manager.com and be sure to put **CP105 - PowerPoint** in the subject line so I'll know which class you attended.

I'll send out PDF captures of the PowerPoint files upon my return to Atlanta.

Reference Materials

You can find a wide range of information on CAD management and business metrics at my www.CAD-Manager.com web site.

Also be sure to check out the AutoCAD help resources under the DEVELOPER GUIDE section for a wealth of information on AutoLISP commands and functions. You may even want to try working the GARDEN PATH example tutorial now that you have a bit of a grasp of how to deal with AutoLISP.

Happy reading and happy coding!