



November 30 – December 3, 2004 ♦ Las Vegas, Nevada

Seven LISP Functions of Highly Effective CAD Users

Anthony P. Meulemans – Jacobs Sverdrup, Technology Group

CP33-3 This course uses seven AutoLISP® functions to teach the basics of the AutoCAD® programming language. These functions can be used by even the novice programmer to create effective commands. This course is designed for the CAD user without any programming experience.

Who Should Attend

AutoCAD users with little or no programming experience

Topics Covered

- *getvar* and *setvar* - basic AutoLISP syntax and data types
- *setq* - AutoLISP symbols
- *command* and *ssget* - interacting with AutoCAD
- *defun* and *princ* - creating your own AutoCAD shortcuts
- Automatically loading your AutoLISP shortcuts into AutoCAD

About the Speaker:

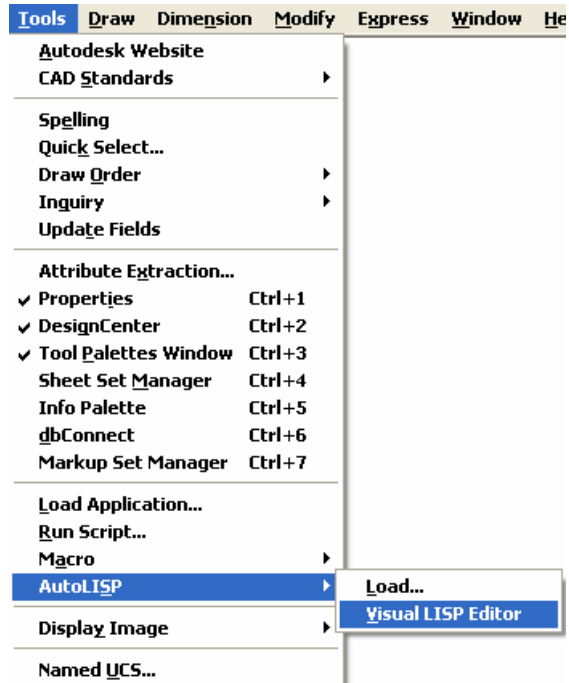
Anthony has 28 years of experience in the engineering field. For the past 7 years, he has been the CADD System Administrator at Jacobs Sverdrup Technology Group. The Technology Group specializes in the design and construction of test facilities for the aerospace and automotive industries. For 9 years previously, he was the manager and principal instructor at an Autodesk® Authorized Training Center.

Introduction

AutoLISP applications or routines can interact with AutoCAD in many ways. These routines can prompt the user for input, access built-in AutoCAD commands directly, and modify or create objects in the drawing database. By creating AutoLISP routines, you can add discipline-specific commands to AutoCAD.

In the past, developing AutoLISP programs for AutoCAD meant supplying a text editor for writing code, then loading the code into AutoCAD and then running it. AutoLISP has been enhanced with Visual LISP (VLISP), which offers an integrated development environment (IDE) that includes a compiler, debugger, and other development tools to increase productivity.

Because AutoCAD can read AutoLISP code directly, no compiling is required. While Visual LISP provides an IDE, you may choose to experiment by entering code at the Command prompt, which allows you to see the results immediately. This makes AutoLISP an easy language to experiment with, regardless of your programming experience.



Basic AutoLISP syntax and data types

The AutoLISP Expression

- An AutoLISP expression is the most fundamental part of writing programs.
- An AutoLISP expression has the syntax:
(function argument)
 - Every expression has opening and closing parentheses.
 - Each element within an expression must be separated by “white space”.
 - The number of spaces is variable but there must be at least one.
 - Every expression has a function name.
 - The name must immediately follow the opening parentheses.
 - Every expression is evaluated (executed) and returns a result.
- A function is a subroutine that tells AutoLISP what task to perform.
- An argument provides data to the function.
 - A function may have any number of arguments or none at all.
 - Arguments may be symbols (variables), literals, or other functions.
 - Some arguments are flags or option parameters that alter the action of the function.

- If a function is defined to accept an argument, you must provide the function with a value for the argument.

(getvar *variable_name*)

- Retrieves the value of an AutoCAD system variable
 - The *variable_name* must be a quoted string.

Examples:

```
(getvar "FILLETRAD") ;returns the current fillet radius  
(getvar "CLAYER") ;returns the current layer
```

AutoLISP Data Types

- AutoLISP expressions are processed according to the order and type of data type used within the parentheses. To fully utilize AutoLISP, you must understand the differences in the data types and how to use them.

Integers

- Integers are whole numbers that do not contain a decimal point.
- AutoLISP integers are 32-bit signed numbers with values ranging from +2,147,483,647 to -2,147,483,648.

Examples:

```
0  
-56  
4133
```

Reals

- A real is a number containing a decimal point.
- Numbers between -1 and 1 must contain a leading zero.
- Real numbers are stored in double-precision floating-point format, providing at least 14 significant digits of precision.
- Reals can be expressed in scientific notation, which has an optional e or E followed by the exponent of the number (for example, 0.0000041 is the same as 4.1e-6).

Examples:

```
3.1415927  
0.25  
-54.321
```

Strings

- A string is a group of characters surrounded by quotation marks. Strings must be surrounded by double quotes (“”).
- Within quoted strings, the backslash (\) denote the following character as a control character (also referred to as escape codes).

Seven LISP Functions for Highly Effective CAD Users

Examples:

```
"GRIDMODE"  
"1234.5"  
"\nMy AutoLISP loaded"  
"" ;Null string
```

Control Characters:

\\	Backslash character
\"	Quote (") character
\n	Newline character
\t	Tab character

Symbols

- AutoLISP uses symbols to refer to data. Symbols are sometimes referred to as variables in other languages.
- Symbol names are not case-sensitive and may consist of any sequence of alphanumeric characters with the following exceptions:
 - May not consist of only numeric characters.
 - May not contain spaces.
 - May not contain the following special characters: () ` , " ; \ .
- Do not name any symbol the name of any internal AutoLISP function or the predefined symbols: *pause*, *pi*, *nil*, and *t*.

Entity-Names

- An entity name is a numeric label assigned to objects in a drawing. It is actually a pointer into a file maintained by AutoCAD, and can be used to find the object's database record and its vectors (if they are displayed).

Example:

```
<Entity name: 27f0540>
```

- This label can be referenced by AutoLISP functions to allow selection of objects for processing in various ways.

VLA-Objects

- Objects in a drawing may be represented as Visual LISP ActiveX (VLA) objects, a data type introduced with Visual LISP.

Selection-Sets

- Selection sets are groups of one or more entity names that may be referenced as a single group.
- You can interactively add objects to, or remove objects from, selection sets with AutoLISP routines.
- These selection sets may be referenced by any AutoCAD command that recognizes the "Last" option.

File Descriptors

- A file descriptor is a pointer to a file opened by the AutoLISP **open** function. The **open** function returns this pointer as an alphanumeric label.

Example:

```
#<file "c:\\projdwgs.csv">
```

- You supply the file descriptor as an argument to other AutoLISP functions that read or write to the file.

Lists

- An AutoLISP list is a group of related values separated by spaces and enclosed in parentheses.

- Three types of common lists are quoted, point, and entity lists:

Quoted Lists

- If a list is preceded by a ' (quote) it evaluates to itself and is not executed, these lists may be thought of as data.

Example:

```
' ("Monday" "Tuesday" "Wednesday" "Thursday" "Friday")
```

Point Lists

- A point is a list with a fixed data structure and is the primary form of data used when dealing with graphics.
- A point is a list of two or three real numbers in (X-value Y-value Z-value) format.

Examples:

```
2D point list: (2.0000 3.0000)
```

```
3D point list: (2.0000 1.0000 0.0000)
```

Entity Lists

- Entity lists are an association list containing the description of the entity.
- This association list is built using dotted pairs (sub-lists).
- These dotted pairs consist of a group code and data field.
- The group code is the same as the group codes used in the “DXF” file definition.

Example:

```
((-1 . <Entity name:60000164>) (0 . "LINE") (8 . "CENTERLINE")
(6 . "CENTER") (62 . 1) (5 . "7") (39 . 1.0) (10 2.0 3.0 0.0)
(11 4.0 4.0 0.0) (210 0.0 0.0 1.0))
```

(setvar variable_name value)

- This function sets a system variable *varname* to *value*, and returns that value.
 - The *variable_name* must be a quoted string.

Examples:

```
(setvar "CLAYER" "0") ;Set the current layer to 0
```

```
(setvar "FILLETRAD" 0.125) ;Set the fillet radius to 0.125
```

```
(setvar "LUPREC" 5) ;Linear Unit Precision
```

```
(setvar "SNAPMODE" 0) ;Sets Snap Mode Off
```

System Variables in AutoLISP

- Many of the modes, sizes, and limits AutoCAD uses are stored in a collection of system variables.
- System variables are valuable when creating AutoLISP routines that change AutoCAD drawing parameters.

Setting AutoLISP Variables

- Variables are the basic means by which a program can organize, remember, and recall information. In AutoLISP, variables are called symbols.
- When a symbol is created, it is given a name.
 - After a symbol is created and named it can then receive a value.
 - When a symbol has an assigned value, it is said to be bound to that value.
- Whenever the symbol name is encountered by a function AutoLISP substitutes the value of the symbol for the symbol name.
- The AutoLISP function that will create symbols and bind a value to it is *setq*.
- The *setq* function can assign multiple symbols in one call to the function.

(setq *symbol expression* [*symbol expression*] ...)

- This function assigns the value of evaluating *expression* to *symbol*.
- This function returns the new value assigned to *sym*.

Examples:

```
(setq i 1) ;Assigns the integer 1 to i, returns 1.0
(setq s "APM") ;Assigns the string "APM" to s, returns "APM"
(setq x 1.0 y 2.0) ;Assigns 1.0 to x and 2.0 to y, returns 2.0
```

- The current binding or value of a AutoLISP symbol may be used on the drawing editor's command line by simply putting an "!" exclamation point in front of the symbol name.

Nesting AutoLISP Expressions

- Nesting is putting an expression inside another expression as a substitute for an argument.
- Results returned by an expression can be used as arguments for other functions in a process called nesting.
- AutoLISP evaluates the innermost or deepest nesting level expression first and expressions from left to right that are at the same nesting level.
- Always keep an equal number of open and closed parentheses in every statement.
- LISP is sometimes known as, "Lost In Stupid Parentheses" due to the fact that it is a parentheses delimited language, that allows functions, atoms and lists to be nested.

Examples:

```
(setq o:fr (getvar "FILLETRAD")) ;Retrieves current fillet radius
(setvar "FILLETRAD" 0.0) ;Set fillet radius to 0.0
(setvar "FILLETRAD" o:fr) ;Resets fillet radius to original
```

Interacting with AutoCAD

(*command* [*arguments*] ...)

- The *command* function executes an AutoCAD command.
 - Any AutoCAD command that functions from the command line may be used.
 - A hyphen (-) before the command suppress the dialog box and display prompts on the command line instead.
 - Use a period (.) before the command name to insure you always access a built-in AutoCAD command.
 - SKETCH command cannot be used because it reads the digitizer directly.
- These arguments must correspond to the types and values expected by that command's prompt sequence; these may be strings, real values, integers, points, entity names, or selection set names.
 - Data such as angles, distances, and points can be passed either as strings or as the values themselves (as integer or real values, or as point lists).
 - A null string ("") is equivalent to pressing ENTER on the keyboard.
 - (*command*) - Invoking *command* with no argument is equivalent to pressing ESC and cancels most AutoCAD commands.
- The predefined symbol *pause* allows direct user input when it is encountered as an argument to the *command* function.
- The command function evaluates each argument and sends it to AutoCAD in response to successive prompts.

Examples:

```
(command ".POLYGON" "6" pause "C" pause) ;Create Hexagon
(command ".BREAK" pause "First" pause "@") ;Breaks at Point (no gap)
(command ".-BHATCH" "Properties" "Solid" pause "");Fills w/solid hatch
```

(*ssget*)

- The *ssget* function creates a selection set from the selected object.
- These selection sets may be used in any AutoCAD command that accepts the *Previous* or *Last* option for set selection, such as ERASE, COPY, MOVE, and ROTATE commands.
- Selection sets can contain objects from both paper and model space, but when the selection set is used in an operation, *ssget* filters out objects from the space not currently in effect.
- Selection sets returned by *ssget* contain main entities only (no attributes or polyline vertices).
- If (*ssget*) is used without any parameters AutoLISP will prompt the user with "Select entity: " in the same manor as other AutoCAD commands.

Example:

```
(setq sset (ssget));Selected items to modify
(command ".MIRROR" sset "" pause pause "Y") ;Mirror w/o copying
```

Create personal AutoCAD shortcuts

With AutoLISP, you can define your own functions. Once defined, you can use these functions at the AutoCAD Command prompt, or within other AutoLISP expressions, just as you use the standard functions. You can also create your own AutoCAD commands, because commands are just a special type of function.

(defun *symbol* ([*arguments*] [/ *variables...*]) *expression...*)

- The *defun* function defines a function.
- The *symbol* names the function.
 - Warning! Never use the name of a built-in function or symbol for the symbol argument to defun. This overwrites the original definition and makes the built-in function or symbol inaccessible.
 - If the *symbol* name is defined in the form of *C:xxx*, it can be issued at the AutoCAD Command prompt in the same manner as a built-in AutoCAD command.
 - The *C:* portion of the name must always be present in the *XXX* portion is a command name of your choice.
 - The function must be defined with no arguments.
- The *arguments* name the symbols for the values expected by the function.
- The *variables* name the local variables for the function.
 - The slash preceding the variable names must be separated from the first local name and from the last argument, if any, by at least one space.
- Any number of AutoLISP *expressions* to be evaluated when the function executes.
- If you do not declare any arguments or local symbols, you must supply an empty set of parentheses after the function name.
- If duplicate argument or symbol names are specified, AutoLISP uses the first occurrence of each name and ignores the following occurrences.
- Return the result of the last expression evaluated.

Examples:

```
(defun C:ZE () ; Zoom Extents
  (command ".ZOOM" "Extents")
  (command ".ZOOM" "0.95x"))
(defun C:LGD () (command ".LENGTHEN" "DYNAMIC")) ;Extend/Trim to Point
```

(princ [*expression*])

- The *princ* function prints an expression to the command line.
- The *expression* is either a string or an AutoLISP expression that returns a string.
 - Only the specific *expression* is printed; a newline or space character is not included.
- If *princ* is called without arguments, it returns a null.

Examples:

```
(princ "\nMy AutoLISP routines loaded")
      ; returns 'My AutoLISP routines loaded' to the command line
(princ) ; returns null (nothing)
```

Sample ACADdoc.lsp:

```
(setvar "AUPREC" 1) ;Angular Unit Precision
(setvar "GRIDMODE" 0) ;Grid Off
(setvar "LUPREC" 5) ;Linear Unit Precision
(setvar "MIRRTEXT" 0) ;Mirror Text Off
(setvar "ORTHOMODE" 0) ;Ortho Off
(setvar "SNAPMODE" 0) ;Snap Off

(defun C:ALT () (command ".DIMEDIT" "New" "<>\X[]") (princ)) ;Adds alternate units to dims
(defun C:ARA (/ o:sna sset) (command ".UNDO" "BEGin") (setq o:sna (getvar "SNAPANG"))
  (setvar "CMDECHO" 1) (setq sset (ssget)) (command ".SNAP" "Rotate" "" pause)
  (command ".ARRAY" sset "" "R" pause pause pause pause)
  (command ".SNAP" "Rotate" "" o:sna) (command ".UNDO" "End") (princ)) ;Array at Angle
(defun C:ATR () (setvar "CMDECHO" 1) ;Rotate and Reposition Attribute Values
  (command ".ATTEDIT" "Y" "" "" "" pause "" "Angle" "0" "Position" pause "") (princ))
(defun C:BAA () (command ".UNDO" "BEGin") (setvar "CMDECHO" 0)
  (princ "\nSpecify internal point of area: ") (command ".-BOUNDARY" pause "")
  (command ".AREA" "Object" "Last") (command ".Erase" "Last" "") (setvar "CMDECHO" 1)
  (command ".UNDO" "End") (princ (getvar "AREA"))) ;Reports Area by Picking Point
(defun C:DIA () (command ".DIMEDIT" "New" "%C<>") (princ)) ;Adds diameter symbol to dims
(defun C:F0 (/ o:fr o:fr) (setq o:fr (getvar "FILLETRAD")) (setvar "FILLETRAD" 0)
  (setq o:tm (getvar "TRIMMODE")) (setvar "TRIMMODE" 1) (setvar "CMDECHO" 1)
  (command ".FILLET" pause pause) (setvar "FILLETRAD" o:fr) (setvar "TRIMMODE" o:fr)
  (princ)) ;Trim or Extend 2 Items to Intersection Point
(defun C:JOIN (/ o:pea sset) (setq o:pea (getvar "PEDITACCEPT"))
  (setvar "PEDITACCEPT" 1) (setq sset (ssget))
  (command ".PEDIT" "Multiple" sset "" "Join" "0" "")
  (setvar "PEDITACCEPT" o:pea) (princ)) ;Joins selected to polyline
(defun C:LIM () (command ".LIMITS" (getvar "EXTMIN") (getvar "EXTMAX")) (princ))
(defun C:O1 (/ o:od) (setq o:od (getvar "OFFSETDIST"))
  (setvar "OFFSETDIST" -1) (command ".OFFSET" "" pause pause "")
  (setvar "OFFSETDIST" o:od) (princ)) ;Offsets selected object
(defun C:PAL () (command ".-LAYER" "Set" "0" "") (command ".PURGE" "All" "" "N")
  (command ".PURGE" "All" "" "N") (princ)) ;Purges all unused symbols
(defun C:SCI (/ sset) (setvar "CMDECHO" 1) (setq sset (ssget))
  (command ".SCALE" sset "" pause "10/254") (princ)) ;Scales selected to Inch
(defun C:SCM (/ sset) (setvar "CMDECHO" 1) (setq sset (ssget))
  (command ".SCALE" sset "" pause "25.4") (princ)) ;Scales selected to Metric
(defun C:SFH (/ o:hpn) (setq o:hpn (getvar "HPNAME")) ;Fills area w/solid hatch
  (command ".-BHATCH" "Properties" "Solid" pause "") (setvar "HPNAME" o:hpn) (princ))
(defun C:ZE () (command ".ZOOM" "Extents") (command ".ZOOM" "0.95x") (princ)) ;Zoom Extents
(princ "\n\tA. P. Meulemans - User Commands Loaded")
```

Seven LISP Functions for Highly Effective CAD Users

Load your AutoCAD shortcuts

- AutoLISP applications are stored in ASCII text files with the *.lsp* extension.
- An AutoLISP application must first be loaded before it can be used.
 - Loading an AutoLISP application loads the code from the LSP file into the system's memory.
- The APPLOAD command loads applications and specifies which applications are to be loaded at start-up.
- If the file is named *acaddoc.lsp* and is stored in a support path folder (directory), AutoCAD will load the contents of the file automatically with the opening of each individual drawing.

