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

# The AUGI® LISP Forums Greatest Hits for Newbies

Peter Jamtgaard P.E. – Cordeck Sales, Inc.

**CP24-1** The AUGI LISP Forum is an online community of AutoCAD® users who exchange information and help each other learn LISP programming, solve problems, and improve productivity. In this course, we'll review a collection of basic, member-developed AutoLISP® and Visual LISP® routines to familiarize you with techniques such as defining functions, entities, ActiveX manipulation, and more. We'll demonstrate and thoroughly explain each program and take your questions. We'll also take your suggestions and develop a new routine using these techniques to demonstrate how to create your own tools. This session is designed for beginning LISP programmers.

Who Should Attend
Beginning-level LISP programmers

Topics Covered
* Defining functions
* Lists
* Variable types
* User input
* Command line-driven functions
* Entities and entity lists
* ActiveX object manipulation

**About the Speaker:**
Peter has 18 years of experience running AutoCAD® in a multidiscipline environment, including Civil Engineering, Structural Engineering, Mechanical, Architectural, and many others. He has served on the AUGI® Board of Directors and is the AUGI programming chair. Peter has been programming AutoLISP® for 18 years and is experienced with VB(A). He holds an Associates degree in Civil Engineering Technology, a B.S. degree in Civil and Environmental Engineering and is a licensed Professional Engineer in the State of Wisconsin.

Email: **cordeck@acronet.net**

## Example 10: Layerzero.lsp

**LayerZero.lsp** is a routine that will change a single selected entity to layer "0" using an old fashioned AutoLISP entity list modification technique. Before the introduction of Visual LISP® this was the way most object modification was performed.

```
; This routine will change a selection to layer "0" using Entity List modification
(defun C:LayerZero (/ lstSelection ; list of selected item and selection point
                      entSelection ; Entity name of selection
                      lstEntity    ; Entity List of selection
                    )
(if (setq lstSelection (entsel "\nSelect item on screen: "))
 (progn
  (setq entSelection (car lstSelection)
        lstEntity    (entget entSelection)
        lstEntity    (subst (cons 8 "0")(assoc 8 lstEntity) lstEntity)
  )
  (entmod lstEntity )
 )
)
```

A semicolon **";"** in front of a line in a routine is a comment and will not be evaluated at runtime.

To create a new command LISP has the "define function" expression **defun.** The **defun** Expression has this format: **`(defun symbol ([arguments] [/ variables...]) expr...)`** .

In the above routine the **defun** expression defines the new symbol function named **LayerZero**. The **C:** in front of the routine name means the routine will be available at the command line. (It does not mean the C drive on your computer.) Since it is a command line function it will have no arguments passed to it. It has three local variables **lstSelection, entSelection** and **lstEntity**. A **local variable** is set to **nil** at the beginning of the routine and reset to **nil** at the end of the routine, and will not affect other variables with the same name in other routines. The symbol **nil means nothing** or the variable is empty. To improve readability I have decided to use a standard variable naming technique that uses prefixes to identify variable types. It is commonly known as **Reddick** variable naming conventions.

The first local variable **lstSelection** is a list. Lists are groups of different values that are stored together in a specific order. The list expression hes the form **`(list [expr...])`**. The prefix **lst** helps your recognize it as a list. The second local variable is **entSelection**. The prefix **ent** helps you recognize it as an entity name. Every object in a drawing has a unique entity name and that is how programs can access and modify them. The third local variable is **lstEntity**. It is also a list, but also a special kind of list called an **entity list**. An entity list is a series of pairs or sub-lists that describe the entity. To elaborate lets step through the routine listed above.

**`(entsel "\nSelect item on screen: ")`** .This expression prompts the user to "Select item on screen: ", and the little **\n** instructs the prompt to appear on a new line. When the user selects an entity on the screen it will return a list of the entity name and a list representing the point selected. For example:

**(<Entity name: 7efa4dc8> (467.375 257.063 0.0))**

The above routine has the expression **`(setq lstSelection (entsel "\nSelect item on Screen: "))`.** The setq expression sets the variable l**stSelection** to the value returned from the **entsel** expression. Reading lisp is very similar to reading a mathematical expression in algebra, you solve the innermost expression first.

The **setq** expression has the format **`(setq sym expr [sym expr]...).`** The sym or variable name is set to each following expression. Multiple items can be set within one **setq** expression. The variables **entselection** and **lstEntity** are both set inside one **setq** expression in the example.

The **setq** expression is inside an **if** statement. This is called nesting or putting one expression inside another. The **if** expression has the format **`(if testexpr thenexpr [elseexpr]).`** The **testexpr** is the (setq lstSelection…) shown above and the **thenexpr** is surrounded by the **progn** expression. The **progn** expression is necessary to group together the expressions of the **thenexpr**.

It the user doesn't select anything or cancels the routine the **testexpr** expression will return a **nil**. In that case the routine needs to stop. The **if** statement recognizes the **nil** as a false condition and jumps over the **thenexpr** to the end. In the case of anything but **nil** is considered the true condition and it proceeds with the entity modification.

In this example the item in this list **lstSelection** that we are interested in is the **entity name** and to access it we need to use the lisp expression **car** inside the **`(setq entSelection (car lstSelection)… )`** expression. The **car** expression refers to the **A** register or simply the first item in a list or pair. (In this routine there is also the expression **cdr** which refers to the **D** register or simply everything else except the first item in a list or the second item in a pair.)

Once the routine has the entity name it needs to convert the entity name into its corresponding **entity list**. Like I mentioned above the entity list is a list of pairs or sub-lists that describe the entity. The **entget** expression has the format **`(entget ename [applist]).`** In this example the variable **lstEntity** is set to be the entity list of the variable **entSelection** in the setq expression: **`(setq … … lstEntity (entget entSelection) … …)`**

Entity lists look similar to this example:

```
((-1 . <Entity name: 7efa4dc8>) (0 . "CIRCLE") (330 . <Entity name: 7efa0cf8>) (5 . "1BC9")
(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "TEST") (100 . "AcDbCircle") (10 460.75
320.5 0.0) (40 . 64.9358) (210 0.0 0.0 1.0))
```

As you can see there are **dotted pairs** of items and **sub-lists** inside an **entity list**. In this case it is the entity list of a **circle**. The first item in each pair or list is what is called a **dxf code**. Each **dxf code** represents a specific property of the entity. Like the -1 code is the entity name, the 0 dxf code is the entity type, the 5 dxf code is a unique permanent string handle of the entity, the 410 dxf code is the layout name the entity resides, the **8 dxf code is the layer name**, the 10 dxf code is the center point of the circle, the 40 dxf code is the radius of the circle, and the 210 dxf code is the normal of the entity. The other dxf codes refer to the owners and object inheritance that is above the scope of this class. Dotted pairs are similar to lists but only have 2 items.

The purpose of the above routine is to change the layer of the selected item to be layer "0". To do this using AutoLISP the routine need to substitute a different dotted pair into the objects entity list and then modify the entity.

To change the layer the routine needs to use the substitute expression **subst**, along with a few more new expressions. In the above routine we used the expression in the format

```
(setq … … … … lstEntity    (subst (cons 8 "0")(assoc 8 lstEntity) lstEntity))
```

The expression cons creates a dotted pair. So **`(cons 8 "0")`** returns **`'(8 . "0")`** . The **assoc** expression will return a corresponding dotted pair from list. In this case we need the layer or 8 dxf code pair. The subst expression has the format **`(subst newitem olditem lst)`** So we subst the **`'(8 . "0")`** pair for the example

**`'(8 . "TEST")`** in the entity list, and then the **entmod** expression as shown above as **`(entmod lstEntity)`** modifies the object on the screen and the routine ends!

To load and use this routine you will need to place this routine inside an ascii text file and save it as layerzero.lsp. Be sure to save it into a directory that is inside your file search path. To locate your the search path go to options command and under the file tab and under the Support Files Search Path selection. (My suggestion is to create a **new directory** on your machine that is called **C:\LISP** and then add it to your search path. Place all of your lisp routines there.) From the command line inside AutoCAD® you can type (load "LayerZero") to load the above routine and (load "anylispfilename") to load any other routines you have.

Now that was old fashioned AutoLISP programming, but with the release of AutoCAD® 2000 came Visual LISP® to the rescue! Visual LISP® gave the lisp programmer over 1000 new functions to manipulate entities that had only previously been available to VB(A) programmers.

Before we jump into Visual LISP® it would be good to take a minute to explain what a **vla-object** is. A **vla-object** is the implementation of a concept called encapsulation. Encapsulation is the packing together of all of the properties and methods (actions) for an entity (hereafter referred to as a **vla-object** or just **object**). An example of a object property is its **layer**, and an example of a method is **copy.** To examine an objects properties and methods we need to create a new routine **VDOT.lsp**.

**Example 9: VDOT.lsp**

```
(defun C:VDOT (/ lstSelection ; list of selected item and selection point
                 entSelection ; Entity name of selection
                 objSelection ; VisualLISP ActiveX object (vla-object) of Selection
               )
 (vl-load-com)
 (if (setq lstSelection (entsel "\nSelect item on screen: "))
  (progn
   (setq entSelection (car lstSelection)
         objSelection (vlax-ename->vla-object entSelection)
   )
   (vlax-dump-object objSelection 'T)
   (textscr)
  )
 )
)
```

The new routine name is **C:VDOT** and a I have added a new local variable called **objSelection**. The **obj** prefix designates a **vla-object.** For individuals who are running pre-AutoCAD 2004 the expression **(vl-load-com)** need to be added to the routine. (This expression needs to be run at least once during a drawing editing session. It loads the Visual LISP® functions into memory.) You might find this routine looks very similar to the previous example, and it is. This new routine converts the entity name into a **vla-object,** (instead of an entity list), using the **(setq objSelection (vlax-ename->vla-object entSelection))** expression. The **VLAX** prefix means **Visual LISP Active X.** It next calls the **(vlax-dump-object objSelection 'T)** to dump (display) to properties on the text screen and (with the 'T) also the methods available for the object. The **(textscr)** expression switches the focus of AutoCAD to the text screen.

I tested this routine on the same circle I used for the entity list above and it returns this.

```
Select object to examine methods and properties:
Select Object:
; IAcadCircle: AutoCAD Circle Interface
; Property values:
;   Application (RO) = #<VLA-OBJECT IAcadApplication 00af9594>
;   Area = 13247.0
;   Center = (460.75 320.5 0.0)
;   Circumference = 408.004
;   Diameter = 129.872
;   Document (RO) = #<VLA-OBJECT IAcadDocument 0100af90>
;   Handle (RO) = "1BC9"
;   HasExtensionDictionary (RO) = 0
;   Hyperlinks (RO) = #<VLA-OBJECT IAcadHyperlinks 03963364>
;   Layer = "0"
;   Linetype = "ByLayer"
;   LinetypeScale = 1.0
;   Lineweight = -1
;   Normal = (0.0 0.0 1.0)
;   ObjectID (RO) = 2130333128
;   ObjectName (RO) = "AcDbCircle"
;   OwnerID (RO) = 2130316536
;   PlotStyleName = "ByLayer"
;   Radius = 64.9358
;   Thickness = 0.0
;   TrueColor = #<VLA-OBJECT IAcadAcCmColor 03966e70>
;   Visible = -1
; Methods supported:
;   ArrayPolar (3)
;   ArrayRectangular (6)
;   Copy ()
;   Delete ()
;   GetBoundingBox (2)
;   GetExtensionDictionary ()
;   GetXData (3)
;   Highlight (1)
;   IntersectWith (2)
;   Mirror (2)
;   Mirror3D (3)
;   Move (2)
;   Offset (1)
;   Rotate (2)
;   Rotate3D (3)
;   ScaleEntity (2)
;   SetXData (2)
;   TransformBy (1)
;   Update ()
```

The list above shows all of the available properties and methods of the selected circle object. In this lecture I am going to focus on the list of properties and invite you to return to my second lecture of "**The AUGI LISP guild Greatest Hits for Power LISPers"**, to learn more about vla-object methods. The next example will demonstrate to you how to change the layer property of a **vla-object** to be layer "0".

### Example 8: vl-Layerzero.lsp

```
; This routine will change a selection to layer "0" using ActiveX Property manipulation
(defun C:vl-LayerZero (/ lstSelection ; list of selected item and selection point
                         entSelection ; Entity name of selection
                         objSelection ; VisualLISP ActiveX object (vla-object) of Selection
                       )
 (vl-load-com)
 (if (setq lstSelection (entsel "\nSelect item on screen: "))
  (progn
   (setq entSelection (car lstSelection)
         objSelection (vlax-ename->vla-object entSelection)
   )
   (vla-put-layer objSelection "0")
 ; (vlax-put-property objSelection "layer" "0") ; Alternate
 ; (vlax-put objSelection "layer" "0")          ; Alternate
  )
 )
)
```

The format of the above routine should start to look very familiar to you by now. You will find that the format of many lisp routine follow the above format only with slight variations. In this example the new routine name is **C:vl-LayerZero.** The (vlax-dump-object…) expression has been removed and add this expression **(vla-put-layer objSelection "0")**. This expression (as if you can't guess) changes the layer of the object to be layer "0" directly. I have included two other ways of **doing the exact same thing** as alternates. You may find that in certain instances one or more of these will not work, and you need to test all three to find the one that does. I have commented them out with the semicolon prefix on them.

Thats it! That is the power of Visual LISP®. You need to remember that some properties are in all objects, and some are only in a few. Like the radius property is only in a circle, and not in a piece of text. These examples up to now have only given the use the ability to change one entity at a time. In order to change multiple entities at one time you need to create selection sets. The next example will change a selection set of objects to layer "0"

### Example 7: SS-Layerzero.lsp

```
; This routine will change a selection set to layer "0" using ActiveX Property manipulation
(defun C:SS-LayerZero (/ entSelection ; Entity name of selection
                         intCount     ; Iteration Counter for Selections Set
                         objSelection ; VisualLISP ActiveX object (vla-object) of Selection
                         ssSelections ; Selection Set of Entities
                       )
 (vl-load-com)
 (princ "\nCreate Selection set to be changed to layer 0: ")
 (if (setq ssSelections (ssget))
  (repeat (setq intCount (sslength ssSelections))
   (setq intCount     (1- intCount)
         entSelection (ssname ssSelections intCount)
         objSelection (vlax-ename->vla-object entSelection)
   )
   (vla-put-layer objSelection "0")
  )
 )
)
```

This new routine has a new name and a couple new variables, **intCount** and **ssSelections** I have introduced a new expression **(princ …)** that creates a prompt on the command line for the user to select a selection set. The variable **ssSelections** stores the selection set returned by the **(ssget)** expression. In order to change each member of the selection set we need to repeat the property modification once for each member of the selection set. To do this it uses the integer variable **intCount** to point at each item in the selection set. It first sets the integer variable **intCount** to be the length of the selection set using the **(sslength ssSelections)** expression. Next because selection sets are 0 indexed (means the first item in the list is the 0th item) you need to minus one from the index, using the **(setq intCount (1- intCount))** expression to minus one from **intCount**.

Then using the **(ssname ssSelections intCount)** expression we get the entity name of the indexed item from the selection set. It converts the entity name into a **vla-object** using the **(vlax-ename->vla-object entSelection)** just like before and then put the layer property to be layer "0". We continue repeating inside the repeat expression for every item in the selection set, incrementing the intCount index number down one for each repetition and setting each item to be layer "0".

As I had mentioned above some properties are exclusive to specific objects, like the radius of a circle. If you tried to change the radius property of a piece of text you would get an error and your program would stop. In order to prevent this, we can use a **selection set filter** to only allow the user to select items that have the right properties that we want to change.

**Example 6: CHRadius .lsp**

```
; This routine will change the radius property of a selection set of circles
(defun C:CHRadius (/ intCount     ; Iteration Counter for Selections Set
                     objSelection ; VisualLISP ActiveX object (vla-object) of Selection
                     sngRadius    ; Single real number for New Radius length
                     ssSelections ; Selection Set of Entities
                  )
 (vl-load-com)
 (princ "\nCreate Selection set to be changed to layer 0: ")
 (setq ssSelections (ssget (list (cons 0 "CIRCLE"))) ; <- Selection Set Filter
       sngRadius    (getdist "\nEnter new radius: ") ; <- User input get distance
 )
 (if (and ssSelections
          (> sngRadius 0.0)
     )
  (repeat (setq intCount (sslength ssSelections))
   (setq intCount     (1- intCount)
         objSelection (vlax-ename->vla-object
                       (ssname ssSelections intCount)
                      )
   )
   (vla-put-radius objSelection sngRadius); <- Change Radius Property
  )
 )
)
```

This new routine has the name **CHRadius** because it will change the radius of selected objects (circles). The single real number **sngRadius** variable is added to the local variables. I removed the **entSelection** variable from the local variables as I will explain later. The next change is I have added a partial entity list to the ssget expression. Like we

saw in the first example **entity lists are lists of dotted pairs, and sub-lists**. In this example I use the **list** expression and the **cons** expression to create a partial entity list **(list (cons 0 "CIRCLE"))** returns **'((0 . "CIRCLE"))**. The cons expression returns a dotted pair and list expression returns a list. This partial entity list tells the ssget expression to only select entities that this dotted pair in their entity lists or select only circles. The next line I use one of the user entry expressions **getdist**. The **getdist** will prompt the user for a length and the user can respond in one of three methods. The first is a real number, the second is in feet and inches, or the third is selecting two points on the screen. In AutoLISP there is also a **getreal** expression that only takes real numbers, and there are several other user entry expressions including **getint**, **getstring**, **getkword**, **getpoint**, and **getangle**.

I also have added an **if** statement to check to see if the user has selected anything and if the **sndRadius** variable is greater than 0.0, before proceeding with the changes. In the section of the routine where I convert the entity name to vla-object, rather than store the **entSelection** variable and then convert it, I nested the **ssname** expression inside the **vlax-ename->vla-object** expression. I really didn't need to save the entity name to a variable because I only use it once. The last modification to the routine was changing the property in the vla-put-**radius** expression.

This CHRadius.lsp routine is a template that we can now use to create many new useful functions. My next function is ScaleRadius.lsp.

**Example 5: ScaleRadius .lsp**

```
; This routine will scale the radius properties of a selection set of circles
(defun C:ScaleRadius (/ intCount       ; Iteration Counter for Selections Set
                        objSelection   ; vla-object of Selection
                        sngScaleFactor ; Scale factor for radius property
                        ssSelections   ; Selection Set of Entities
                      )
 (vl-load-com)
 (princ "\nCreate Selection set to be changed to layer 0: ")
 (setq ssSelections   (ssget (list (cons 0 "CIRCLE")))   ; <- Selection Set Filter
       sngScaleFactor (getdist "\nEnter scale factor: ") ; <- User input
 )
 (if (and ssSelections
          (> sngScaleFactor 0.0)
     )
  (repeat (setq intCount (sslength ssSelections))
   (setq intCount     (1- intCount)
         objSelection (vlax-ename->vla-object
                       (ssname ssSelections intCount)
                      )
   )
   (vla-put-radius objSelection (* (vla-get-radius objSelection) sngScaleFactor))
  )
 )
)
```

This routine is again very similar to the previous example. I have changed the function name to **ScaleRadius**, and substituted the **sngScaleFactor** for the **sngRadius** variables. I modified the **getdist** expression to prompt for a scale factor instead of a radius, and modified the if statement to check to see if the sngScaleFactor is greater than 0.0. Now I have introduced a new expression in **vla-get-radius** that returns (or gets) the radius property of the circle object. So the **(vla-get-radius objSelection)** returns the radius of the circle and multiplies the value times the scale factor **sngScaleFactor** and use the **vla-put-radius** expression change the radius property of the circle object to the new value.

This first routine will change a selection set of text to be upper case letters, and the next example will change the selected text lowercase letters using the **strcase** expression. The string case **strcase** expression has the form **(strcase string [which])** and it will change a text string to uppercase letters, or if the **[which]** argument is added and is **T** will change the letters to be lowercase letters.

### Example 4: UpperCase .lsp

```
; This routine will change selected to be uppercase letters.
(defun C:UpperCase (/ intCount       ; Iteration Counter for Selections Set
                      objSelection   ; vla-object of Selection
                      ssSelections   ; Selection Set of Entities
                   )
 (vl-load-com)
 (princ "\nSelect text to be uppercase: ")
 (setq ssSelections   (ssget (list (cons 0 "TEXT,MTEXT"))))   ; <- Selection Set Filter
 (if ssSelections

  (repeat (setq intCount (sslength ssSelections))
   (setq intCount     (1- intCount)
         objSelection (vlax-ename->vla-object
                       (ssname ssSelections intCount)
                      )
   )
   (vla-put-textstring objSelection (strcase (vla-get-textstring objSelection)))
  )
 )
)
```

### Example 3: LowerCase .lsp

```
; This routine will change selected to be uppercase letters.
(defun C:LowerCase (/ intCount       ; Iteration Counter for Selections Set
                      objSelection   ; vla-object of Selection
                      ssSelections   ; Selection Set of Entities
                   )
 (vl-load-com)
 (princ "\nSelect text to be lowercase: ")
 (setq ssSelections   (ssget (list (cons 0 "TEXT,MTEXT"))))   ; <- Selection Set Filter
 (if ssSelections

  (repeat (setq intCount (sslength ssSelections))
   (setq intCount     (1- intCount)
         objSelection (vlax-ename->vla-object
                       (ssname ssSelections intCount)
                      )
   )
   (vla-put-textstring objSelection (strcase (vla-get-textstring objSelection) 'T))
  )
 )
)
```

### Example 2: Prefix .lsp

```
; This routine will add a string prefix to selected text.
(defun C:Prefix (/ intCount       ; Iteration Counter for Selections Set
                   objSelection   ; vla-object of Selection
                   ssSelections   ; Selection Set of Entities
                   strPrefix      ; Prefix text string
                )
 (vl-load-com)
 (princ "\nSelect text to be modified: ")
 (setq ssSelections   (ssget (list (cons 0 "TEXT,MTEXT")))   ; <- Selection Set Filter
       strPrefix      (getstring "\nEnter string Prefix: ")  ; <- String entry
 )
 (if ssSelections
  (repeat (setq intCount (sslength ssSelections))
   (setq intCount     (1- intCount)
         objSelection (vlax-ename->vla-object
                       (ssname ssSelections intCount)
                      )
   )
   (vla-put-textstring objSelection (strcat strPrefix (vla-get-textstring objSelection)))
  )
 )
)
```

In this routine the name was changed to **Prefix** and a new variable was added **strPrefix** (str is the variable prefix for string). The routine utilizes the string concatenation expression **strcat** to join two strings together. So it gets the text property from each text object and adds the prefix onto it and then puts it back into the textstring property.

At this time I would like to continue by taking some suggestions from the audience. What kind of routine would you like to see created. Can you give me several object types and properties to change and we will develop the routines for them using this template.

### Example 1: ???????????

I really hope you all learned a little and motivated you to learn more about programming Visual LISP. For those interested in participating in the AUGI LISP forum go to www.augi.com and go to the forums tab. If you do not have direct web access you can join me at the www.waun.org website and join in an email based discussion group.

Please take the time to fill out the evaluation and **Thank You** for attending!