



Las Vegas, Nevada

November 28 – December 1

Speaker Name: **Randy Kintzley**

Course: **PG33-1**

Course Description: **AutoCAD Hacker's Handbook**

AutoCAD Hacker's Handbook

Step into the mind of an Express Tools programmer and learn his favorite tricks for programming with Lisp, VB and VBA. Some of the topics this course will cover are tips for debugging, entity creation, finding information, techniques for accessing VB functionality from lisp, as well as tips on software design.

How to display a dialog version of a command from lisp

If you type **LAYER** AutoCAD will display the layer dialog. On the other hand; if you type **(command "layer")** AutoCAD will invoke the command line version of layer. But... What if you *want* the layer dialog to display from a lisp routine?

Here's how:

```
(initdia)  
(command "layer")
```

The `initdia` function causes the next AutoCAD command to display the dialog version of the command. Unfortunately it does not work with all commands. XREF is one command that does NOT work with `initdia`. See the section on `vla-sendcommand` for a way to cheat and get the dialog to appear...

Tips for Entity creation with Lisp:

- Look at the entity list for existing entities, remove the unwanted information and cut and paste
- Make sure referenced symbol tables exist (with exception of layers)
- Watch out for CECOLOR, CELTYPE...etc. For example: If you do not explicitly include a color then the current color will be used. To specify, color ByLayer, use color 256.

Some debugging tips:

- Putting in break points with `getstring` and printing results with `print`
- Try it at the command line to see how it works
- Comment closing parens to match opening ones
- Use local variables when you can
- Remember to initialize counters and other variables

Finding information:

- Using the LSP command included with the Express Tools
- DXF files
- Using the utilities in rk-util.lsp:
 - LL (Lisp-List)
 - DXFHELP
 - OHELP (object help)
 - EDIFF (entity-diff)

Recommended programming practices:

- Comment closing parentheses to match opening ones.
- Divide and conquer. Divide the program up into separate tasks and write separate functions to handle each of them. Remember; it's not a contest to see who can nest the most parentheses.
- Use local variables
 - Pass in needed values
 - Return results. If more than one value return a list.
- For functions that are likely to change; pass in a single list and extract the arguments on the fly. NOTE: See "make-circle.lsp" for an example.
- Avoid modifying acad2000.lsp or acad2000doc.lsp. Instead create an acad.lsp or acad.doc.lsp.
- Menu changes:
 - When possible, use partial menus instead of modifying the main acad.mnu.
 - If you do modify the main menu then put in easy to see comments to let you know where the changes are so that you can easily move the changes to the next AutoCAD menu when the next release comes out.
- Look at other people's code. Surf the web, read CAD mags ...etc.

Some Menu tips

- Buttons – AUX3 and AUX4... under used powerhouses.
- Control (layer, color, dim, view, ...etc.)
- Context menu tricks with "Object_name" (also show c:newtext)
- Diesel (Tom)

Error handling with the Express Tools error handler:

The Express Tools error handler is a good general purpose error handler that you can use in your own lisp routines. Some of the benefits are, UNDO handling, SYSVAR management, and extensibility.

- Initialize using `(acet-error-init (list ...))`
- Always restore the original error handler `(acet-error-restore)`
- UNDO flags
 - no undo - nil
 - undo begin and end - 0
 - undo on cancel - 1
- Sysvars
 - set and store original value for restore later
 - pass nil to avoid changing a value but still save the value

For example, I will need to turn highlight off later but not yet, I want to wait until after I ask them to select some objects.

Example:

```
;; highlight value does not change but its value
;; is saved so it can be restored later
(acet-error-init (list 1 '("highlight" nil)))
(setq ss (ssget))
(setvar "highlight" 0) ;; now turn highlight off
;; do some other stuff here....
(acet-error-restore) ;; restore original error and
highlight value
```

- Common sysvar gotchas:
 - OSMODE - turn osnaps off
 - OSNAPCOORD - controls how coordinate input in scripts and lisp behave with osnap settings. The default is to use explicitly entered coords unless lisp or script is active. In other words, AutoCAD will, by default, honor osnap settings. You can turn OSNAPCOORD to 1 to always use explicit entry within lisp and scripts.
 - PICKSTYLE - controls group selection. You can set it to 0 to pick just one object within a group and not get the entire group.
 - UCSFOLLOW - turn it off if plan to mess with the UCS.


```

    );setq then paper space
    (setq flt (list '(67 . 0))) ; else model space
);if
flt
);defun ssget-current-space-filter

```

Here's are some examples of how you would typically use it:

```

(setq flt (ssget-current-space-filter))
;; restrict selection to any object in the current space
(ssget flt)

```

```

;; restrict selection to LINES in current space
(ssget (append '((0 . "LINE")) flt))

```

```

;; only allow objects in the current space that are not on locked
layers
(ssget ":l" flt)

```

Mistake 3 - Assuming a certain type of behavior based on your sysvar settings.

There is no one cure-all for this one. Always research the variable settings that affect a command or groups of commands that you plan to use. Then make sure you set them the way you want them and put them back when you are done.

Mistake 4 - Expecting one object type and getting another.

Don't just ask your user to select the proper type of object. Filter for that type of object automatically. For example: If you are expecting TEXT then use `ssget` to mandate that the user only select text. i.e. `(ssget '((0 . "TEXT")))`

Lisp performance tips:

- o cmdecho=0, highlight=0
- o Do as little as possible within loops. Prepare needed values in front of the loop so fewer operations need to be done within the loop.

- o Entget tends to be slow. Hang on to the data as long as you need it instead of doing multiple entgets.
- o When possible, use `cons` instead of `append` when building lists. It's much faster.
- o Use `'=` instead of `'equal` for numeric values.
- o Structure if statements so that least costly conditional is executed first.
- o For the example below, lets say `'doit` is a simple T/nil flag and the `find-the-perfect-ent` function is somewhat costly in terms of time. The following if statement will not call

`find-the perfect-ent` unless the `doit` flag is non-nil.

```
(if (and doit
      (setq na (find-the-perfect-ent))
      (setq ent (entget na))
    )
    (princ "\ndoing it")
  );if
```

- o Avoid calling `(gc)` within a loop.
- o Improve load times: compile lsp files into fas files.

Example:

```
vslide
(vlisp-compile 'st "mystuff.lsp" "mystuff.fas")
```

Tips for optimizing list processing

Avoid this:

```
(setq ent [...some entity list])
(setq lst [...some list of real values])
(setq n 0)
(repeat (length lst)
  (setq x (nth n lst)
        h (cdr (assoc 40 ent))
        lst2 (append lst2 (list (* h x))))
  )
(setq n (+ n 1))
)
```

What's wrong with the above code?

- Can be done in fewer statements
- Repeatedly sets 'h' to the same value. Should be done before entering the loop.
- Uses append instead of cons

Something more like this is better:

```
(setq h (cdr (assoc 40 ent)))      ;; set value ahead of time
(foreach x lst                    ;; use foreach to avoid need for a
counter variable
  (setq lst2 (cons (* h x) lst2));; use cons instead of append
)
```

Or even better still:

```
(setq h (cdr (assoc 40 ent)))      ;; set value ahead of
time
(setq lst2 (mapcar '(lambda (x) (* x h)) lst)) ;; use mapcar
```

Error trapping with vl-catch-all-apply ...How to catch a lisp error and continue processing.

There are three functions that can be used together to give your lisp routines the ability to catch errors and continue processing.

- vl-catch-all-apply
- vl-catch-all-error-p
- vl-catch-all-error-message

The following example demonstrates the use of these functions. The example function asks you to enter Color or Linetype and then takes you to a sub-prompt based on that answer. Using traditional methods; If the user cancels at a sub-prompt, lisp will cancel the entire lisp command. This example demonstrates how to catch the cancel and continue processing. The 'LType' option uses traditional methods so if you cancel at that sub-prompt the entire command will be canceled.

If you enter the 'Color' option and then cancel at that sub-prompt, you will be taken back to the main prompt. This behavior is achieved through the use of vl-catch-all-apply

```
(defun c:catchit ( / c lt )
  (while (progn
    (initget "Ltype Color")
    (setq ans (getkeyword "\nEnter [Color/Ltype]: "))
  ))
)
```

```

(if (equal ans "Color")
  (progn
    ;; call getstring using 'vl-catch-all-apply'
    (setq result (vl-catch-all-apply 'getstring '("\nEnter a
color: ")))
    (if (not (vl-catch-all-error-p result))
      (setq c result)
      (progn
        ;; print the error information
        (princ (strcat "\nTrapped error: "))
        (princ (vl-catch-all-error-message result))
        (princ "\nContinuing processing...")
      );progn else
    )
  );progn then
(progn
  ;; call getstring the standard way...
  ;; if the user cancels this, then it's all over.
  (setq lt (getstring "\nEnter a lintype: "))
);progn else
);if
);while

(princ "\nYou entered Color=")(princ c)
(princ "\nYou entered Ltype=")(princ lt)
(princ)
)

```

How to call VBA/ActiveX functions from lisp

Ever want to change a setting in AutoCAD that is only changeable from the options dialog box?

For example, set the current profile or turn the screen menu on from lisp. There's no system variable for displaying the screen menu and the current profile can be viewed as a sysvar but it's read-only so you can't set it. However, from VBA these things can be done quite easily. The

screen menu is controlled by a property on the preferences display object. The current profile is controlled by a profiles object that can also be accessed through the preferences object.

The good news is that you can use the same methods and properties that are accessible in VBA from LISP as well.

Here are the basics:

- Initialize COM support in lisp using `(vla-load-com)`. This only needs to be done one time per AutoCAD session.
- Get the AutoCAD application object: `(setq app (vlax-get-acad-object))`.
Once you have the AutoCAD application object you can access it's properties and get other sub-objects, such as a collection of open drawings, the preferences object or just about anything you can access from VBA.

Once you have an object; there are three basics to manipulating it using lisp.

1. Getting the value of a property.

```
(vla-get-propertyName <object>)
```

or the more general version `(vlax-get <object> "propertyname")`

2. Setting a property value.

```
(vla-put-propertyname <object>)
```

or the more general version `(vlax-put <object> "propertyname")`

3. Invoking methods or functions on an object.

```
(vla-methodname <object> [arg1] [arg2] [arg3] ...)
```

or the more general version:

```
(vlax-invoke <object> "methodname" [arg1] [arg2] [arg3] ...)
```

Tips for getting VBA/ActiveX API information.

- It is important to learn how to translate documentation in VBA form into lisp syntax.

For example; To get the preferences object in VBA the syntax is documented like this:

```
Application.Preferences
```

So in VBA you might write something like:

```
Dim pref as AcadPreferences  
Set pref = Application.Preferences
```

In lisp you would write something like this...

```
(setq app (vlax-get-acad-object)) ;; get the Application object  
(setq pref (vla-get-preferences app))
```

Note the `vla-get-propertyname` syntax.

- o Another great resource for finding VBA information is 'vlax-dump-object' This one is one of my favorite functions when I'm working with ActiveX from lisp. (vlax-dump-object <obj>) will display a list of the properties for an object.

The following is an example of how you can change a setting that is not associated with sysvar. The following code will toggle the screemenu on or off

```
(defun scrmenu ( on / app pref disp )
  (vl-load-com)
  (setq app (vlax-get-acad-object)
        pref (vla-get-preferences app)
        disp (vla-get-display pref)
  )
  (if on
    (setq on :vlax-true)
    (setq on :vlax-false)
  )
  (vla-put-DisplayScreenMenu disp on)
)
```

See also: profile.lsp for an example of setting the current profile.

How to call functions in an activex dll.

You can load ActiveX dll files into the AutoCAD memory space and call them from VBA or from lisp. The high level steps involved are as follows:

1. Get or make yourself an ActiveX dll. VB5 and VB6 are great for creating ActiveX dlls.
2. Write the needed lisp to load the dll and set a variable to the dll object.
3. Use that object as a parameter to special lisp functions that allow you to access methods and properties within the dll.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

This sample shows how you can call a function within an ActiveX dll.

```
;; This sample dll contains functions that allow you to maximize or  
minimize a window.
```

```
;; This sample maximizes the AutoCAD window.
```

```
;;
```

```
(defun c:maximize ( / app cls cap )
```

```
;; Initialize the COM engine. This must be done before calling
```

```
;; any of the ActiveX functions.
```

```
(vl-load-com)
```

```
;; Get the AutoCAD application object
```

```
(setq app (vlax-get-acad-object))
```

```
;; Get the dll class object
```

```
(setq cls (vla-GetInterfaceObject app "windowstuff.class1"))
```

```
;; get the AutoCAD caption
```

```
(setq cap (vla-get-caption app))
```

```
(vlax-invoke cls "maximizeIt" cap)
```

```
;; clean up
```

```
(vlax-release-object cls)
```

```
(setq cls nil)
```

```
(gc)
```

```
(princ)
```

```
);defun c:maximize
```

Using `vla-sendcommand`

The `vla-sendcommand` method allows you to send a string to the command line. It pretty much behaves as though you cut and pasted a string to the AutoCAD command line.

For example: If you wanted to force the xref dialog to display from within a lisp routine you could use `vla-sendcommand` to do it.

```
(defun c:lspxref ( / app doc )
  (setq app (vlax-get-acad-object)
        doc (vla-get-activedocument app)
  )
  (vla-sendcommand doc "xref ")
  (princ)
)
```

Use VBA from lisp to open, close or start new drawings. (VBASTMT)

The VBASTMT command allows you to execute a VBA statement from the command line. You can call this command within lisp using the (`command ...`) function. One example of where this function comes in handy is opening or closing drawings from lisp while in multiple document mode (SDI=0).

The following example opens a specified drawing. Note the section of code for dealing with SDI=0.

NOTE: For other examples see "lspopen.lsp".

```
;;;;;;;;;;;;;;
(defun lspOpen ( dwgname / )
  (if (and dwgname
          (findfile dwgname)
      )
      (progn
        (if (= 1 (getvar "sdi"))
            (progn
              (if (= 0 (getvar "dbmod"))
                  (command "_open" dwgname)
                  (command "_open" "_y" dwgname)
                )
            )
        )
      )
  )
)
```

```
)
);progn
(progn
  (command "_vbastmt"
    (strcat "Application.Documents.Open(\" \" dwgname \" \")"
    )
  )
);progn else
);if
);progn then
(princ "\nlsopen error- File not found")
);if
(princ)
)
```

Methods of storing data so that it can be accessed across drawings and/or across AutoCAD sessions.

Sometimes you need a little more flexibility than just a regular lisp variable that can be accessed from one drawing only. The following are various ways of storing data so that it can be accessed from one drawing to the next or even across AutoCAD sessions.

SDI=1, LISPINIT=0 - When in single document mode you can take advantage of the ability to turn LISPINIT off. This means that lisp will not re-initialize each time you start a new drawing. So variables you set in the previous drawing will still be set when you open another drawing or start a new drawing. Your lisp files will remain loaded and variables will retain their values when you start a new drawing. This is only for a single session of AutoCAD.

(vl-bb-set ...) and **(vl-bb-ref ...)** - The BB stands for Bulletin-Board. This pair of functions allows you to store data on an application wide basis. In other words, values set using vl-bb-set, are shared for all open drawings. These functions work both in single and multiple document modes. The data is only valid for the life of a single session of AutoCAD.

(setcfg ...) and **(getcfg ...)** - This pair of functions allows you to store data beyond the life of an AutoCAD session or instance. In other words, you can store a value using setcfg, shut down AutoCAD, re-start it, and retrieve that value using getcfg.

(acet-setvar (list ...)) and **(acet-getvar (list ...))** - If you have the Express Tools installed, this pair of functions are a great resource for storing and retrieving data in various places. These functions work particularly well for storing custom command settings and information across editing sessions. You can store values with the current drawing, the current profile, or in a general section of the registry.

The syntax for the acet-setvar is as follows:

```
(acet-Setvar (list <varname> <value> [location]))
```

Where:

varname is a string representation of the variable name

value is a real, int, string, list, or entity name.

location is an optional integer bit coded value. Location is the sum of one or more of the following values:

1 = Store the value in the DWG

2 = Store the value with the current AutoCAD profile

4 = Store in the fixed profile section of the registry

NOTES:

- If the `location` value is not provided then 2 is used as a default, resulting in the value being stored with the current profile.
- List data and entity name data can only be stored in the `dwg`. Do not attempt to store list data or an entity name using the current profile or fixed profile location flags.
- Some lists with nested dotted pairs cannot be stored using `acet-setvar`.

The syntax for `acet-getvar` is as follows:

```
(acet-Getvar (list <varname> [location]))
```

Where `varname` is the string name of the variable and `location` is the sum of one or more of the bit coded values described above. If the optional location argument is omitted, the drawing will be searched first, and then the current profile and then finally the fixed profile. The first location that has the specified variable stored there will be used for the return value.

Recommended books

- AutoCAD VBA (Bill Krammer & John Gibb)
- VB guide to WIN API by Dan Appleman