Autodesk University 2007

AutoLISP® for CAD Managers

Robert Green – Robert Green Consulting Group

CM305-1L

AutoLISP/Visual LISP is a powerful way to extend the AutoCAD functionality, but many avoid using it because they think it's too hard to learn. This class lays out the basics of AutoLISP using a building-block approach with practical examples you can use every day to tailor AutoCAD to your specific needs. We'll cover using ACADDOS.LSP to control system startup, how to load files from a network location, basic syntax, accessing the command line, creating your own user command functions, and undefining/redefining commands -- all in a hands-on lab environment. This class assumes you have intermediate AutoCAD skills and the ability to edit/save files using a text editor such as Notepad. No prior AutoLISP experience is necessary.

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 *Cadalyst* 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

Autodesk®


Autodesk
University
2007

My Philosophy Regarding AutoLISP

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.

Of particular importance to me, given my CAD manager emphasis, is the ability to really standardize the way AutoCAD behaves, load external commands and centralize all of this from a standardized network location.

I realize there is a wide range of AutoLISP skills in this class so I've tried to build our examples starting with more simple concepts and ratchet the complexity of the code up as we move along.

For This Lab

Since we can't completely shred the setup of these lab machines we've installed a folder onto your machine that contains all the files you'll need for this lab. I'll supply you with the name of this folder just prior to starting the lab. Go ahead and write the folder name here:

Folder name: _____

The lab will involve placing an ACADDOC.LSP file into your AutoCAD SUPPORT directory. Please remember to delete this file when you exit so we don't mess up the lab machines. I thank you and so will the people who use the lab later.

Key Files and Variables

The first thing to understand is that AutoLISP has a couple of key files and a key function that perform startup operations for you. The key files are called ACAD.LSP and ACADDOC.LSP and the key function is called S::STARTUP and their operations are as summarized here:

ACADDOC.LSP This file loads every time a new drawing session is started in AutoCAD 2xxx based products. Therefore any programming you place in this file will be loaded automatically every time a drawing is opened or started no matter how the ACADLSPASDOC variable is set. Like the ACAD.LSP file, ACADDOC.LSP is normally located in the SUPPORT subdirectory of the AutoCAD installation.

MENUNAME.MNL This file loads whenever its menu counterpart (either an MNU, MNC or CUI 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.

Key Functions

The first thing to understand is that AutoLISP has a couple of key files and a key function that perform startup operations for you. The key files are called ACAD.LSP and ACADDOC.LSP and the key function is called S::STARTUP and their operations are as summarized here:

S::STARTUP function This function is typically defined in the ACADDOC.LSP file (or ACAD.LSP file for AutoCAD R14 installations) and its sole job is to execute customized commands you need to initialize your new drawing environment. This function is the perfect place to set system variables like DIMSCALE, VIEWRES parameters, current layers, etc. The most powerful aspect of the S::STARTUP function is that it invokes automatically and it lets you control exactly how the AutoCAD environment is initialized.

Reactors These embedded "reactor" like functions were added when Visual Lisp was bound into AutoCAD 2000. Working with these functions allows HUGE advances in what AutoLISP can do with only minimal overhead. These functions must be invoked (typically from the ACAD.LSP file) by loading the **VL-LOAD-COM** reactors. We'll examine reactors in much more detail a bit later.

Basic ACADDOC.LSP

First let's set some key variables so that each drawing session will get the same basic setup. Just open a text editing program and enter in the following (note the use of ; characters for insertion of comments):

```
(command "viewers" "y" "5000")          ; sets view resolution for no jaggy circles
(command "-color" "BYLAYER")            ; set color to BYLAYER
(command "-linetype" "set" "BYLAYER" "") ; set color to BYLAYER
(command "menu" "menuname.mnc")         ; load standard menu file at startup
(prompt "\nACADDOC.LSP loaded ... ")    ; send a diagnostic prompt to the command line
```

Now save this file into your SUPPORT folder with the name ACADDOC.LSP and restart AutoCAD to see if it worked. If everything loaded fine you'll see the prompt appear in your command line window. If not the file was most likely not saved in the correct path.

Extra: Make a mistake in the file on purpose like this and see what happens:

```
(command "-colllor" "BYLAYER")          ; color is purposely misspelled
```

Conclusion: Obviously you could do much more but even these simple ideas allow you to gain a lot of control with minimal AutoLISP knowledge.

Network Support

What if we wanted to store our AutoLISP routines in a single network location so that EVERYBODY had the same AutoLISP files? This approach would allow us to only maintain a single repository of AutoLISP routines and would mean that keeping all machines in sync would only require a single maintenance activity on our part; updating only a single AutoLISP file on the network. I've used this network support approach with many clients and it has not only worked but it has allowed me to keep machines on wide area networks standardized without having to travel or run my feet to a nub in campus style environments.

Here's the basic idea of how to gain control:

- Seed the user's machine with an ACAD.LSP and/or ACADDOC.LSP file in their support directory that will point to an external network location using a variable.
- Execute all load instructions from the ACAD.LSP and/or ACADDOC.LSP using the variable location.
- Test for the presence of all files you'll load using the variable location prior to loading to assure that errors are avoided.

Example ACADDOC.LSP

In this case we wish to load some external AutoLISP files (called UTILS1.LSP and UTILS2.LSP) we've written to add commands to AutoCAD's vocabulary. The case where commands are being added suggests using ACAD.LSP because the commands need only be loaded once, when AutoCAD starts up. Further, we'd like to load these files from a network drive and path that I'll call X:\AUTOLISP as an example.

The contents of the ADADDOC.LSP would look like this:

```
(setq lisp_path "X:\\AUTOLISP\\")   ; sets the path

(if (findfile (strcat lisp_path "utils1.lsp"))
  (load (strcat lisp_path "utils1.lsp"))
)

(if (findfile (strcat lisp_path "utils2.lsp"))
  (load (strcat lisp_path "utils2.lsp"))
)
```

Notice that the LISP_PATH variable is set first because the following statements make use of the variable to locate the files UTILS1.LSP and UTILS2.LSP in the network drive. This approach gives us the benefits of loading files from a remote location, as opposed to putting all the code in the ACAD.LSP itself, thus we can maintain the remote files separately!

On the down side though, we still have to maintain the ACAD.LSP file on each machine as we include more AutoLISP files for loading. We need a way to get around this problem.

Example ACADDOC.LSP (con't)

In this case we load in a single remote file from the network location that loads all external functions for us. The new file I'll reference is called INIT.LSP because I use it to INITIALIZE all my file load statements.

The contents of the ADAD.LSP would look like this:

```
(setq lisp_path "X:\\AUTOLISP\\")   ; sets the path

(if (findfile (strcat lisp_path "init.lsp"))
  (load (strcat lisp_path "init.lsp"))
)
```

In the X:\AUTOLISP\ folder we'll now have a file called INIT.LSP that looks like this:

```
(if (findfile (strcat lisp_path "utils1.lsp"))
  (load (strcat lisp_path "utils1.lsp"))
)

(if (findfile (strcat lisp_path "utils2.lsp"))
  (load (strcat lisp_path "utils2.lsp"))
)
```

Now we have complete control because the only thing our ACADDOC.LSP file does it set a network location to "look to" and a single initializing file. Now I can make any changes I want in the INIT.LSP file and I'll never have to go to the local machine again unless I need to change the value of the LISP_PATH variable!

This is the way it should be, zero maintenance at the remote machine, total control from the network support path.

Bonus: Note that by setting a variable for the network location in memory that we DO NOT have to make any pathing adjustments in the AutoCAD profile! Again, zero maintenance at the local machine.

USER.LSP Architecture

I've found that many companies have power users who have a collection of their own favorite AutoLISP files. The problem becomes one of balancing the need to standardize the way AutoCAD operates without crippling the power users that can do their own AutoLISP work. The solution? Implement an automatic routine that allows users to create their own USER.LSP file that will load in all their AutoLISP code without stepping on the ACAD.LSP or ACADDOC.LSP files you'll use to standardize the installation.

A sample code segment for loading the USER.LSP looks like this:

```
(if (findfile "user.lsp")
  (load "user.lsp")
)
```

The only trick here is that the power user must name his/her file USER.LSP and the file must be located in a path that AutoCAD can find (typically the SUPPORT folder) on the local machine (remember the user is local).

USER.MNU/MNC Architecture

Want to load a user specified menu file? Simple! Use this code segment:

```
(if (findfile "user.mnc")
  (command "menu" "user.mnc")
)
```

Again, the only requirements are the file name and path as in the USER.LSP case.

Multi Keystroke Shortcuts

You've probably all dabbled with setting up keystroke shortcuts for AutoCAD commands either by editing the ACAD.PGP file directly or by using the Tools -> Customize menu in AutoCAD's interface. This customization methodology allows you to trigger a command using a single keystroke like this:

F triggers the FILLET command.

The problem with this approach is that the FILLET command will always use the last setting for the FILLETRAD variable. What if I want to trigger a zero radius fillet? Let's use an AutoLISP function like this:

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

Make it even better like this:

```
(defun c:fz ()
  (setq old_filletrad (getvar "filletrad"))     ; Store the old fillet radius value in a variable
  (setvar "filletrad" 0.0)
  (command "fillet" pause pause)
  (setvar "filletrad" old_filletrad)            ; Put the fillet radius back the way you found it
  (princ)
)
```

This example illustrates that simple AutoLISP routines can automate multikeystroke functions with ease!

ZOOM ALL

Here's another favorite of mine, the ZA function that peforms a zoom to extents and then zooms out by a factor of 0.95 to create a slight margin around the outside of engineering title blocks:

```
(defun c:za ()
  (command "zoom" "e" "zoom" "0.95x")
  (princ)
)
```

AUTO PURGE

Here's a function that does an autopurge and save by keying in ATP:

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

Note: If you're unsure how to build a function just go to AutoCAD and type in your commands taking careful notes of what you typed. Then just use the syntax I've given here and substitute your values in. Just watch the typos and you'll be writing cool functions very quickly.

Undefining Commands

You can undefine a command like this:

```
(command ".undefine" "QSAVE")
```

Redefining Commands

Now take it another step and redefine the command like this:

```
(command ".undefine" "QSAVE")
(defun C:QSAVE ()
  (command "-purge" "a"  "*" "n" ".qsave")
  (princ)
)
```

Use INIT.LSP to Gain Control

Now look at the INIT.LSP file I provided and use the various sections I've provided for undefining and redefining functions and have some fun! Nothing you're doing is permanent and if you mess something up by mistake you can always delete the INIT.LSP.

Challenge - Basic Reactor Concepts

Reactors are a very sophisticated (yet not too complex) way to get some great results from AutoLISP. The following code segment illustrates a problem I solved for a client who wanted their DIMSCALE, LTSCALE and TEXTSIZE parameters to switch as they moved between model space and paper space layouts. In order to set these parameters "on the fly" I utilized reactors to detect when layout tabs were switched and then employed some basic programming to set the variables accordingly based on storing the drawing's DIMSCALE in the USERR1 variable register. This may sound complicated right now but we'll walk though each code segment so you can see this powerful example clearly.

```
(vl-load-com)

(defun DoThisAfterLayoutSwitch (Caller CmdSet)
  (prompt (strcat "\nLayout tab switched to: " (getvar "ctab")))
  (if (= (getvar "userr1") 0.0)
    (setvar "userr1" (getvar "dimscale"))
  )
  (if (= (getvar "ctab") "Model")
    (progn
      (setvar "textsize" (* (getvar "dimtxt") (getvar "userr1")))
      (setvar "dimscale" (getvar "userr1"))
      (setvar "ltscale" (* 0.375 (getvar "userr1")))
    )
    (progn
      (setvar "textsize" (getvar "dimtxt"))
      (setvar "dimscale" 1)
      (setvar "ltscale" 0.375)
    )
  )
  (princ)
)
(setq MyReactor1
  (vlr-miscellaneous-reactor
    nil
    '((:vlr-layoutSwitched . DoThisAfterLayoutSwitch)
     )
  )
)
(defun Clear_My_Reactors ()
  (if (and MyReactor1 (vlr-added-p MyReactor1))
    (vlr-remove MyReactor1)
  )
)

(defun Clear_All_Reactors ( / TMP)
  (vlr-remove-all :vlr-dwg-reactor)
)
```

More Info on Reactors

Using the AutoCAD Developer help you can search on REACTORS and find a wealth of information about the various VLR functions in AutoLISP that allow access to reactors. If you take some time to experiment with the additional reactors AutoCAD offers you can do a lot more than simply detect layout tab switches.

I don't have time in this class to cover all the possibilities but I will say that the following reactors have served me well in client work I've performed:

Of particular interest are the following:

VLR-SYSVAR-REACTOR	For detecting changes to SYSVAR (DIMVAR) settings
VLR-XREF-REACTOR	For detecting changes to XREF attachments
VLR-DWG-REACTOR	For detecting file operations like OPEN, SAVE, etc.

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.

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 **CM305 - 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!