



Walt Disney World Swan and Dolphin Resort
Orlando, Florida

Keyboard Macros Revealed!

Steven LaKose - CAD & Computer Consulting

CP33-2 Learn how to create keyboard macros that will instantly increase your productivity. Attendees will learn how to create "single line" macros to automate both one-step commands and multiple-step commands, or multiple commands. We will cover entry-level AutoLISP using AutoCAD's build-in VLIDE editor. We will also cover several methods for automatic loading of these macros. Also covered will be how to take your personal macros with you from computer to computer or from job to job. You need not have any prior programming experience but should have a solid knowledge of AutoCAD commands.

About the Speaker:

Steve has 18+ years of design and drafting experience both from the production side and the CAD management side, and has been using AutoCAD since version 2.0. Steve's field is primarily in capital equipment, and he has wide experience in other fields, including electrical, P&IDs, HVAC, architectural, and concrete foundations. He specializes in 3D modeling and programming and customizing AutoCAD and its various products. Steve is now working as a consultant and is located in Connecticut.

slakose@superdata.com

What are they?

Source: *Webster's Revised Unabridged Dictionary (1913)*

Macro- \Mac"ro-\ [Gr. makro`s, adj.]
A combining form signifying long, large, great;
as Macrodiagonal, Macrospore.

In other programs, such as Ms Word, Ms Excel, a Macro usually refers to a VBA (Visual Basic for Applications) routine, or a recorder program to record keystrokes.

With AutoCAD, keyboard Macros are AutoLISP routines (usually small) assigned to a series of keystroke or keystrokes.

What they are not

They are not combination of Ctrl, Alt, Shift keys along with and other keys such as Ctrl+C.

They don't work well as transparent command or functions. If you want a function for zoom previous in the middle of another command, a toolbar button is much easier to program. Some macros that set variables can be programmed to work transparently.

The editing Macros demonstrated will only work in verb noun mode (you start a command and then you select something). They can be programmed to work in either verb/noun or noun/verb, but the code is much more complex.

Why

SPEED! Typing at the keyboard is still one of the fastest ways to input.

Why Macros versus PGP file

Can't I create them in the PGP file just as easy?

Yes If they are a single step (such as the Line command)
No If they are a multi-step (such as the Zoom Window)

Pro's

- Multiple step Macros (PGP is only a single step Macro)
- Last defined wins (The PGP file is loaded before)
- Easier portability between machines for users
- Different ones for different tasks or different disciplines
- You can load several "sets" of them at the same time

Con's

- They are harder to program than a PGP file Macro
- You have to know a bit of AutoLISP
- The program version follows "old school" command structure. For example, a copy Macro will not use the new "automatic" multiple feature.
- Warning... AutoLISP can be addictive

Example

First off, let's define a Macro to zoom previous. We will use the DEFUN AutoLISP function. This function has several options, and I will only go into what is needed.

```
(Defun C: ZP () (Command "Zoom" "P") (Princ))
```

(Open parenthesis. This is the open parenthesis for the defun function.
Defun	This is the AutoLISP function that defines a routine.
C:	Declare that our AutoLISP command can be used at the Command prompt. Without this you would have to (ZP) and the Command prompt to run it.
ZP	Defines the name of our command. This is what you will actually type at the Command prompt to run it.
()	Acts as a kind of customs department where you declare any arguments or declared variables we would use in the program and want to make local. Sound confusing? Don't worry, you will do a lot of Lisp programming before you ever have to put anything there. Since we don't have any of the above, it will work with an open and a close parentheses. The code below is what does the actual work. The code above is the trigger that initiates the program.
(Open parenthesis. This is the open parenthesis for the command function
Command	This is the AutoLISP function that performs the tasks. This particular function executes an AutoCAD command
"Zoom"	The command we are defining.
"P"	Sub function of the zoom command for previous.
)	Close parenthesis. This is the close parenthesis for the command function. The code below closes out the trigger that initiates the program.
)	Closing parenthesis. This is the close parenthesis for the defun function.

Two kinds of Macros

"Completed" AutoLISP functions

Starts and completes the entire command. An example is the Zoom Previous Macro.

```
(Defun C: ZP () (Command "Zoom" "P") (Princ))
```

"Dropped" AutoLISP functions

Starts the command, but drops out and lets the user complete it.

```
(Defun C: ZW () (Command "Zoom" "W") (Princ))
```

The user is left in the zoom command with the window option and is waiting for the user to input the two corners of the window.

What's the difference?

Not much, they both work well. They don't require the user to do anything until after the Macro is completed.

The problem comes about when the function has to interact (like pick points or select an entity) with the user. Once you exit the lisp routine, you can't re-enter it. If you wanted to put together several commands together, you need to use "complete" AutoLISP functions.

An example would be a command that changes you to another layer, starts the line command and when done, you want to return the original layer. Because you don't know how many line segments the user is going to input, you can't easily write a Macro that will do that. It can be done, but the code is much more complex, to the point is not as much as Macro as a full fledged AutoLISP routine.

Now, a few ground rules:

Parentheses

(This is an open parenthesis

) This is a close parenthesis

In AutoLISP, the number of open parentheses must equal the number of closed parentheses for the code (program) to work. So, if you have an open parenthesis you must have a close parenthesis to balance it. Please note: you can't add a bunch of spare close parentheses at the end of the program, it won't work, I've tried it.

The balancing parenthesis need not be on the same line.

That is why you see an AutoLISP program with parentheses placed seemingly at random across the "page", they are actually lined up vertically below the opening *parenthesis*.

Quotes

" This is an open quotation

" This is a close quotation

It is the same character, for open and close. Quotation marks are used in AutoLISP to indicate the starting and ending points for data or information. Just like parentheses, the number of open quotations must equal the number of closed quotations for the code {program} to work. So if you have an open quotation you must have a close quotation to balance it.

For the most part, *spaces do not matter*. When in doubt, add them in.

The AutoLISP function is always next to the left parenthesis.

This is one area where spaces do matter. The function must be placed next to the left parenthesis. In more advanced programming this may not be the case, but it will always be to the left of the data.

Integers (whole numbers) Vs Reals (decimal numbers)

Be careful with numbers. If you divide an integer into an integer, you will get an integer answer. Example, (/ 3 2). This is read as divide three by two. You will get 1 as an answer. That is how many times the whole number two goes into whole number three. If any one of the numbers is a Real, you will get a Real as an answer. You can, for the most part, use Reals at all times; it avoids problems such as this.

AutoLISP will not accept .5, it must be preceded by a zero for 0.5 If you try use .5, you will get "error: misplaced dot in input"

The exception to this is if you are using it in a command function and you quote it. Example, (Command " - Insert" "Block" Pause ". 5" ". 5" "0"). This code inserts a block named "Block", pauses for the user

to select an insertion point and then proceeds to the block scale factor of .5. AutoLISP accepts the information as a string (text) and converts it for you. Granted it works, but it is visually much harder to pick up the decimal point. At a quick glance it appears that the block is being inserted at 5 times scale factor. A better way is (Command "-Insert" "Block" Pause "0.5" "0.5" "0") or (Command "-Insert" "Block" Pause 0.5 0.5 0). Learn good programming etiquette.

Suggestions for “newbies”

KISS

Keep it simple. A Macro with a couple of lines is easier to maintain and read than a Macro this is a half a page that covers every possible scenario.

Small steps

Don't try to program a Macro that will perform a multiple steps, creating layers, styles etc, as your first attempt. You didn't learn design by tackling a major project, you probably learned by taking on a small detail or assembly.

Consistency

Just like creating consistent drawings, programming in a consistent manner makes it easier to read/rework/reuse your code. Good programmers have “blocks” of code they reuse.

Consistency Gotcha's

Things don't always work as planned. The statement “well, it works fine on my computer” is no excuse for poor programming. Try to anticipate the problems and add code to address them. Like what?

Locked layers

Make sure that you unlock a layer

```
(Command "-Layer" "U" "" <Layer name> "" )
```

Layers that don't exit

User started with a new drawing instead of the standard template drawing.

```
(Command "-Layer" "M" "" <New Layer name> "" ) Layer Make creates a layer and makes it current.
```

Text styles with fixed height (or vise versa)

Use the style command and “reset” or create the style.

```
(Command "-Style" "Romans" ""Romans.shx" 0 1 0 "N" "N" "N")
```

Fillets or chamfers that don't trim lines

Make sure you set Trim/No Trim to Trim.

Units input that don't work

Assuming that units is your unit of measure, (and that can “Gotcha” at times too) If you need 2' (two feet) give 24 instead. AutoCAD uses integers and for storing unit values

Some of these “gotcha's” you can address, some you can't. An example is starting a new drawing and trying to use the Zoom Previous Macro. There is no previous zoom. Sometimes you can't protect the user from them selves.

Note, the difference between stupidity and ignorance, is that genius has its limits.

Document

You don't have to document every line of code, (see KISS above) but a couple of commented lines don't hurt when you are creating more complex Macros or are developing a more complex one. Sometimes you will revisit a piece of code months or years after you completed it. A few lines of notes, examples, known flaws or ideas will make working on it or even reading it much easier (for either you or your predecessor).

The faintest ink is better than the fondest memory.

```

===== Add Diameter symbol
;Adds the diameter symbol to a dimension or dimensions. It will
;also clear any other text modifiers or forced text from dimensions

(Defun C:AddDia (/ SSet1)

  (Princ "\nSelect Dimensions to add diameter symbol to ")
  (Setq SSet1 (Ssget))

  (If SSet1
    (Command "DimEdit" "new" "%<c>" SSet1 "")
    (Princ "\nNothing selected")
  )

  (Princ)
)
;Close Defun on AddDia

;This is a macro within a macro. AddDia is logical, but sometimes
;you want follow a pattern. Most dimension macros start with D such as
;DLI Dimension Linear. I just following the dimension here.

(Defun C:DAD () (C:AddDia)(Princ))
  
```

Etiquette

The Macro should perform as much like the “normal” command or commands as possible. It should also work every time, regardless of a user’s local settings (within reason).

No nils

One give away is a nil after the command finishes. This is because AutoLISP always returns its last value, (if you if you type in (+ 2 3) and ENTER you will get 5). Commands like Zoom previous return nothing, so AutoLISP returns nil. Adding a (Princ) as shown in the example below avoids this problem. Note, the Macro works fine, it's just that it's sloppy programming.

Works

```
(Defun C: ZW () (Command "Zoom" "W"))
```

Leaves a nil

Better

```
(Defun C: ZW () (Command "Zoom" "W") (Princ))
```

This avoids a nil

One flaw to this Macro code is that the user may not know what command they are windowing for... It could be an erase as easily as a zoom window.

Best

```
(Defun C: ZW () (Princ " Zoom Window") (Command "Zoom" "W") (Princ))
```

This Macro tells the user what is going on and avoids a nil.

Readability

The code below is aligned and much more readable.

```
(Defun c: ZD () (Command "Zoom" "d") (Princ))
(Defun c: ZE () (Command "Zoom" "e") (Princ))
(Defun C: ZI () (Command "Zoom" "2.0X") (Princ))
(Defun C: ZO () (Command "Zoom" "0.5X") (Princ))
(Defun c: ZP () (Command "Zoom" "p") (Princ))
(Defun c: ZW () (Command "Zoom" "w") (Princ))

(Defun C: EL () (Command "Erase" "L" "") (Princ))
(Defun c: TTT () (Command "Dtext") (Princ))
(Defun C: S () (Command "Stretch") (Princ))
```

Spaces have been added to align pieces of the code. I aligned the group of the zoom commands the same way to increase readability and streamline the debugging process.

The code below is not aligned and less readable to the eye. It's not too bad as most of the lines of code are short, but a more complex code would be increasingly more difficult to read quickly and/or look for errors.

```
(Defun c: ZD () (Command "Zoom" "d") (Princ))
(Defun c: ZE () (Command "Zoom" "e") (Princ))
(Defun C: ZI () (Command "Zoom" "2.0X") (Princ))
(Defun C: ZO () (Command "Zoom" "0.5X") (Princ))
(Defun c: ZP () (Command "Zoom" "p") (Princ))
(Defun c: ZW () (Command "Zoom" "w") (Princ))

(Defun C: EL () (Command "Erase" "L" "") (Princ))
(Defun c: TTT () (Command "Dtext") (Princ))
(Defun C: S () (Command "Stretch") (Princ))
```

Single line versus multi-line

```
(Defun C: ZP () (Command "Zoom" "P") (Princ))

(Defun C: ZP ()
  (Command "Zoom" "P")
  (Princ)
)
```

Both Macros work exactly the same way. The difference is in how they are aligned and indented for readability. Remember, *the balancing parenthesis need not be on the same line*. For every line that opens a parenthesis, the next line is indented. By determining how far indented you are, you can determine how many closing parentheses are required. How far you indent is somewhat personal preference, with most people indent between 2 to 4 spaces, I personally use 3.

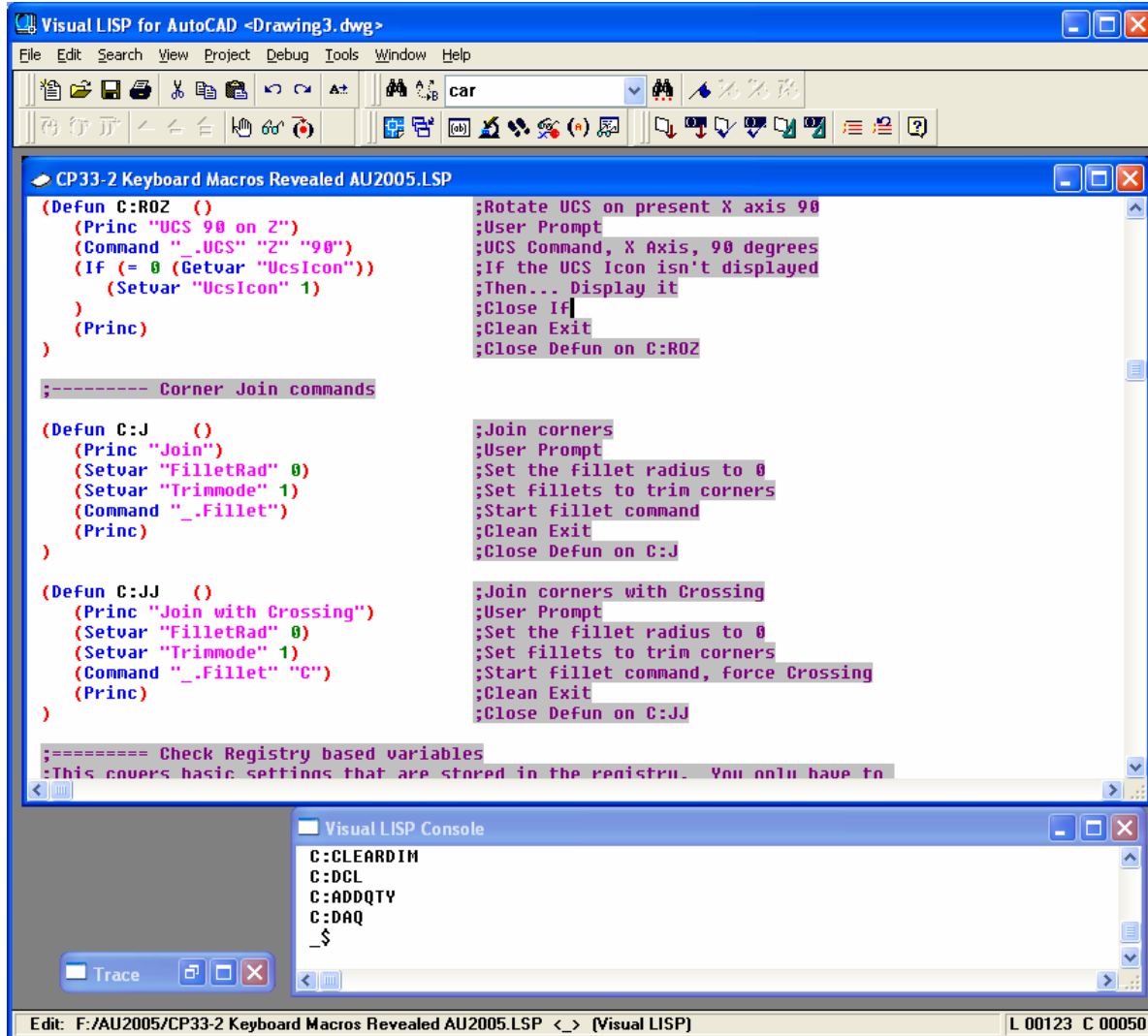
Tools for writing the Macros

Notepad

It works, every computer has the program, and it saves text in ASCII format. However, it does not provide any specific tools to help you write code. It is commonly the default editor used if you double click on a .lsp file. Its greatest asset is that it does not need AutoCAD to run.

VLIDE

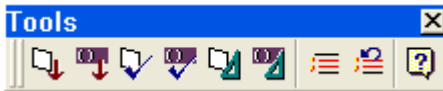
Visual Lisp Integrated Development Environment. It has been supplied with AutoCAD since AutoCAD 2000 and is designed specifically for writing AutoLISP and Visual lisp code. Visual lisp and AutoLISP are one in the same here for our purposes here. Visual lisp provides access to Active-X advanced programming abilities. Its one drawback is that you must have AutoCAD (or a vertical application) running to use it.



The example shows the code for our class, color coded with blue indicating lisp commands, magenta indicating strings, green indicating numbers both real and integers and grey indicating comments.

The console window displays the output from evaluated code. It's comparable to the text window in AutoCAD.

The bottom edge of the IDE window contains the Status bar. This is where messages are displayed following every action in the IDE. The bottom-right panel is the code editor cursor location display. This shows the current position of the cursor in the code file where *L nnnnn* is the line number and *C nnnnn* is the character offset number. In the Example above, the cursor is positioned on line 123 on the 50th character of that line.



This is the toolbar we will be using, primarily the load selection and check selection buttons.

The TOOLS toolbar contains general editor features from left to right:

- Load File
- Load Selection
- Check File
- Check Selection
- Format File
- Format Selection
- Comment Selection
- Uncomment Selection
- Help

Developing the Macros

Has the letter or letters already been used?

Easy way to tell, launch AutoCAD (what ever flavor you have) and type the letter or letters at the Command prompt and press enter. If you get:

“Unknown command “XXX”. Press F1 for help.”

It’s available! If it launches a command or does something, it’s used or defined somewhere.

Note: If you are a multi discipline user (you use AutoCAD and ADT or AutoCAD and Mechanical) I strongly suggest you check it in all disciplines. This goes back to Etiquette issues.

If you are not sure where it is defined (and you need to know), at the Command prompt, type in “!C:EL” and press ENTER (where EL is the Macro). If it’s an AutoLISP defined command, you will see something like #<SUBR @0f921c1c C: EL>. If it’s defined in the PGP file you will see a nil. Knowing this is useful if you have to try to trace down where it’s defined and/or by whom.

If it’s used, can you redefine it?

Yes you can redefine it. If the Acad.pgp file defines a Macro and Macros is defined in the AcadDoc.lsp file the last defined wins. Below is a chart describing the load order of files, with s::startup being the last loaded.

File	For use by:
<i>Acad.pgp</i>	AutoCAD / User
<i>acad200x.lsp</i>	AutoCAD (x is for the AutoCAD version number)
<i>acad.rx</i>	User (Optional, doesn't exist)
<i>acad.lsp</i>	User (Optional, doesn't exist)
<i>acad200xdoc.lsp</i>	AutoCAD (x is for the AutoCAD version number)
<i>acetutil.fas</i>	Express Tools
<i>acaddoc.lsp</i>	User (Optional, doesn't exist)
<i>Custom.mnc</i>	User (Optional, exist in later releases of AutoCAD)
<i>Custom.mnl</i>	User (Optional, exist in later releases of AutoCAD)
<i>acad.mnc</i>	AutoCAD (or vertical application version of this)
<i>acad.mnl</i>	AutoCAD (or vertical application version of this)
<i>acetmain.mnc</i>	Express Tools
<i>acetmain.mnl</i>	Express Tools
<i>Startup Suite</i>	User
<i>s::startup</i>	AutoCAD / User (Defined in the Acaddoc.lsp file)

If it's used, can you and/or should you redefine it?

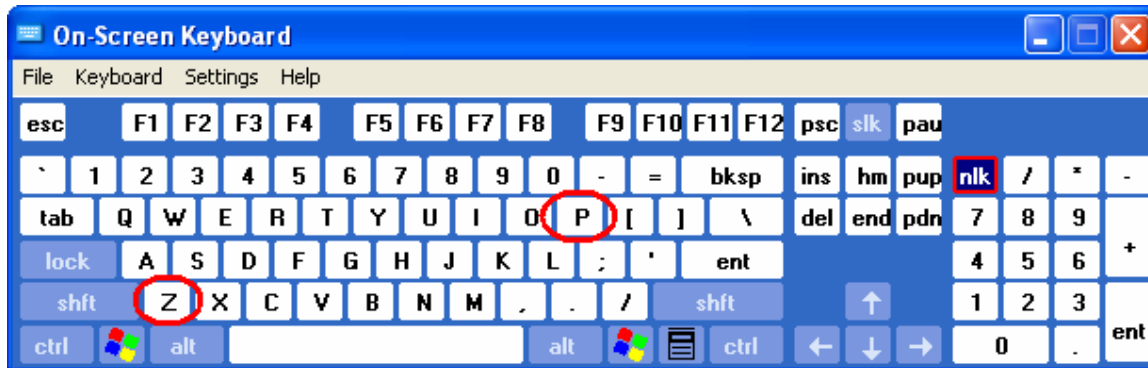
This question is much bigger. If you are the only user on a system and you are only programming for yourself, go ahead and do it. The idea behind Macros is speed. Redefining an un-used or rarely used Macro for one you would use all the time it makes sense. An example is EL. In stock AutoCAD it's the ellipse command. I rarely draw an ellipse, so I use it for Erase Last (I make a lot of mistakes).

Very few employers are going to complain that you are performing the same quantity of work faster.

If you are the CAD manager and/or power user who is writing this for everyone to use, it's time to sit down and have a talk with the users.

Speed versus Logic

A common Macros most users first write is one for Zoom Previous. The logical Macro for this is ZP, makes sense. Did you ever look where the Z key and the P keys are located?



They are located at opposite ends of the keyboard. A faster Macro would be ZZ, but a lot less logical.

Sometimes you have to balance logic and speed together.

Programming Examples

Basic

```

CP33-2 Keyboard Macros Revealed AU2005.LSP
;----- Zoom macros
(Defun c:ZD () (Command ""_.Zoom" "d") (Princ))
(Defun c:ZE () (Command ""_.Zoom" "e") (Princ))
(Defun c:ZI () (Command ""_.Zoom" "2.0X")(Princ))
(Defun c:ZO () (Command ""_.Zoom" "0.5X")(Princ))
(Defun c:ZP () (Command ""_.Zoom" "p") (Princ))
(Defun c:ZW () (Command ""_.Zoom" "w") (Princ))
    
```

Defines 6 simple (and handy) zoom Macros

More advanced

```

;----- Corner Join commands

(Defun C:J ()
  (Princ "Join")
  (Setvar "FilletRad" 0)
  (Setvar "Trimmode" 1)
  (Command "_Fillet")
  (Princ)
)

(Defun C:JJ ()
  (Princ "Join with Crossing")
  (Setvar "FilletRad" 0)
  (Setvar "Trimmode" 1)
  (Command "_Fillet" "C")
  (Princ)
)

;Join corners
;User Prompt
;Set the fillet radius to 0
;Set fillets to trim corners
;Start fillet command
;Clean Exit
;Close C:J

;Join corners with Crossing
;User Prompt
;Set the fillet radius to 0
;Set fillets to trim corners
;Start fillet command, force Crossing
;Clean Exit
;Close C:JJ

```

These are a couple of simple Macros to cleanup or close corners. Since these are sort of hybrid commands, we are telling the user that it is a Join or Join with Crossing.

It would work the same with:

```
(Defun C:J () (Command "._Fillet" "T" "Trim" "R" 0) (Princ))
```

But I am using this to introduce the Setvar function.

Setvar tells AutoCAD to set a variable. One big difference is that the variable names that control this command don't work the same as in the command.

In the fillet command it's Trim/No Trim, while the variable FilletRad has a binary value, either 0 no or No Trim and 1 for yes or Trim.

The radius value in the fillet command gives you a prompt of the current value (displayed in the current units format), it's retrieving the value stored in FilletRad variable. In this example 0 will work regardless of the units setting.

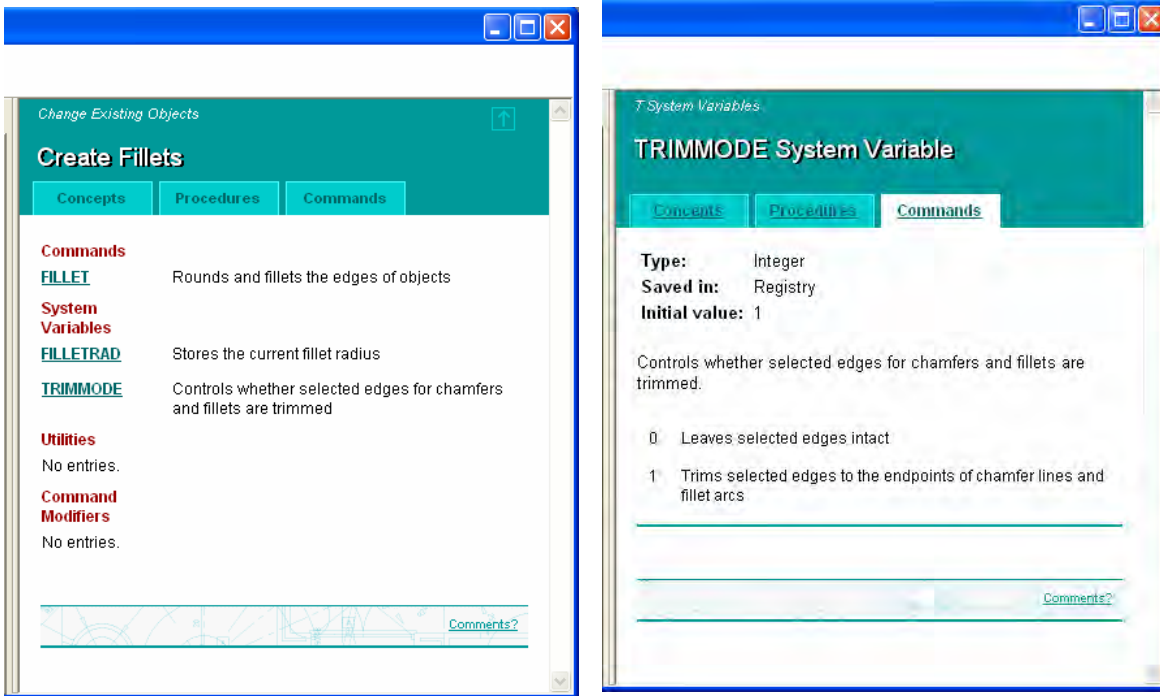
If you tried to enter 2' (two feet) it may or may not accept the value depending on the units settings.

Note: Using KISS mentality, I don't actually check to see if the fillet radius is already at 0, or that the fillets are set to trim the corners, I just set them.

Finding System Variables

How do you find these variables names? Simply, launch a command in AutoCAD and press the F1 key. The help screen comes up with help for that command. Press the Commands tab and any a list of system variables is displayed.

Click on the system variable and it will display information about that variable including, the type of values it stores, where it is stored and its initial value. It also describes what it is and how different values affect the command.



Why?

To avoiding unexpected results. Most users have fillet command set to trim, but just in case it's set to not trim, this command will work as expected. Just like your designs, you want your Macros to work and work every time.

Even more advanced

```

CP33-2 Keyboard Macros Revealed AU2005.LSP

;----- Rotate the UCS

(Defun C:RO3 ()
  (Princ "UCS 3 points")
  (Command "_UCS" "3")
  (If (= 0 (Getvar "UcsIcon"))
    (Setvar "UcsIcon" 1)
  )
  (Princ)
)

;Rotate UCS to 3 point
;User Prompt
;UCS Command, 3 points
;If the UCS Icon isn't displayed
;Then... Display it
;Close If
;Clean Exit
;Close C:RO3

(Defun C:ROE ()
  (Princ "UCS Entity")
  (Command "_UCS" "E")
  (If (= 0 (Getvar "UcsIcon"))
    (Setvar "UcsIcon" 1)
  )
  (Princ)
)

;Rotate UCS to Entity
;User Prompt
;UCS Command, Entity
;If the UCS Icon isn't displayed
;Then... Display it
;Close If
;Clean Exit
;Close C:ROE

(Defun C:ROP ()
  (Princ "UCS Previous")
  (Command "_UCS" "P")
  (If (= 0 (Getvar "UcsIcon"))
    (Setvar "UcsIcon" 1)
  )
  (Princ)
)

;Rotate UCS to Previous
;User Prompt
;UCS Command, Previous
;If the UCS Icon isn't displayed
;Then... Display it
;Close If
;Clean Exit
;Close C:ROP
    
```

This introduces the If function along with the equals and the Getvar functions. Again applying KISS mentality, it would work just setting the Ucsion to 1.

If is relatively simple function, it asks a “question” and depending on it’s answer (either T for True or nil for False) processes either the Then or an optional Otherwise sections of code.

Equals works almost as easily, it’s the question part used in the if function.

Getvar asks AutoCAD to retrieve or get a setting or variable. Here it’s asking to get the value for the UcsIcon variable. (Note, it’s not case sensitive).

Setvar tells AutoCAD to set a variable.

Read all together, If the Ucsion is set to 0 (or off) Then turn it on. There is no otherwise code to perform, so of the UCS icon is already on, it will do nothing.

Why?

Unexpected results happen. Since the RO is the rotate Macro, a user could accidentally press RO and one of the other keys and ENTER. All of a sudden a user is drawing lines but in only one axis. If you don’t have the UCS Icon turned on, it’s difficult to tell what happened.

Side Note:

You will see that the example use “._Fillet” or “._Zoom”. What do the underscore and the period do?

The Underscore is for language issues. If this is used on a language other than English, the prompts should come out in the other language.

The period overrides undefined or redefined commands. This helps you avoid unexpected results.

Lets write some Macros!

Beyond Simple Macros

Think outside the box... You don't need to only use Macros that shorten the key strokes necessary to perform the zoom previous command.

Macros to load other Macros

Do you need to work in different disciplines where you would have conflicting Macros? Define a Macro that does nothing but load other Macros, or other additional Macros.

```
(Defun C: LM1 () (Load "Macro1")(Pri nc))
(Defun C: LM2 () (Load "Macro2")(Pri nc))
```

Where we have created a series of Macros and saved to a filename called Macro1.lsp. This also introduces the Load function. Remember, if "C" is defined in both, the last loaded wins.

Macros to set things

Ever had the task of cleaning up drawings from a vender or sub-contractor? Write individual Macros for each of those tasks such as loading layers, text styles, purging etc. You can then either run then or even call them from a toolbar menu. (Sometimes it's easier to modify a Macro than change a toolbar button definition).

Macros to automate a series of tasks

Ever have the task of "issuing" a set of drawings, which require adding a date and initials stamp or stamps, purging, saving and then plotting both a hard copy and a DWF? Write individual Macros for each of those tasks (small steps) and then string them all into a single Macro?

Plotting Automation Macros

- Macros that plot
- Macros that plot and then close the drawing
- Macros that load page standard setups

Additional Macros in the example file

These are a few of the Macros I use on a regular basis and have been provided in the example file. You can download off of the website listed on the last page.

ECD	Explorer to Current Directory. Handy when you have to navigate an Explorer when your drawing is multiple folders deep.
ONF	Open Next File. Opens the next sequential drawing in the folder.
OPF	Open Previous File. Opens the previous sequential file in the folder.
DSS	Double Snap setting
HSS	Half Snap Setting
ADD DIA	Adds the diameter symbol to a dimension or dimensions. It will also clear any other text modifiers or forced text from dimensions.
CLEARDIM	Clears all dimension text modifiers and forced text from dimensions
ADDQTY	Adds a quantity to a dimension or dimensions. It will also clear any other text modifiers or forced text from dimensions. It leaves a global variable #Qty to use next time its run.
FIX1	Sets basic settings and variables that are stored in the registry.
FIX2	Sets basic settings and variables that are stored in the drawing or not stored at all.

Loading your Macros

Now that you have developed all of these wonderful Macros, how do you load them? The answer will depend on your own work setup. One of the options below should fit your unique work environment

At the Command prompt

At the Command prompt,

Command: (Load "Macro name") and ENTER

No file extension is required, AutoCAD automatically will look for .lsp and other AutoLISP/Visual lisp file extensions. If you do provide a file extension, AutoCAD will only look that file extension.

Pro's

- Automatically searches all support folders, You never have to leave the command area

Con's

- You have to remember the file name
- Typing is required
- You also have to be perfect on the syntax
- You can only do one file at a time

This is a simple fast way to load your Macros, especially if you type. Drawback is that you must do it EVERY time you open a drawing.

Drag and drop load

Find the file in Explorer, drag and drop it into any part of the AutoCAD drawing area (not on the toolbars or the Command prompt).

Pro's

- You can visually search for files
- No typing involved

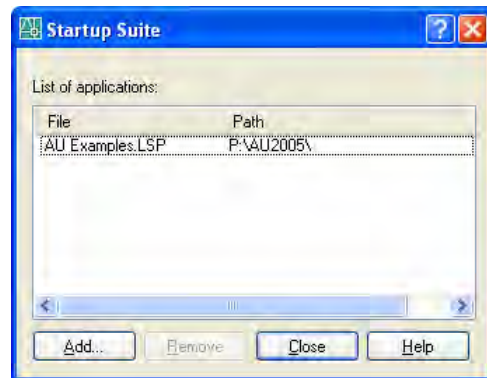
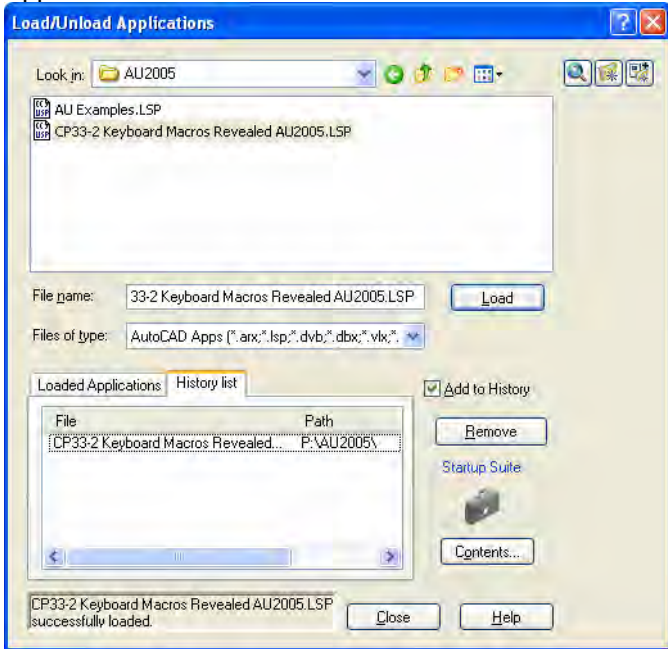
Con's

- You have to navigate to the correct folder
- You can only do one file at a time

This is another simple fast way to load your Macros and gets you into the habit of using drag and drop. Again, the drawback is that you must do it EVERY time you open a drawing.

Load them in the Load Application Startup Suite

AutoCAD provides way to load repetitive files with a dialog box interface. You can even automatically load files when a drawing opens. You can either type APPLOAD at the Command prompt or Tools Pulldown > Load Application...



If “Add to history” is checked, it keeps a log of files loaded along with their path.

If you click on the Startup suite contents button it shows you which routines are to be loaded every time AutoCAD opens a drawing. You can add or remove files as required.

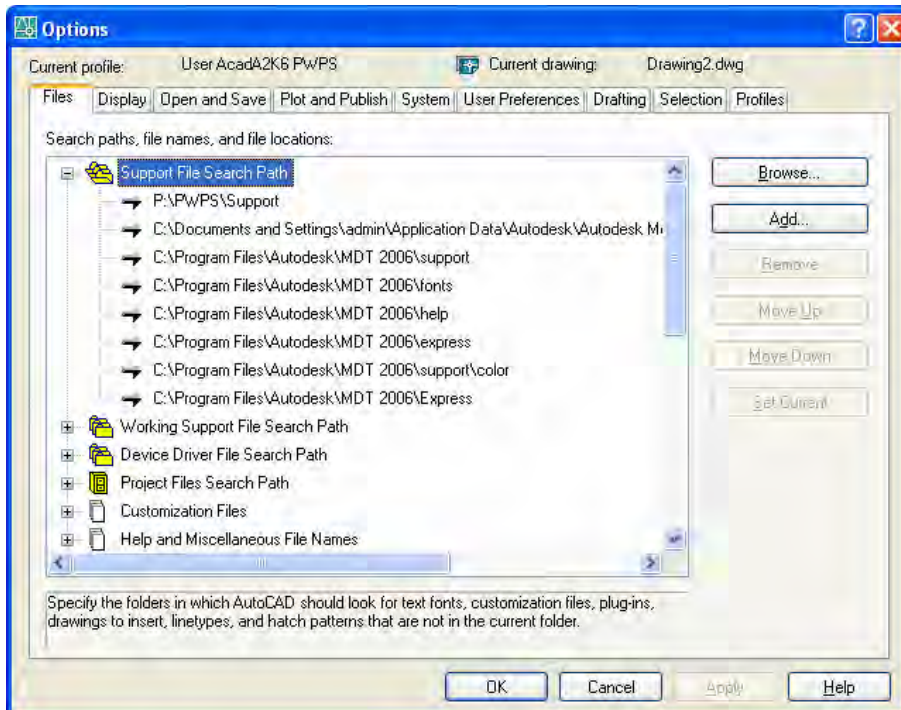
This is the easiest way to load your Macros automatically on a single machine or where you already have an existing AcadDoc.lsp file defined and you can't change it. Drawbacks are that it has to be setup on each machine, and a user can easily change it.

Note, there is only one startup suite and its contents are not associated to the profile. This means that if you need different Macros loaded for different disciplines, you may need to use the AcadDoc.lsp file instead.

In the AcadDoc.lsp

If you create your Macros in a file called Acaddoc.lsp and place it in the one of the support paths, AutoCAD will automatically load it each time it opens a drawing. This is a very powerful file, as is it cannot only define them, it can load or do other things.

If you are a CAD Manager or a power user who needs to load Macros on several machines, this is the easiest way to go. Simply place it in a network location, and add that folder to everyone's search paths. Any changes you make will be reflected the next time users open a drawing. In the example below, I add P:\PWPS\Support to the top of the search file search path in the options dialog box. Trivia: The search order is from top to bottom.



Problems will arise if the file already exists. How do you find out if one has been loaded? At the Command prompt type (Fi ndfi le "Acaddoc. lsp") and press ENTER. This function searches for the file in all of the support folders. If it returns a path and file name, it exists, if it returns nil, it does not exist in its search paths.

If the file does exist, it may have been created by someone in your organization (maybe your CAD manager or another power user?) Even by an Autodesk vertical application (Mechanical Desktop creates one).

In the case of Mechanical Desktop (MDT), I just added the code to my existing Acaddoc.lsp and made sure the support folder for my Acaddoc.lsp file was above the folder where the Autodesk Acaddoc.lsp file was located. To see the order of the folders, go to tools > Options, select the files tab and expand the search folders. AutoCAD finds the first Acaddoc.lsp file and load it.

In my code example, I have a section for use with users who want some of their own Macros along with the company standard Macros. It searches for a user Macro file and if found, loads it.

If it is placed above the definitions of your "standard" Macros, the company standard Macros will always supercede (last loaded wins). If you place it after your "standard" Macros the users Macros will supercede. Your choice...

Function definitions

Defun

Defines a function

```
(defun sym ([arguments] [/ variables...]) expr...)
```

sym

A symbol naming the function.

arguments

The names of arguments expected by the function.

/ variables

The names of one or more 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.

expr

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 Values

The result of the last expression evaluated.

Command

Executes an AutoCAD command

```
(command [arguments] ...)
```

Arguments

The arguments to the command function can be strings, reals, integers, or points, as expected by the prompt sequence of the executed command. A null string ("") is equivalent to pressing ENTER on the keyboard. Invoking command with no argument is equivalent to pressing ESC and cancels most AutoCAD commands.

The command function evaluates each argument and sends it to AutoCAD in response to successive prompts. It submits command names and options as strings, 2D points as lists of two reals, and 3D points as lists of three reals. AutoCAD recognizes command names only when it issues a Command prompt.

Note that if you issue command from Visual LISP, focus does not change to the AutoCAD window. If the command requires user input, you'll see the return value (nil) in the Console window, but AutoCAD will be waiting for input. You must manually activate the AutoCAD window and respond to the prompts. Until you do so, any subsequent commands will fail.

Return Values

nil

Princ

Prints an expression to the command line, or writes an expression to an open file

```
(prin c [expr [fi l e-desc]])
```

This function is the same as prin1, except control characters in expr are printed without expansion. In general, prin1 is designed to print expressions in a way that is compatible with load, while princ prints them in a way that is readable by functions such as read-line.

Arguments

expr

A string or AutoLISP expression. Only the specified expr is printed; no newline or space is included.

file-desc

A file descriptor for a file opened for writing.

Return Values

The value of the evaluated expr. If called with no arguments, princ returns a null symbol.

If

Conditionally evaluates expressions

```
(i f testexpr thenexpr [el seexpr])
```

Arguments

testexpr

Expression to be tested.

thenexpr

Expression evaluated if testexpr is not nil.

elseexpr

Expression evaluated if testexpr is nil.

Return Values

The if function returns the value of the selected expression. If elseexpr is missing and testexpr is nil, then it returns nil.

See Also The [progn](#) function.

= (equal to)

Compares arguments for numerical equality

```
(= numstr [numstr] ...)
```

Arguments

numstr

A number or a string.

Return Values

T, if all arguments are numerically equal; otherwise nil . If only one argument is supplied, = returns T.

Getvar

Retrieves the value of an AutoCAD system variable

```
(getvar varname)
```

Arguments

varname

A string or symbol that names a system variable. See the Command Reference for a list of current AutoCAD system variables.

Return Values

The value of the system variable; otherwise nil, if varname is not a valid system variable.

Setvar

Sets an AutoCAD system variable to a specified value

```
(setvar varname value)
```

Arguments

varname

A string or symbol naming a variable.

value

An atom or expression whose evaluated result is to be assigned to varname. For system variables with integer values, the supplied value must be between -32,768 and +32,767.

Return Values

If successful, setvar returns value.

Load

Evaluates the AutoLISP expressions in a file

```
(load filename [onfailure])
```

The load function can be used from within another AutoLISP function, or even recursively (in the file being loaded).

Arguments

filename

A string that represents the file name. If the filename argument does not specify a file extension, load adds an extension to the name when searching for a file to load. The function will try several extensions, if necessary, in the following order:

.vlx

.fas

.lsp

As soon as load finds a match, it stops searching and loads the file.

The filename can include a directory prefix, as in "c:/function/test1". A forward slash (/) or two backslashes (\\) are valid directory delimiters. If you don't include a directory prefix in the filename string, load searches the AutoCAD library path for the specified file. If the file is found anywhere on this path, load then loads the file.

onfailure

A value returned if load fails.

If the onfailure argument is a valid AutoLISP function, it is evaluated. In most cases, the onfailure argument should be a string or an atom. This allows an AutoLISP application calling load to take alternative action upon failure.

Return Values

Unspecified, if successful. If load fails, it returns the value of onfailure; if onfailure is not defined, failure results in an error message.

Findfile

Searches the AutoCAD library path for the specified file or directory

(findfile filename)

The findfile function makes no assumption about the file type or extension of filename. If filename does not specify a drive/directory prefix, findfile searches the AutoCAD library path. If a drive/directory prefix is supplied, findfile looks only in that directory.

Arguments

filename

Name of the file or directory to be searched for.

Return Values

A string containing the fully qualified file name; otherwise nil, if the specified file or directory is not found.

The file name returned by findfile is suitable for use with the open function.

References

Available on the Midpoint LLC web site at <http://www.midpointcad.com/au/2005.htm>

CP33-2 (this document).

CP33-2 Keyboard Macros Revealed AU2005.lsp

Acaddoc.lsp

The Visual LISP Developers Bible.

AutoLISP for Dummies I,

AutoLISP for Dummies II,

PDF by Steven LaKose

Sample Macro file

Sample file

Ebook by David M. Stein

PDF by Steven LaKose

PDF by Steven LaKose