AS322254

Making Your AutoCAD Work Faster (for People Who Don't Want to Learn to Code)

T.J. Meehan AIA, LEED AP
CADD Microsystems, Inc.

Learning Objectives

- Understand the simplest way to build your own functions and take them with you
- Learn how to create dozens of time-saving functions in AutoCAD
- Walk away with a library of hundreds of these functions that you can use
  Understand the differences between script files, LISP files, and PGP files

Description

Are you an experienced AutoCAD user? Do you want to get even better and faster at it? Have you looked at other sessions about coding or customizing AutoCAD and thought they seemed too complicated? This session is for you! Learn how to do simple automation tasks to save you time and be more productive. Even though we will use LISP, this is NOT A CODING CLASS. If you can type some commends at the Command Line, we'll use that knowledge to help you build some simple automated functions. For example, what if you could just type in "4" at the command line and you're automatically offsetting 4", or type in "ZZ" and you zoom to extents and then back out 10%? You will learn many, many more of these simple tools and how you can set them all up in one or two text files to take with you the rest of your AutoCAD career.

Speaker

As an expert and thought leader in the AEC industry around BIM (Building Information Modeling) and FM (Facilities Management), T.J. assists architects, engineers, contractors, and building owners with successfully implementing efficient workflows with regards to design, build, and operate. After receiving his architectural degree, he began his career by working in several architectural firms across the U.S. where he gained experience on both commercial and residential projects. Transitioning from the design-side of the industry, he has now become a recognized expert on the technology-side, working for one of the nation's most successful Autodesk Partners as their VP of Technology Solutions. He is an Autodesk Implementation Certified Expert and a veteran presenter at many industry events, including Autodesk University. A registered architect and LEED accredited professional, T.J. leverages his skills and experience to help clients successfully implement BIM and FM solutions.

AUTODESK. UNIVERSITY

Table of Contents

Introduction 3
File Types 4
Program Parameters (PGP) File 5
- Purpose 5
- Location 5
- Format 6
- How to Use 6
- Tips / Best Practices 7

Script (SCR) Files 8
- Purpose 8
- Format 8
- How to Use 9
- Tips / Best Practices 10

AutoLISP (LSP) Files 11
- Introduction 11
- A Very Brief History 11
- Purpose 12
- Format 12
- How to Use 13

Deeper Dive 14
- Basics of Defining a Function 14
- Acad.lsp 15
- Tips / Best Practices 15

Setting It Up 19
- Folders and Files 19
- Settings in Your AutoCAD OPTIONS 19
- Acad.lsp Settings 20

Examples 21
Conclusion 28

Introduction

As the title implies, this handout – and its corresponding class at Autodesk University 2019 in Las Vegas – is all about learning different ways to be faster in AutoCAD through automation and shortcuts to commands, without having to learn to do software coding.

Whether you are just trying to use less steps when drawing in AutoCAD or trying to automate some repetitive tasks, this handout will help.

It is assumed you are experienced using AutoCAD and know most of the commands people use every day. The content in this handout is geared towards those with at least a couple years' worth of continuous AutoCAD use. It is designed for intermediate level to "Power Users".

> *If you can type a command at the command line and understand the different options displayed, that is the only skillset you need to take advantage of this handout.*

Even though we will be discussing a coding language, AutoLISP, this is not a tutorial on coding. We will focus on how to format what you are already typing in at the command line in LISP so you can group commands together. This handout will not cover anything more complicated than replicating commands on the command line in AutoCAD. We will not be covering topics such as:

- IF/Then statements
- Variables
- Anything else that looks like a propeller head wrote it
  (even though I am a propeller head in training myself)

Also, everything we build in this handout should work on any version of AutoCAD (all the way back to the "Release" days) as well as any "flavor" of AutoCAD, including:

- AutoCAD
- AutoCAD Architecture
- AutoCAD MEP
- AutoCAD LT
  (only for part of this handout, as LT does not support AutoLISP)
- AutoCAD Electrical
- AutoCAD Mechanical
- AutoCAD Map 3D
- AutoCAD P&ID
- AutoCAD Plant 3D
- Civil 3D

If you ever have any questions or comments about the content, don't hesitate to reach out to me directly at tj.meehan@caddmicrosystems.com.

Thank you! I hope you enjoy this handout.
T.J.

File Types

There are three file types we are going to cover in this handout.

PGP	SCR	LSP
		
AutoCAD Program Parameters	AutoCAD Script	AutoLISP Application

These are files that you can **edit with any text editing program, such as Notepad**. You don't need to use specialized software like Microsoft Visual Studio or even the built-in Visual LISP Editor in AutoCAD.

Each of these files has a special purpose, explained in detail in the following pages. These are **files you can take with you**. You can keep them in a folder and no matter where you go, you can utilize them. More on this in the section titled "Setting It Up".

Program Parameters (PGP) File

Purpose

The PGP file in AutoCAD has a specific purpose. There is only one of these files and it is part of every AutoCAD installation. Sometimes people call it the “pig pen” file.

The PGP file’s sole purpose is to remember the “key-ins” or “aliases” for the standard commands. In other words, the shortcuts you can type. For example:

- Typing in “T” at the command line starts the MTEXT command
- Typing in “C” at the command line starts the CIRCLE command (even though we all want it to be COPY…more on this later)

This, of course, saves you time. Time both searching for the command on the ribbon and time typing it in. As you are probably aware, most AutoCAD Power Users work with one hand on the mouse and one hand on the keyboard, using that keyboard hand to type in these “key-ins” followed by the Space Bar with their thumb to execute them.

Location

This file, by default, is located here:

C:\ Users \ ***USERNAME*** \ AppData \ Roaming \ Autodesk \ ***ACA 2020*** \ enu \ Support

Obviously, replace the *USERNAME* with yours and *ACA 2020* with your product.

You can also access it directly within AutoCAD 2020…

1. From the “Manage” ribbon
2. Go to the “Customization” panel
3. Click the flyout (down ▼ arrow)
4. Click the “Edit Aliases” dropdown
5. Click the “Edit Aliases” button again

This will open the default Acad.pgp file on your computer.

Format

The format of a PGP file is easy. It's a simple text document with each line/row representing an alias. For example:

```
      T,          *MTEXT
      C,          *CIRCLE
```

You will also see a lot of other lines/rows that begin with a semicolon. This denotation is just a way to "remark" out a line/row so it is ignored by the software.

You can add your own aliases by following the format above. The number of spaces between the comma "," and the asterisk "*" does not matter. In fact, using TABS helps keep it clean. Upper or lower case does not matter as well.

How to Use

There are instructions at the bottom of the PGP file for how you should use it:

```
;  -- User Defined Command Aliases --
;  Make any changes or additions to the default AutoCAD command aliases in
;  this section to ensure successful migration of these settings when you
;  upgrade to the next version of AutoCAD.  If a command alias appears more
;  than once in this file, items in the User Defined Command Alias take
;  precedence over duplicates that appear earlier in the file.
```

Simply place your aliases below this set of instructions. That way, as it says, you can easily transfer them to other installations or newer versions of the software.

Tips / Best Practices

- Only one PGP file can load in AutoCAD. This means you cannot have your own with just your items loading and then have the out-of-the-box one load afterwards with the remaining standard commands.

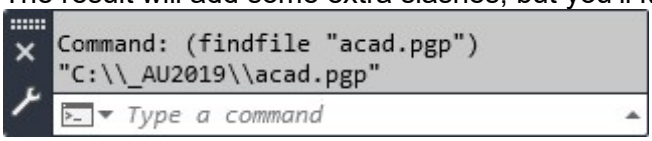
- Place the path to your customization folder in the Support File Search Path higher in the list than the default location (more on this in the section titled "Setting It Up") to ensure your PGP file loads first.

- I suggest you don't edit the Acad.pgp file that gets installed by default. Instead, make a copy and place it in your customization folder (more on this in the section titled "Setting It Up"). There are two reasons for this:
  1) the default installation folder is not usually backed up so you could lose your edits
  2) having your own folder makes it easier to manage

- If you are not sure which PGP file is loading, you can type following at the command line to know the path

```
(findfile "acad.pgp")
```

  The result will add some extra slashes, but you'll know which file is loading:



- Suggested additions:

```
C,   *COPY          ;(replacing C for CIRCLE)
CI,  *CIRCLE
Q,   *QSAVE         ;(another way to save your drawing quickly vs a full SAVE)
R,   *RECTANG       ;(replacing R for REDRAW)
T,   *TRIM          ;(replacing T for MTEXT)
```

Script (SCR) Files

Purpose

Script files are ideal for saving a series of AutoCAD commands to a text file to run all those commands on a drawing by simply calling that script file and standing back while it runs. Examples of the types of tasks people often put in script files include:

- Creating all your standard layers in a drawing
- File cleanup steps (set Model Space current, Purge, Audit, Zoom Extents, etc.)

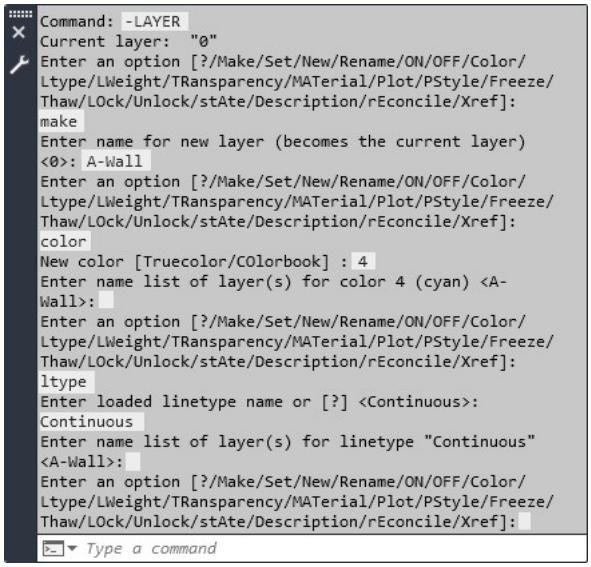
You can build as many script files as you like and share them with others. **Script files cannot access dialog boxes, toolbar buttons, palettes, etc.** They are command line syntax only!

Format

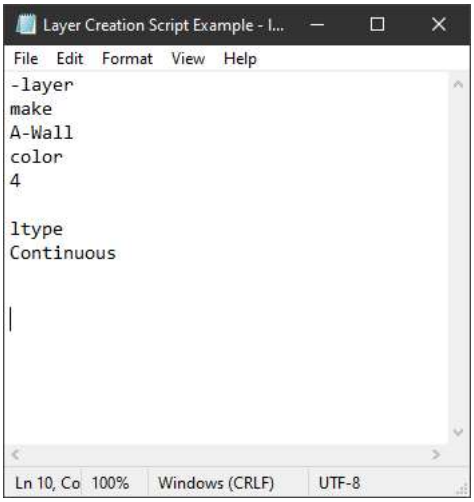
Script files are simply text (TXT) files saved with an SCR extension. You can edit them in any text editor, the easiest of which is Notepad in Windows. For this handout, all the examples will be shown in Notepad.

The format of a script file can take a few forms, but at its simplest, it mimics how you would type at the command line in AutoCAD. For example, here is a side-by-side comparison of creating a new layer on the command line (not through the Layer Properties palette) and what a script file would look like.

COMMAND LINE



SCRIPT FILE



A few things to notice:

- Just like the command line, an ENTER or a SPACE count the same (see the "Tips / Best Practices" section for more help on this).

- Notice the three line returns in the Script File example on the right.

- You'll also notice that the `-layer` command starts with a "-" (dash/hyphen) at the beginning. That's the universal designation in AutoCAD to force a command to not use its dialog box but, instead, run completely from the command line. There are many popular commands that support this, including:
  -LAYER
  -PLOT
  -BLOCK
  -PURGE
  And many others

- There used to be a great tool by Autodesk called "ScriptPro" that would allow you to run a script file on a group of DWG files. It was free, but it hasn't been updated or released since at least 2013. If this is something you need, there are some other paid utilities on the market to help you with this.

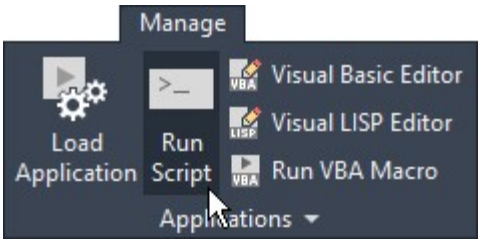
How to Use

This is easy. Simply type SCRIPT (or even SCR, which is the default alias for it…see the PGP section above for more on this) at the command line.

You can also access it directly in AutoCAD 2020…

1. From the "Manage" ribbon
2. Click the "Run Script" button

This will open the browse window for you to navigate and find your script file.



Tips / Best Practices

- Since every line in a script file includes an ENTER, be careful of extra lines in the file which will act as blank ENTER's on the command line, thereby repeating the last command.

  If you "Select All" (by using CTRL + A) to copy everything in your script file and then paste it into a blank Microsoft Word document, you can turn on the "Show/Hide Non-Printing Marks" button to see all the extra lines and spaces: The button is on the Home ribbon and looks like this:

  You'll be amazed at all the stuff you can find with this button. It's one of my favorites in Word!

- You don't need to use a separate line for every piece of text. You can keep the entire command and all its options in one line, utilizing SPACE's instead of ENTER's. For example, this script file…

```
-layer
make
A-Wall
color
4

ltype
continuous
```

  …could instead be written this way…

```
-layer make A-Wall color 4  ltype continuous
```

  Often, people will put an entire command and it's options on one line in the script file this way to save space and better see all the commands that are running. Imagine if you were creating 100 layers how long the script file could be if you didn't place each command on a single line.

- Since SPACE is considered an ENTER, if you have something to type in that has a space in it (like a layer name), simply enclose it in quotes. For example:

```
-layer make "Arch Walls" color 4  ltype continuous
```

- These files are not usually version specific (and can be run by any version of AutoCAD), but Autodesk occasionally changes the options within a command, the order of the options, or even the entire command. Make sure you test your scripts on new versions of the software you rollout before giving everyone access to them.

AutoLISP (LSP) Files

Introduction

Don't panic! We are going to talk about AutoLISP in this handout and the class. We're only going to cover it related to utilizing it for some special shortcut commands. Our goal is to replicate a series of commands you would type at the AutoCAD command line and save that to a couple letters you can type in.

We will **NOT** be learning this…

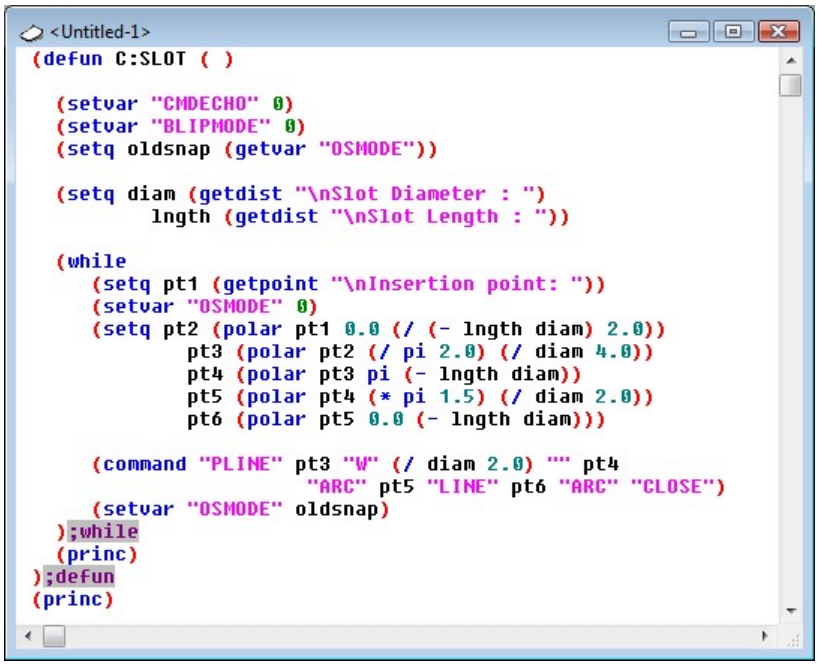


IMAGE FROM AFRALISP, WWW.AFRALISP.NET

There are many other classes and wonderful online resources for this type of in-depth coding.

A Very Brief History

- AutoLISP is a programming language used in some of the very first releases of AutoCAD (circa 1986)
- It's a modified version of earlier LISP and XLISP programming languages, both of which designed for "list processing"
- AutoLISP continues to be supported today in all AutoCAD-based products.

Purpose

AutoLISP is extremely powerful and it can dig deep into entities and the DWG file overall. It even includes its own User Interface (UI) called Dialog Control Language (DCL), albeit a very limited one. Technically, you can write almost any function in AutoCAD via AutoLISP, gathering user input, querying the entities in the drawing, performing if/then statements and condition loops, and the original language is even the basis for some Artificial Intelligence (AI) applications today.

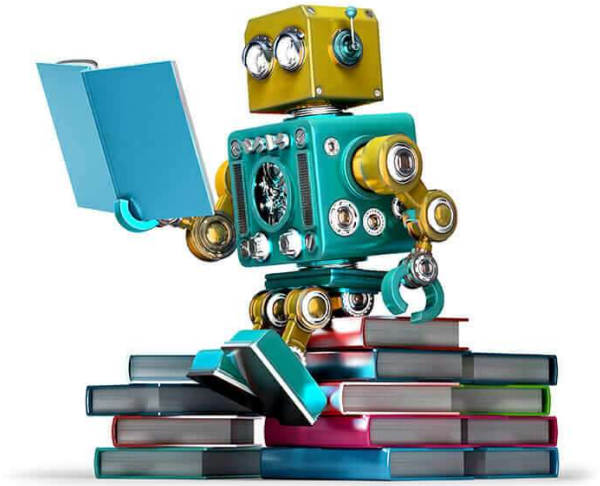


IMAGE COURTESY OF AUTODESK
HTTPS://WWW.AUTODESK.COM/REDSHIFT/MACHINE-LEARNING/

Format

Similar to the PGP and SCR files from the previous sections in this handout, an AutoLISP file is simply a text (TXT) file with the file extension changed to LSP. You can edit them in any text editor, the easiest of which is Notepad in Windows. AutoCAD has a built-in editor for AutoLISP called the Visual LISP Editor (accessed through the VLISP command). For this handout, all the examples will be shown in Notepad.

Whereas a script file contains one or more commands and when the script file is run, all those commands execute, an AutoLISP file can have one or more functions defined in it, each with their own command string to run them. An AutoLISP file doesn't run as a whole, but rather holds all the functions in it you want to run, and you call any one of them from the command line.

To edit an AutoLISP text file, you will need to follow a particular format. For this handout and class, we're going to focus on recreating what you would type in at the command line in a LISP format. Building on our example from the previous sections, in AutoLISP it would look like:

```
(command "-layer" "make" "A-Wall" "color" "4" "" "ltype" "Continuous" "" "")
```

Some notes on this format:

- The entire line is wrapped in parenthesis – "open" or "left" parenthesis to start and "closed" or "right" parenthesis to end
- The first word is `command` which denotes that the next word will be a command you can type in at the command line. In this example, it is followed by the non-dialog box version of the `-layer` command (denoted by the "-" in front of it)
- After the command starts, everything in quotes is what gets entered in the command line just as if you typed it. That includes command options (such as `"make"`, `"color"`, and `"ltype"`) as well as inputs (such as `"A-Wall"`, `"4"`, and `"Continuous"`).
- You may also notice the sets of quotes with nothing in them `""`. This denotes an ENTER. The is no need to put a space between those quotes.
- The amount of spaces between the quoted strings does not matter and you can use that to keep your files clean and readable. For example, if you were going to create several layers, you may format it this way:

```
(command "-layer" "make" "A-Wall"      "color" "4" "" "")
(command "-layer" "make" "A-Wall-Patt" "color" "8" "" "")
(command "-layer" "make" "A-Wall-D"    "color" "1" "" "")
```

One of the great things about the AutoLISP format is that it is very easy to see exactly what is being typed in at the command line, including the ENTER's. Also, you can run this format in your script files (from the previous section), which is what I do to help maintain a clean look.

How to Use

AutoLISP files need to be loaded in AutoCAD. There are several ways to do this:

1. Using the command line by typing `(load "MyLISPfunction.lsp")`.
2. Using the APPLOAD command to either do it each time you need it, or by adding it to the Startup Suite.
3. Using one AutoLISP file to load another one (or a series of other ones). For example, you can setup your AutoCAD to automatically load one AutoLISP file whose sole function is to load all your other ones. That way, you only need to manage one.
4. Use some of the built-in functionality in AutoCAD to have the AutoLISP file load for you, as you will learn in the next section.

Deeper Dive

Let's dig in a little more on using AutoLISP to help you speed up your AutoCAD.

Basics of Defining a Function

We've already seen how you can write the commands in an AutoLISP format, so now we want to bundle one or more commands in their own custom command. The example we're going to use is one I've been using for going on two decades. I love to zoom my drawings to their extents before saving and exiting. But, when I zoom to extents, it makes it hard to window around everything. Also, even the shortcuts for zooming to extents requires four keys to be typed: "Z" "SPACE" "E" "SPACE". Because of this, we're going to build my most used zoom function:

Zoom to the extents of the drawing
Zoom out 10%
Do so with only typing in "ZZ" and then "SPACE"

You've already learned how to do the first two parts. The code would look like this:

```
  (command "zoom" "extents")
  (command "zoom" ".9x")
```

Now, we need to bundle these two commands together so we can simply type "ZZ" to run them. We do this by using the very basic method in AutoLISP of "defining a function". The format looks like this:

```
(defun c:ZZ ()
  (command "zoom" "extents")
  (command "zoom" ".9x")
)
```

Some notes on this format:

- Notice we added a line before our commands and then another afterwards, essentially wrapping up all the commands into the one function.
- As you probably guessed, "`defun`" means to define a function.
- The `c:` is simply saying that this will be a full, global function. You will always use this right before what you want to type in at the command line to run your new AutoLISP function. In this example, `ZZ`.
- The empty parenthesis `()` at the end of the first line are where variables can be defined. We are not doing this for this handout and class, so you will also include empty parenthesis in your functions.

- The very end line is simply a “closed” parenthesis `)`, which is what compliments the very first “open” parenthesis `(` before the word `defun`. AutoLISP functions will not work without equal open and close parenthesis and this is the first thing you should check for when a function doesn’t work.

- You don’t have to span your functions across multiple lines. This format will function exactly the same:
```
(defun c:ZZ ()(command "zoom" "extents")(command "zoom" ".9x"))
```

- The extra spaces before the command lines are not necessary. They simply help format the text to make it easier to read and are a best practice. Typically, people use two spaces, but some use a TAB instead (I like two spaces).

Acad.lsp

For this class, we are going to only work in one particular AutoLISP file. Instead of creating a file and naming it what we want, we are going to take advantage of some built-in functionality in AutoCAD. Every installation of AutoCAD looks for a file called Acad.lsp, even though it is not installed. This means that we can add our own Acad.lsp file, place our AutoLISP functions in it, and AutoCAD will load it for us ensuring all our functions are available. There is more on where to place this file in the section titled “Setting It Up”.

Tips / Best Practices

The next section provides many tips and best practices when using AutoLISP files and, in particular, your Acad.lsp file.

Semicolons

Just as with PGP files, a semicolon is a way to place a remark. Any text written after a semicolon is ignored, so this is how you can add notes to your lines of text. For example:

```
(command "zoom" "extents")     ;Zooms to the extents
```

It is always good practice to add lots of these notes to your AutoLISP files so you can retrace your steps and understand what you wrote. **You can't have too many of these!** You will see good examples of this in the class Acad.lsp file provided.

Testing Via the Command Line

Always make sure to test your AutoLISP functions at the command line. You can type in AutoLISP directly in the command line by simply starting with an open parenthesis `(` and AutoCAD will keep waiting for more of your "code" until you put in a close parenthesis `)`.

Also, make sure you are stepping through all the options in the commands you are using. Some commands change their options depending on what options you choose. The -PLOT command is notorious for this and can be very difficult to put into static AutoLISP functions.

Testing in Chunks

If you have an AutoLISP function that you built that you just can't seem to get to work, test it in chunks. You can do this by either:

1. Copying and pasting parts of your code into the command line to make sure they work as expected

2. Add semicolons to remark out some lines of your code, slowly removing them to narrow down which line has the problem

Case

AutoLISP is not case sensitive – you can use either upper or lower case whenever you want. Because of this, it is a best practice to use one or the other consistently, depending on what that piece of text represents. I tend to use lower case for everything except the function name and the values being input. For example, this function creates my standard revision layers:

```
(defun c:REVLAYERS ()
  (command "-layer" "make" "A-ANNO-REV1" "ltype" "CONTINUOUS" "" "")
  (command "-layer" "make" "A-ANNO-REV1" "ltype" "CONTINUOUS" "" "")
  (command "-layer" "make" "A-ANNO-REV1" "ltype" "CONTINUOUS" "" "")
)
```

Periods

It is possible with AutoLISP to actually "undefine" standard AutoCAD commands so you can "redefine" them the way you want. A practical example of this involves the LAYER command where, when it switched to a palette, some of my colleagues were undefining that command and creating a new function called "Layer" that launched the classic layer dialog box. The point is that you can avoid having commands in your Acad.lsp file that have been undefined and no longer function by adding a period to the beginning of them. For example:

```
(command ".-layer" "make" "A-Wall" "color" "4" "" "")
```

This period can be coupled with the dash/hyphen for the command line version, but the dash must be just before the command name.

Underscores

Along the same lines as the period, you can force the English language versions of commands by adding an underscore. For example:

```
(command "_.-layer" "make" "A-Wall" "color" "4" "" "")
```

Again, this can be coupled with the period and the dash/hyphen and it doesn't matter what order the period and the underscore are in, just that the dash/hyphen is next to the command.

Pause

If you are using a command that requires someone to pick a point, add the `pause` text in your function. For example.

```
(command "._break" pause "first" pause "@")
```

S::STARTUP

This string has a special function in the Acad.lsp file. When placed at the end of the file, instead of the standard `(defun c:??? ()` opening line, everything defined in it will run every time the Acad.lsp is loaded without having to type a command.

This can be very useful when you are using your Acad.lsp to set drawing variables. You can use it in one particular way, as an indicator that your Acad.lsp file has loaded properly. You can do so by adding the following to the end of your Acad.lsp file:

```
(defun s::startup ()
  (princ "\n\n...My Custom AutoLISP functions loaded.\n\n")
  (princ)
)
```

A couple points on this:

- The `(princ)` function sends text to the command line (or a blank line)
- The `\n` is just a line break

Other Commands/Functions of Which to be Aware

Again, this is **NOT** a coding class. But, if you are interested in going the next step, here is a list of some of the AutoLISP commands you should definitely investigate.

- SETVAR / GETVAR
  Sets or reads a drawing variable

- SETQ
  Sets a variable to a value; the value can be through user input or by grabbing a drawing variable

- PROMPT
  To show text on the command line

- Mathematical functions (+ - * /)
  Format is function symbol first, then numbers; for example `(- 4 1)` returns `3` as the result

- ALERT
  A dialog box with just an “OK” button

- WHILE
  Keeps looping through commands until everything is done

- USERI1, USERS1, USERR1
  Variables hidden in every DWG file that can save data so you can access it

- DXF Codes (used with CDR, CADR, ASSOC)
  this is how you dig into an entity to see its information such as what type of entity it is, what layer it’s on, scale, rotation, etc.

- STRCAT
  Use to concatenate a string of multiple pieces of information together

- Error handling
  Make sure you add some error handling in case someone uses ESC in the middle of your function to reset everything back.

Setting It Up

You only need to take a few simple steps to be able to use the custom files you've created after following this handout. You will be able to take these files with you wherever you go and quickly get them up and running.

Folders and Files

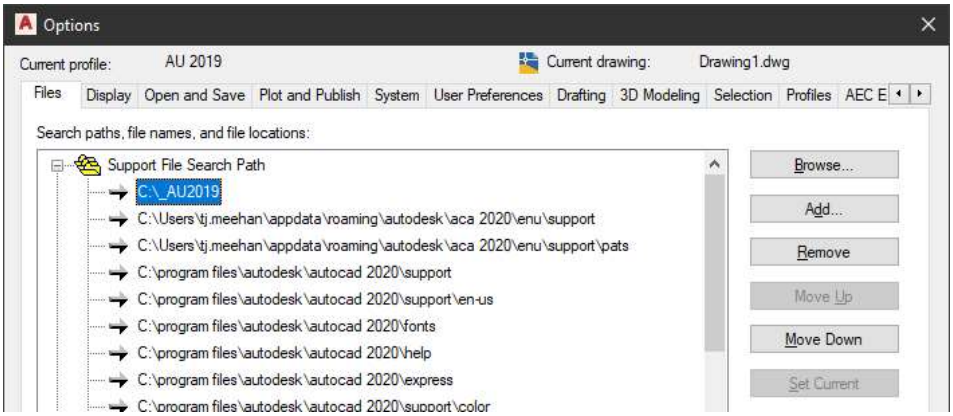
By this point, you should have your own custom files that you have created:

- One Acad.pgp file
- AutoCAD Script files
- One Acad.lsp file

The first step is to put all these files into a folder. **Make sure this folder is easily accessible and is backed up**.

Settings in Your AutoCAD OPTIONS

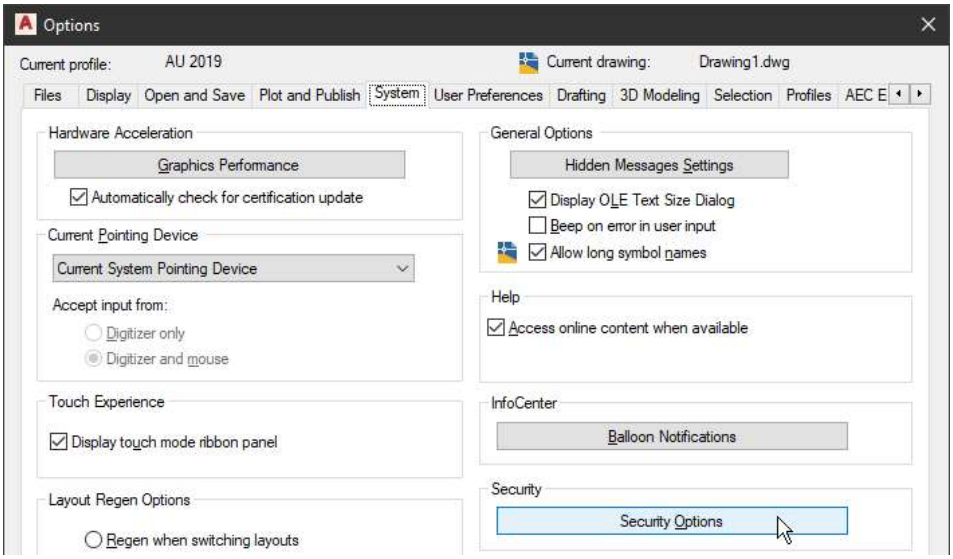
In order for your files to be found and available in AutoCAD, they must be in a folder that is part of the AutoCAD Support File Search Path. There is very easy to do. Once you go into your AutoCAD OPTIONS, it's the very first node on the very first tab in the dialog.



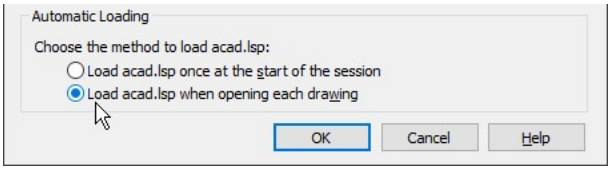
Simply use the Add… button and then Browse… to navigate to your folder. It will end up at the bottom of the list, so use the Move Up button to move it all the way to the top, as AutoCAD looks in the folders at the top of this list first.

Acad.lsp Settings

The last step you need to take is to make sure your Acad.lsp file is loading with every drawing. This is done from the “System” tab in your OPTIONS. From there, click on the “Security Options” button.



At the bottom of the dialog, select the “Load acad.lsp when opening each drawing” option.



Examples

Below are many examples of AutoLISP functions you can use. These are also included in the sample Acad.lsp included as part of the downloadable class files.

Offsets

As an architect, I have used the OFFSET command in AutoCAD a lot. During one stint working on wood framed houses, I continually offset stud distances of 3 1/2", 5 5/8", etc. I decided to speed that up by removing the initial launch of the OFFSET command and all the punctation, boiling it down to just the numerical values. It was a huge timesaver for me. Here is my OFFSET list I built from that experience.

These OFFSET by an increment of 1:

```
(defun c:1  ()(command "._offset" "1"))
(defun c:2  ()(command "._offset" "2"))
(defun c:3  ()(command "._offset" "3"))
(defun c:4  ()(command "._offset" "4"))
(defun c:5  ()(command "._offset" "5"))
(defun c:6  ()(command "._offset" "6"))
(defun c:7  ()(command "._offset" "7"))
(defun c:8  ()(command "._offset" "8"))
(defun c:9  ()(command "._offset" "9"))
(defun c:10 ()(command "._offset" "10"))
(defun c:11 ()(command "._offset" "11"))
(defun c:12 ()(command "._offset" "12"))
(defun c:13 ()(command "._offset" "13"))
(defun c:14 ()(command "._offset" "14"))
(defun c:15 ()(command "._offset" "15"))
(defun c:16 ()(command "._offset" "16"))
(defun c:17 ()(command "._offset" "17"))
(defun c:18 ()(command "._offset" "18"))
(defun c:19 ()(command "._offset" "19"))
(defun c:20 ()(command "._offset" "20"))
(defun c:21 ()(command "._offset" "21"))
(defun c:22 ()(command "._offset" "22"))
(defun c:23 ()(command "._offset" "23"))
(defun c:24 ()(command "._offset" "24"))
(defun c:25 ()(command "._offset" "25"))
(defun c:26 ()(command "._offset" "26"))
(defun c:27 ()(command "._offset" "27"))
(defun c:28 ()(command "._offset" "28"))
(defun c:29 ()(command "._offset" "29"))
(defun c:30 ()(command "._offset" "30"))
```

These OFFSET by an increment of 2:

```
(defun c:32 ()(command "._offset" "32"))
(defun c:34 ()(command "._offset" "34"))
(defun c:36 ()(command "._offset" "36"))
(defun c:38 ()(command "._offset" "38"))
(defun c:40 ()(command "._offset" "40"))
(defun c:42 ()(command "._offset" "42"))
(defun c:44 ()(command "._offset" "44"))
(defun c:46 ()(command "._offset" "46"))
(defun c:48 ()(command "._offset" "48"))
(defun c:50 ()(command "._offset" "50"))
(defun c:52 ()(command "._offset" "52"))
(defun c:54 ()(command "._offset" "54"))
(defun c:56 ()(command "._offset" "56"))
(defun c:58 ()(command "._offset" "58"))
(defun c:60 ()(command "._offset" "60"))
(defun c:62 ()(command "._offset" "62"))
(defun c:64 ()(command "._offset" "64"))
(defun c:66 ()(command "._offset" "66"))
(defun c:68 ()(command "._offset" "68"))
(defun c:70 ()(command "._offset" "70"))
```

These OFFSET by an increment of 12 (1 foot):

```
(defun c:72  ()(command "._offset" "72"))
(defun c:84  ()(command "._offset" "84"))
(defun c:96  ()(command "._offset" "96"))
(defun c:108 ()(command "._offset" "108"))
(defun c:120 ()(command "._offset" "120"))
(defun c:132 ()(command "._offset" "132"))
(defun c:144 ()(command "._offset" "144"))
```

These OFFSET the standard stud sizes:

```
(defun c:158 ()(command "._offset" "1-5/8"))
(defun c:35  ()(command "._offset" "3-1/2"))
(defun c:358 ()(command "._offset" "3-5/8"))
(defun c:55  ()(command "._offset" "5-1/2"))
(defun c:558 ()(command "._offset" "5-5/8"))
```

Zoom

A collection of my favorite zoom commands. Some are simply shortcutting to the zoom command's options, some are based on scale, and of course my all-time favorite, ZZ.

ZA ZOOM all

```
(defun c:ZA ()
  (command "._zoom" "all")
)
```

ZE ZOOM extents

```
(defun c:ZE ()
  (command "._zoom" "extents")
)
```

ZI ZOOM in 10%

```
(defun c:ZI ()
  (command "._zoom" "1.1x")
)
```

ZO ZOOM out 10%

```
(defun c:ZO ()
  (command "._zoom" ".9x")
)
```

ZP ZOOM previous

```
(defun c:ZP ()
  (command "._zoom" "previous")
)
```

ZW ZOOM window

```
(defun c:ZW ()
  (command "._zoom" "window")
)
```

Z2 ZOOM to 1/2" scale

```
(defun c:Z2 ()
  (command "._zoom" "1/24XP")
)
```

Z4 ZOOM to 1/4" scale

```
(defun c:Z4 ()
  (command "._zoom" "1/48XP")
)
```

Z8 ZOOM to 1/8" scale

```
(defun c:Z8 ()
  (command "._zoom" "1/96XP")
)
```

Z16 ZOOM to 1/16" scale

```
(defun c:Z16 ()
  (command "._zoom" "1/192XP")
)
```

Z25 ZOOM to 1:25 scale

```
(defun c:Z25 ()
  (command "._zoom" "1/25XP")
)
```

Z50 ZOOM to 1:50 scale

```
(defun c:Z50 ()
  (command "._zoom" "1/50XP")
)
```

Z100 ZOOM to 1:100 scale

```
(defun c:Z100 ()
  (command "._zoom" "1/100XP")
)
```

Z200 ZOOM to 1:200 scale

```
(defun c:Z200 ()
  (command "._zoom" "1/200XP")
)
```

ZZ ZOOM extents then out 10%

```
(defun c:ZZ ()
  (command "._zoom" "extents")
  (command "._zoom" ".9x")
)
```

Views

There are three different groups of AutoLISP functions related to views. One group is to jump to a view angle, another is to rotate the view, and the third is to build one or two viewports.

A series of commands to let you change your **View Direction** by typing in the option:

```
(defun C:TOP   ()(command "._-view" "top"))
(defun C:FRONT ()(command "._-view" "front"))
(defun C:LEFT  ()(command "._-view" "left"))
(defun C:RIGHT ()(command "._-view" "right"))
(defun C:BACK  ()(command "._-view" "back"))
(defun C:SW    ()(command "._-view" "swiso"))
(defun C:SE    ()(command "._-view" "seiso"))
(defun C:NW    ()(command "._-view" "nwiso"))
(defun C:NE    ()(command "._-view" "neiso"))
```

A series of commands to you change your **View Rotation** by typing in the angle:

```
(defun c:RV0   ()(command "ucs" "world")(command "plan" "world"))
(defun c:RV45  ()(command "ucs" "new" "z" "45" )(command "plan" "c"))
(defun c:RV90  ()(command "ucs" "new" "z" "90" )(command "plan" "c"))
(defun c:RV135 ()(command "ucs" "new" "z" "135")(command "plan" "c"))
(defun c:RV180 ()(command "ucs" "new" "z" "180")(command "plan" "c"))
(defun c:RV225 ()(command "ucs" "new" "z" "225")(command "plan" "c"))
(defun c:RV270 ()(command "ucs" "new" "z" "270")(command "plan" "c"))
(defun c:RV315 ()(command "ucs" "new" "z" "315")(command "plan" "c"))
```

1V Restores one viewport

```
(defun c:1V ()
  (command "-vports" "single")
)
```

2V Creates two tiled viewports

```
(defun c:2V ()
  (command "-vports" "2" "vertical")
  (command "-view" "swiso")
  (setvar "CVPORT" 2)
  (command "zoom" "extents")
  (setvar "CVPORT" 3)
  (command "zoom" "extents")
)
```

Layers

It's surprising to me that these functions don't already exist in AutoCAD.

LOA Layer on all

```
(defun C:LOA ()
  (command "._-layer" "on" "*" "")
)
```

LTA Layer thaw all

```
(defun C:LTA ()
  (command "._-layer" "thaw" "*" "")
)
```

Productivity

A few that have helped me over the years be more productive in AutoCAD. There is one function, =, that does not follow the rules of this handout in that it uses some AutoLISP code instead of just command line functions. But, it's such a great utility, I threw it in. Please try it out for yourself as coupling it with ORTHO is one of the best tools for drawing consistent angles.

BA BREAK an entity at a point

```
(defun C:BA ()
  (command "._break" pause "first" pause "@")
)
```

CP COPY previous

```
(defun c:CP ()
  (command "._copy" "p" "")
)
```

MP MOVE previous

```
(defun c:MP ()
  (command "._move" "p" "")
)
```

FR Set the FILLET radius

```
(defun C:FR ()
  (command "._fillet" "radius")
)
```

F0 Set the FILLET radius to 0

```
(defun C:FR ()
  (command "._fillet" "radius" "0")
)
```

0 Sets SNAPANG back to 0

```
(defun c:0 ()
  (setvar "SNAPANG" 0)
)
```

= Sets SNAPANG selected element

```
(defun C:= (/ EL PT1 PT2)
  (setq EL (entget (car (entsel))))
  (setq PT1 (cdr (assoc 10 EL)))
  (setq PT2 (cdr (assoc 11 EL)))
  (setvar "SNAPANG" (angle PT1 PT2)))
)
```

File Maintenance

These last few help you keep your files clean and running smoothly.

AU AUDIT the file

```
(defun c:AU ()
  (command "._audit" "yes")
)
```

BB Sets all entities to “byblock”, which is great when creating block libraries

```
(defun c:BB ()
  (command "._change" "all" "" "properties" "color" "ByBlock" "ltype"
"ByBlock" "lweight" "ByBlock" "")
  (command "._regenall")
)
```

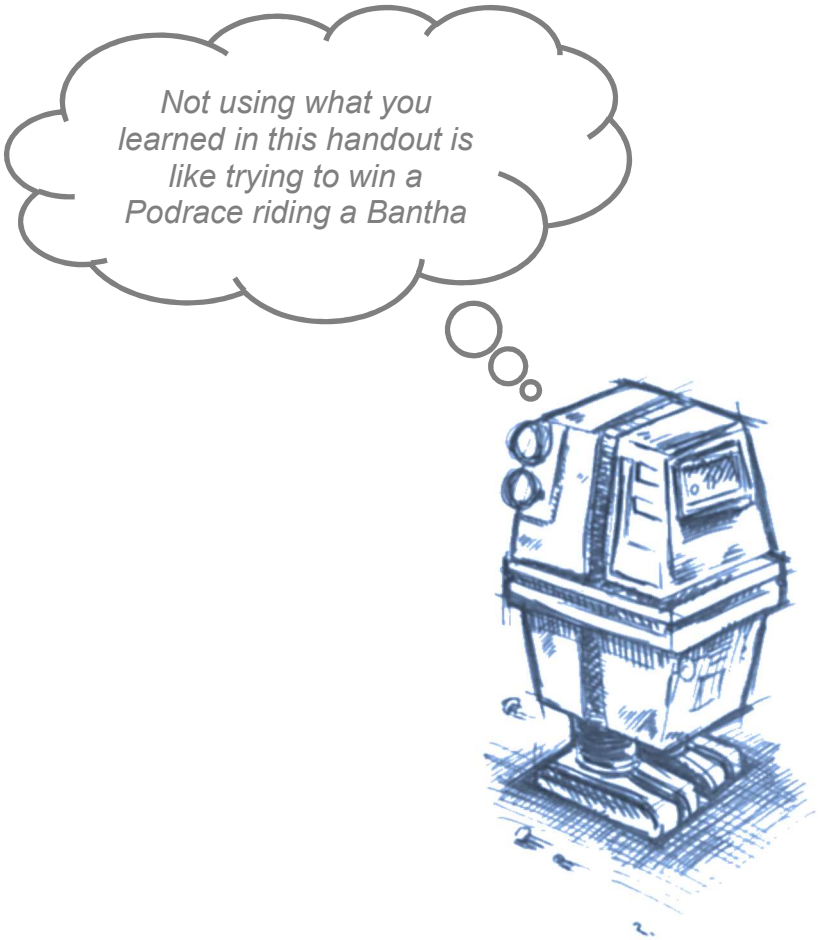
PG PURGE all 4 times

```
(defun c:PG ()
  (command "._-purge" "all" "*" "no")
  (command "._-purge" "regapps" "*" "no")
  (command "._-purge" "zero")
  (command "._-purge" "empty")
  (command "._-purge" "all" "*" "no")
  (command "._-purge" "regapps" "*" "no")
  (command "._-purge" "zero")
  (command "._-purge" "empty")
  (command "._-purge" "all" "*" "no")
  (command "._-purge" "regapps" "*" "no")
  (command "._-purge" "zero")
  (command "._-purge" "empty")
  (command "._-purge" "all" "*" "no")
  (command "._-purge" "regapps" "*" "no")
  (command "._-purge" "zero")
  (command "._-purge" "empty")
)
```

Conclusion

You now have all the tools and information you need to increase your speed and efficiency while working on AutoCAD. Our droid friend below is summing it up well. As mentioned, if you ever have any questions or would like to provide feedback, please don't hesitate to reach out to me at tj.meehan@caddmicrosystems.com.

Good luck to you and happy AutoCADing!



DRAWING BY JASON KUNKEL
A SUPER TALENTED ARTIST
FIND HIM ON TWITTER @RVIT