The Many Ways of Using Scripts and AutoLISP® to Automate, Save Time, and Enforce Standards in AutoCAD®

michael trachtenberg – AEBACK OFFICE

CM2323

In this class you will learn how to effectively use some of the strongest AutoCAD tools, such as AutoLISP routines and scripts, to rid yourself of repetitive tasks. This class will also cover effective ways to automate and enforce almost all of your company's standards by showing you how to use automation tools in AutoCAD menus, at startup, for customization, and as stand-alone programs and routines. Learning these techniques will prove to be invaluable and save more time and money than anyone could believe. When seconds is just too long, see how you can use these methods to set up a pagesetup, print a PDF, or bind a file with just one click. Attendees should have some knowledge about AutoLISP and AutoCAD scripts

At the end of this class, you will be able to:

- Rid yourself of simple time consuming repetitive tasks
- Implement routines/programs a number of different ways
- Silently enforce standards
- Draft faster while enforcing more standards
- Be lazier thanks to your new toys

About the Speaker

michael trachtenberg is a Global CAD\BIM managing consultant and has been optimizing AutoCAD®, Revit MEP®, and many other design products in the AEC industry for roughly 10 years. With a history in programming & engineering he continually increases productivity with the creation of custom programs and routines that make Architects, Designers, and Engineers lives' much easier. All while focusing on standardization, design technology management, training, and providing personal design support. Michael has worked on and managed technology use for some of the largest construction projects in the US and abroad, including the Chicago Spire, the Shanghai Tower, the Devon Energy Headquarters, the Al Ain Wild life Resort and the WTC redevelopment project.

mtrachtenberg@hlw.com

The many ways of running Scripts

Using AutoCAD scripts is a great way to string together AutoCAD commands or apply settings using a minimal amount of user input. Scripts enter commands into AutoCAD just as they would be entered by a user at the command line. Scripts are easiest to make using a simple text editor such as notepad or notepad++ . Script files have a .scr file extension

- AutoCAD command SCRIPT
  Using the script command in AutoCAD will bring up a dialog box for you to browse to your script, once selected and opened, your script will run.
  This is the slowest method of running a script

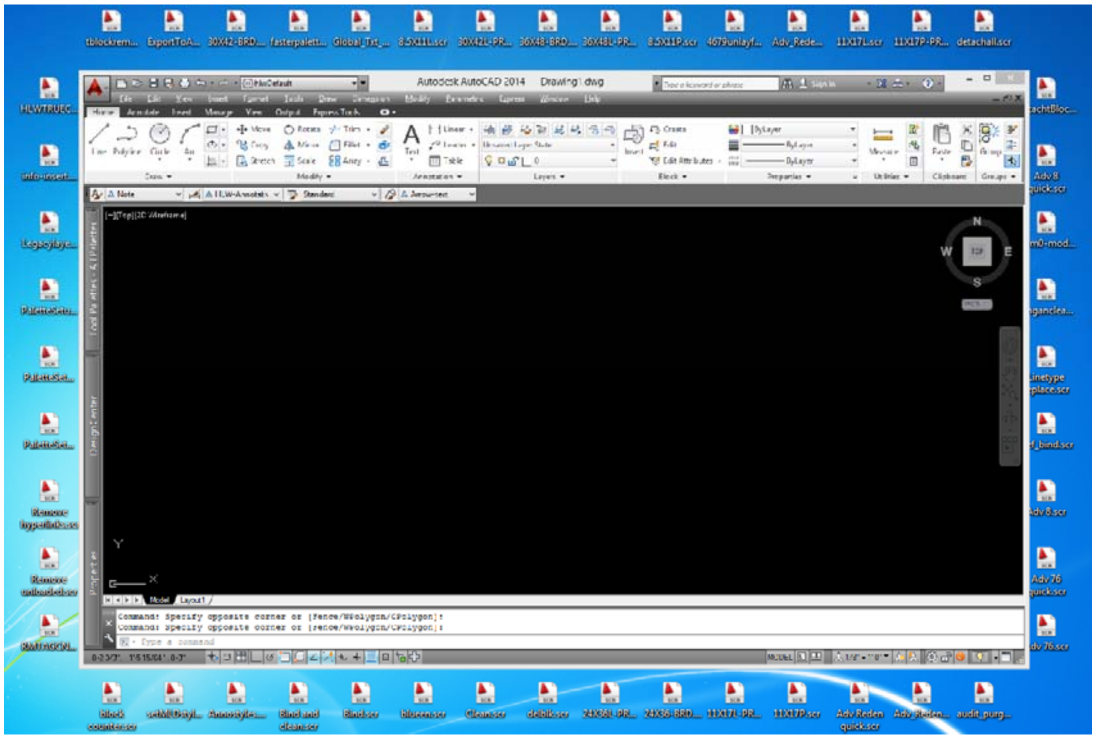
- AutoCAD menu

  Button

  Ribbon: Manage tab ➤ Applications panel ➤ Run Script

  Menu: Tools ➤ Run Script

- Drag and Drop
  .SCR files can be dragged and dropped into an open AutoCAD file. This is a perfect method for high level production drafters. You can quickly apply numerous custom routines without having extra or specific menus open.



```
Command: Specify opposite corner or [Fence/WPolygon/CPolygon]:
Command: Specify opposite corner or [Fence/WPolygon/CPolygon]:
Type a command
```

- Tool Palettes
  Scripts can be ran from the tool palettes as well.
  It is easiest to copy a tool palette button using the right-click menu and pasting the button onto the palette of your choice.

  You can go to the properties of any tool palette button using the same right-click process and select "Properties"

  Under the "Command String" field use the following format to run a script

  ^c^cSCRIPT "[PATH]/[FILENAME.scr]"

  ^c^cSCRIPT "C:/Scripts/BIND.scr"

By lining up commonly used scripts in a single palette of scripts that can be used one after another, you can turn 20 tasks into 3 clicks

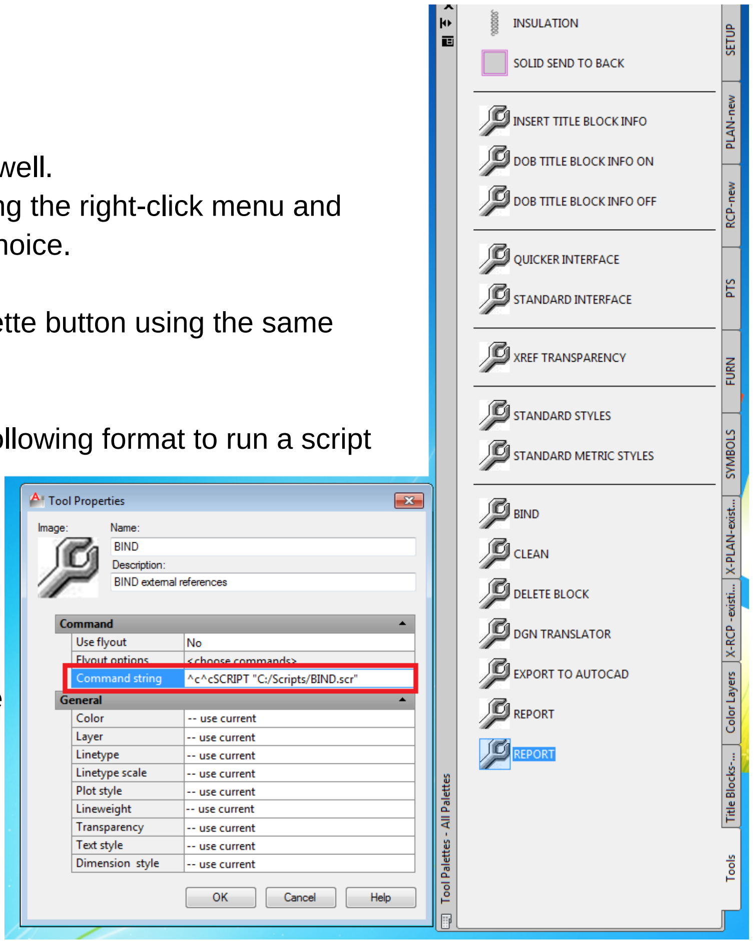


Image: Name: BIND
Description: BIND external references

Command	
Use flyout	No
Flyout options	<choose commands>
Command string	^c^cSCRIPT "C:/Scripts/BIND.scr"

General	
Color	-- use current
Layer	-- use current
Linetype	-- use current
Linetype scale	-- use current
Plot style	-- use current
Lineweight	-- use current
Transparency	-- use current
Text style	-- use current
Dimension style	-- use current

- Menu & Ribbon
  Through use of CUI manipulation, you can add scripts to the AutoCAD menus and ribbon

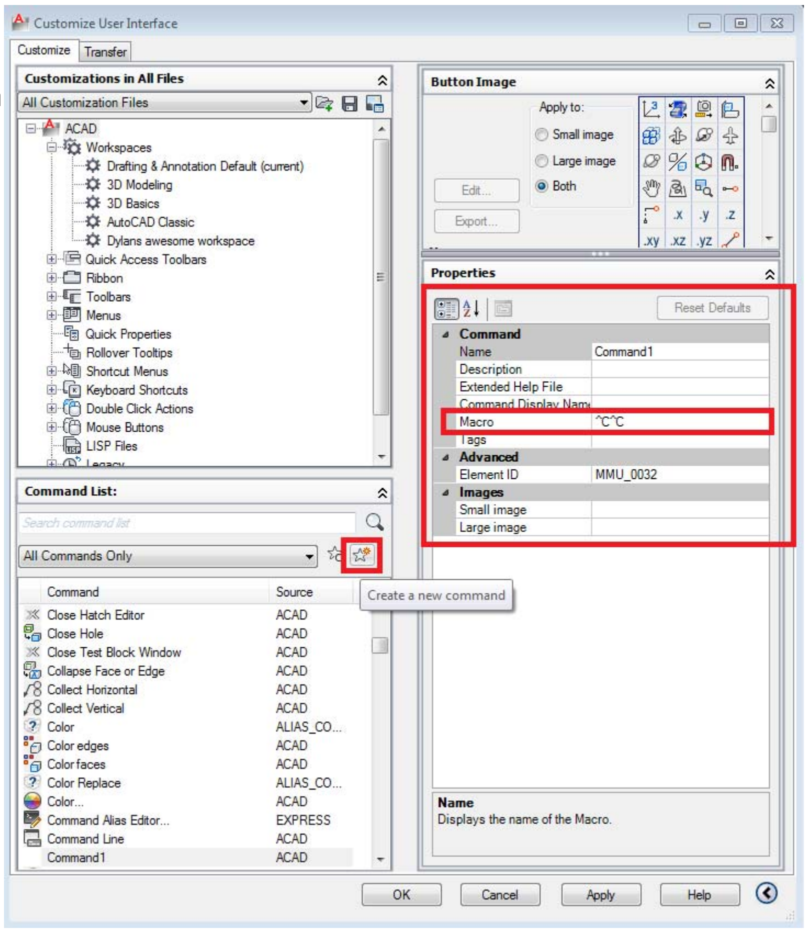
  From within the CUI menu you can create a new command by selecting the "Create a new command" icon

  Then in the command properties on the right enter the name and info of the command you want to make

  Under the "Macro" property, point to your script using the same syntax as you would for a tool palette button

  ^c^cSCRIPT "[PATH]/[FILENAME.scr]"

  ^c^cSCRIPT "C:/Scripts/BIND.scr"



Command	Source
Close Hatch Editor	ACAD
Close Hole	ACAD
Close Text Block Window	ACAD
Collapse Face or Edge	ACAD
Collect Horizontal	ACAD
Collect Vertical	ACAD
Color	ALIAS_CO...
Color edges	ACAD
Color faces	ACAD
Color Replace	ALIAS_CO...
Color...	ACAD
Command Alias Editor...	EXPRESS
Command Line	ACAD
Command1	ACAD

Properties

Command	
Name	Command1
Description	
Extended Help File	
Command Display Name	
Macro	^C^C
Tags	
Advanced	
Element ID	MMU_0032
Images	
Small image	
Large image	

Name
Displays the name of the Macro.

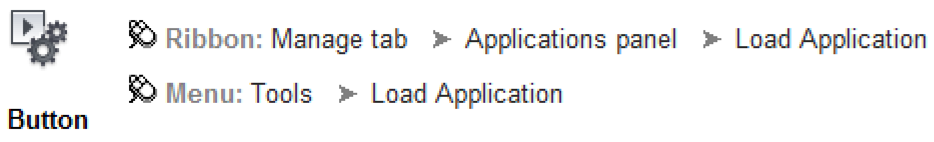
- Scriptpro 2.0
  Autodesk labs provides an easy to use tool for running the same script on multiple drawings. With Scriptpro 2.0 there is need to open, edit, save and close each drawing yourself

  http://labs.autodesk.com/utilities/ADN_plugins/catalog/
  http://labs.autodesk.com/utilities/scriptpro/

The many ways of loading Lisps

Lisps are much more diverse and are extremely more robust that script files. Lisps can change any setting in AutoCAD with or without user input and almost anything you would need to automate can be done with a lisp. Lisps are limited to being ran on the current active drawing. Lisps are easiest to make using a simple text editor such as notepad or notepad++ . lisp files have .lsp, .fas, .vlx file extensions

- AutoCAD command APPLOAD
  Using the appload command you can browse to and load any lisp file. This works the same as the SCRIPT command

- AutoCAD menu 

  Button

  Ribbon: Manage tab ➤ Applications panel ➤ Load Application

  Menu: Tools ➤ Load Application

- Drag and Drop
  Dragging and dropping lisps works the same way as it does for script files

- Tool Palettes
  Only the syntax is different between loading lisps & apply a script from a the tool palette

  ^c^c(LOAD "[PATH]/[FILENAME.lsp]")

  ^c^c(LOAD "C:/Lisps/xtrans.lsp")

- Menu & Ribbon
  Again the only difference is the syntax under the Macro properties

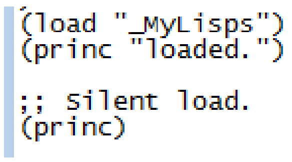
  ^c^c(LOAD "[PATH]/[FILENAME.lsp]")

  ^c^c(LOAD "C:/Lisps/xtrans.lsp")

- Startup
  AutoCAD itself uses a number of lisp files during its startup process. You can add your own lisps to this start-up process. Doing so will enable you to load custom commands, apply any and all custom settings and enforce better standards.

Under your AutoCAD support directory locate your "acadYYYYdoc.lsp" file. You can simply add all of your custom lisp files to this start up lisp, or even better, load a single lisp file that will then reference all of your other lisps.

Example: at the end of the acad2012doc.lsp add (load "_MyLisps") Then under the same support location have a single lisp file named "_MyLisps.lsp" which sets all of your variables, loads all of your commands and everything else you want to accomplish using lisp



```
(load "_MyLisps")
(princ "loaded.")

;; Silent load.
(princ)
```

- Support Directories
  If a lisp file is located in any of your AutoCAD's support directories you can load it at the command line using the following syntax

  (load "[LispName]")
  (load "_MyLisps")

The many ways of running VB .NET

VB .NET developed routines can also be used by AutoCAD. These types of programs can become extremely complex and could do all of the same things lisps and scripts can do. These programs can be a standalone executable, they can be ran on any amount of files and are the most diverse while being the most complicated to write. Specific external API and SDK programming environments are needed to create these types of programs. These programs also require the most work to maintain since language, 32/64 bit and other factors all needed to be kept up to date as Autodesk products change.

For official AutoCAD development platform info check the following link

http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=1911627

For a free .NET editor see

http://www.icsharpcode.net/

- AutoCAD command NETLOAD
  Using the netload command you can browse to and load any .NET dll file. This works the same as the SCRIPT & APPLOAD commands

- AutoCAD menu & Ribbon
  ^c^cNETLOAD “C:/DotNet/Counter.dll”

- Tool Palettes
  ^c^cNETLOAD “C:/DotNet/Counter.dll”

- Registry entry
  .NET dlls can also be loaded from the windows system registry

Under your registry keys for your product go to:
(HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\{VERSION}\Applications\{FOLDERNAME})

Then create the following values:

-DESCRIPTION A value describing you program
-LOADCTRLS A DWORD (numeric) value describing the reasons for loading the app
-MANAGED Another DWORD that should be set to "1" for .NET modules
-LOADER A string value containing the path to the module

Example:

HKEY_CURRENT_USER\Software\Autodesk\AutoCAD\R18.2\ACAD-A001:409\Applications\MyDotNet]

"DESCRIPTION"=" test application"
"LOADCTRLS"=dword:0000000c
"MANAGED"=dword:00000001
"LOADER"="C:\\ DotNet\\ Counter.dll"

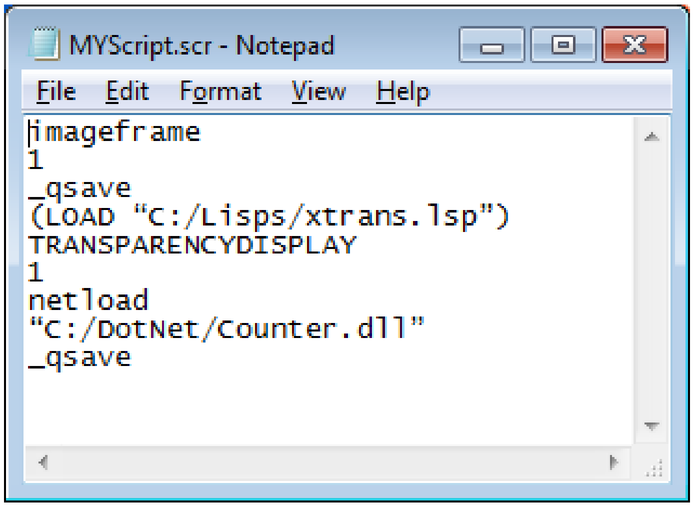
Conjunction & Tandem

Aside from just running a scripts, lisps or .NET separately, it is often more powerful to run them together. It is sometimes beneficial for one to start another or be part of a chain of actions. As an example you can use a script to run a number of lisp routines and load at .NET program all on 1 file. Or you may want to run a lisp on 15 files but scriptpro 2.0 only works for scripts, you have can than create a script that does nothing more than reference that lisp file.

- Scripts
  Lisp
  (LOAD “[PATH]/[FILENAME.lsp]”)
  Ex:
  (LOAD “C:/Lisps/xtrans.lsp”)

  .NET
  NETLOAD
  “[PATH]/[FILENAME.dll]”

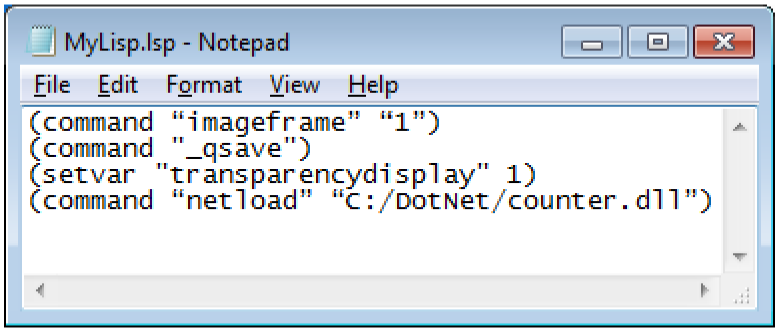
  Ex:
  NETLOAD
  “C:/DotNet/counter.dll”



```
imageframe
1
_qsave
(LOAD "C:/Lisps/xtrans.lsp")
TRANSPARENCYDISPLAY
1
netload
"C:/DotNet/Counter.dll"
_qsave
```

- Lisps
  Scripts
  There is no need for a lisp to call a script. A lisp can always do everything a script can do, simply use the command function
  (command “imageframe” 1)

  .NET
  (command “netload” “C:/DotNet/counter.dll”)



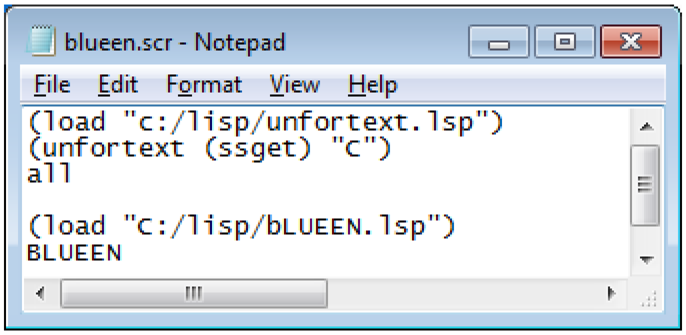
```
(command "imageframe" "1")
(command "_qsave")
(setvar "transparencydisplay" 1)
(command "netload" "C:/DotNet/counter.dll")
```

- .NET
  There is usually no need for .NET to run a script or a lisp file since .NET can do all of the functions of both.

  If needed you can use
  Scripts: (command “imageframe” 1)
  Lisps: (LOAD “C:/Lisps/xtrans.lsp”)

Many ways of bringing it all together

Using a script to run 2 different lisps and pass a value back to the first script. The two lisps turn all text in a file to a single standard and all lines or objects to another



```
(load "c:/lisp/unfortext.lsp")
(unfortext (ssget) "c")
all

(load "C:/lisp/bLUEEN.lsp")
BLUEEN
```

Using a lisp to make registry edits

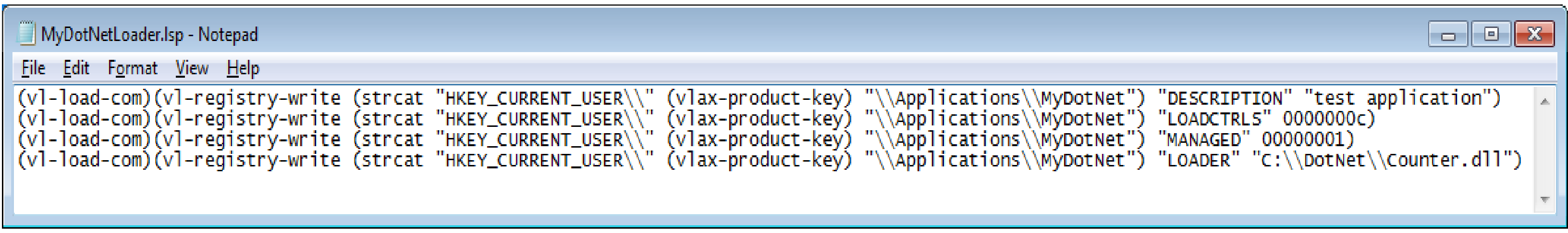
Example: Setting the bind type for a standard ETransmit setup

(vl-load-com)(vl-registry-write (strcat "HKEY_CURRENT_USER\\" (vlax-product-key) "\\ETransmit\\setups\\SendingOut") "BindType" 0)

Using lisps to change registry values regardless of AutoCAD version or platform, this code will never have to be updated or change for new versions of CAD and will work with all profiles.

(vl-load-com)(vl-registry-write (strcat "HKEY_CURRENT_USER\\" (vlax-product-key) "\\Profiles\\" (vla-get-activeprofile (vla-get-profiles (vla-get-preferences (vlax-get-acad-object)))) "\\Dialogs\\ModifyETransmitDialog") "DestinationFolder1" "C:\\CAD\\ SendingOut")

Using this type of registry edits with lisps, we can have a lisp routine load a .NET dll from startup



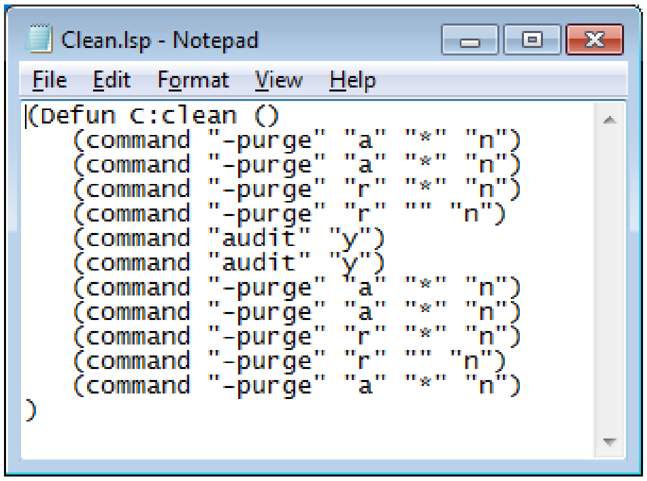
```
(vl-load-com)(vl-registry-write (strcat "HKEY_CURRENT_USER\\" (vlax-product-key) "\\Applications\\MyDotNet") "DESCRIPTION" "test application")
(vl-load-com)(vl-registry-write (strcat "HKEY_CURRENT_USER\\" (vlax-product-key) "\\Applications\\MyDotNet") "LOADCTRLS" 0000000c)
(vl-load-com)(vl-registry-write (strcat "HKEY_CURRENT_USER\\" (vlax-product-key) "\\Applications\\MyDotNet") "MANAGED" 00000001)
(vl-load-com)(vl-registry-write (strcat "HKEY_CURRENT_USER\\" (vlax-product-key) "\\Applications\\MyDotNet") "LOADER" "C:\\DotNet\\Counter.dll")
```

Add MyDotNetLoader.lsp to the “acadYYYYdoc.lsp” by putting (load “MyDotNetLoader”) in that file and you now can have your .NET program loaded on start-up.

Lisps to define a new command.

This example shows a
new command called "clean"

Add clean.lsp to the "acadYYYYdoc.lsp" by putting (load "Clean") in that file and you now can have your clean command loaded on start-up

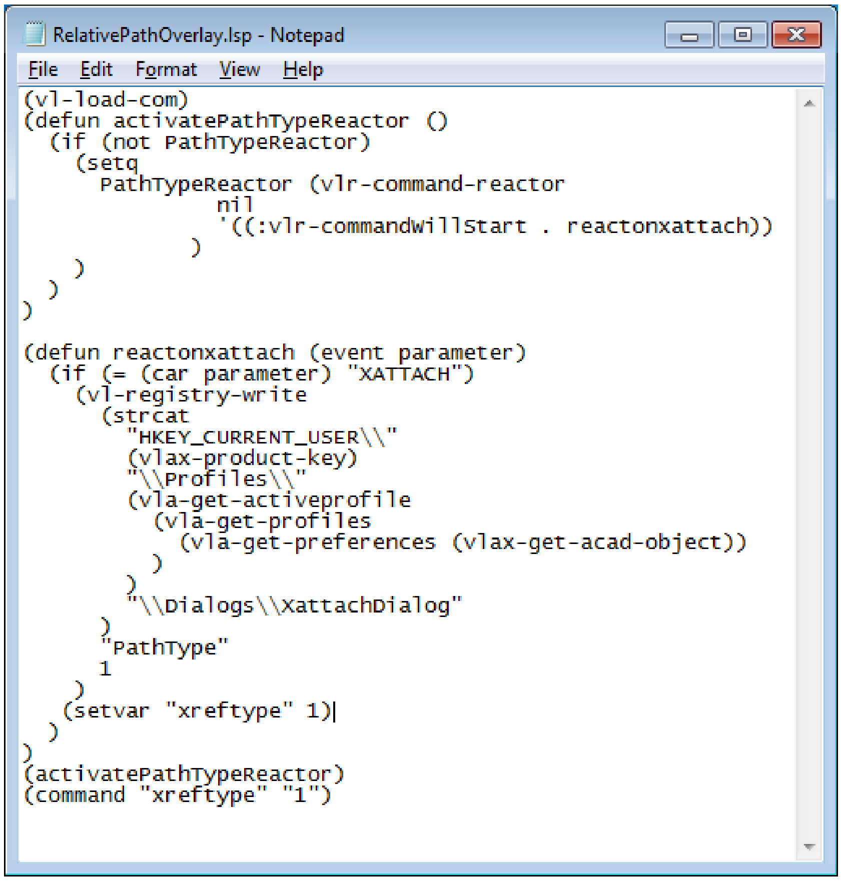


```
(Defun C:clean ()
    (command "-purge" "a" "*" "n")
    (command "-purge" "a" "*" "n")
    (command "-purge" "r" "*" "n")
    (command "-purge" "r" "" "n")
    (command "audit" "y")
    (command "audit" "y")
    (command "-purge" "a" "*" "n")
    (command "-purge" "a" "*" "n")
    (command "-purge" "r" "*" "n")
    (command "-purge" "r" "" "n")
    (command "-purge" "a" "*" "n")
)
```

Reaction Lisps

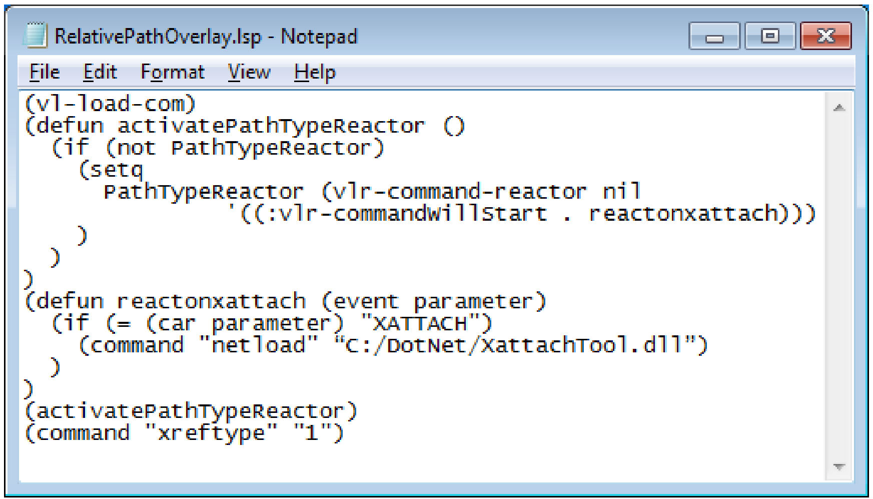
Reactors are very powerful tools in lisps and can help enforce standards silently.

This is an example of a reactor that when referencing a file the xref will be forced to be overlay and relative.



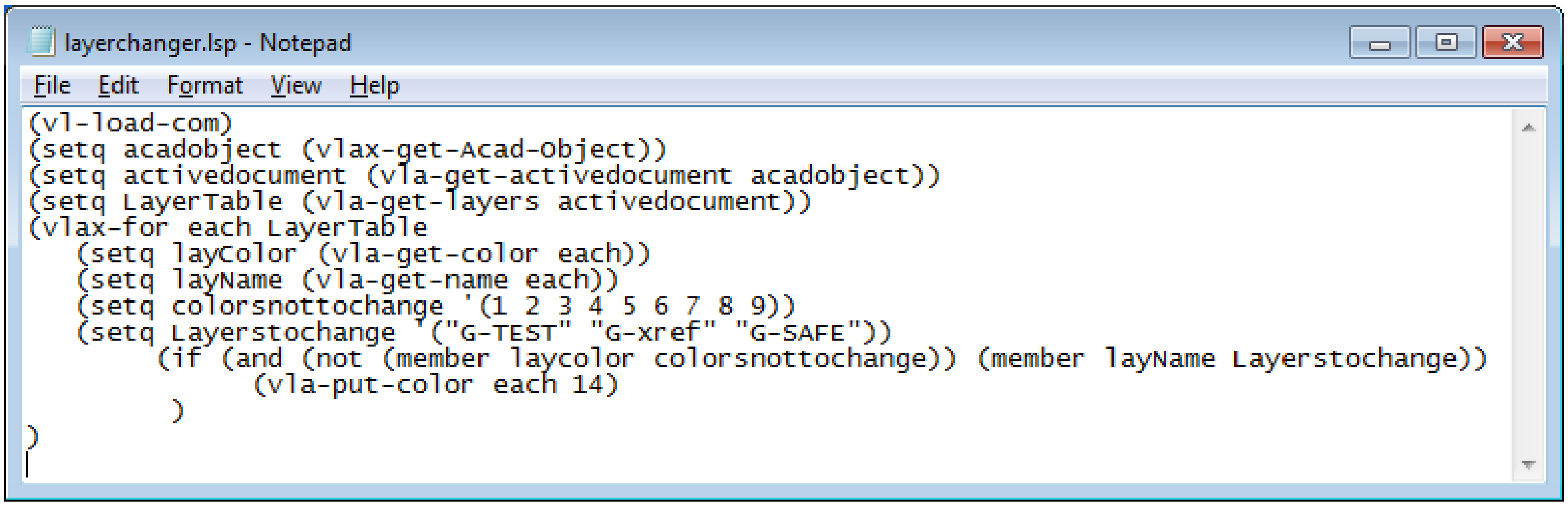
```
(vl-load-com)
(defun activatePathTypeReactor ()
  (if (not PathTypeReactor)
    (setq
      PathTypeReactor (vlr-command-reactor
                        nil
                        '((:vlr-commandWillStart . reactonxattach))
                      )
    )
  )
)

(defun reactonxattach (event parameter)
  (if (= (car parameter) "XATTACH")
    (vl-registry-write
      (strcat
        "HKEY_CURRENT_USER\\"
        (vlax-product-key)
        "\\Profiles\\"
        (vla-get-activeprofile
          (vla-get-profiles
            (vla-get-preferences (vlax-get-acad-object))
          )
        )
        "\\Dialogs\\XattachDialog"
      )
      "PathType"
      1
    )
    (setvar "xreftype" 1)
  )
)
(activatePathTypeReactor)
(command "xreftype" "1")
```

Instead of forcing just standards like shown above, you can have a reactor load a custom .NET application which a user can interact with if desired



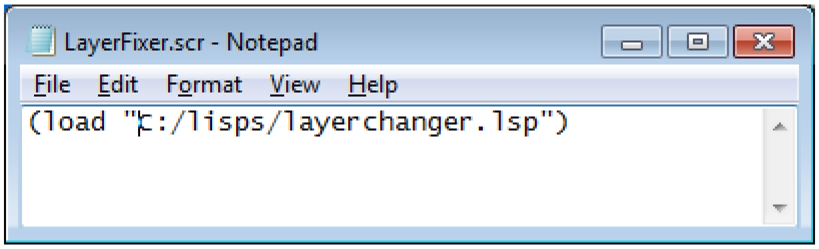
```
(vl-load-com)
(defun activatePathTypeReactor ()
  (if (not PathTypeReactor)
    (setq
      PathTypeReactor (vlr-command-reactor nil
                        '((:vlr-commandWillStart . reactonxattach)))
    )
  )
)
(defun reactonxattach (event parameter)
  (if (= (car parameter) "XATTACH")
    (command "netload" "C:/DotNet/XattachTool.dll")
  )
)
(activatePathTypeReactor)
(command "xreftype" "1")
```

This is a lisp to change any setting for any layer or groups of layers. You can alter the If statement in this code to get any result you desire.



```
(vl-load-com)
(setq acadobject (vlax-get-Acad-Object))
(setq activedocument (vla-get-activedocument acadobject))
(setq LayerTable (vla-get-layers activedocument))
(vlax-for each LayerTable
   (setq layColor (vla-get-color each))
   (setq layName (vla-get-name each))
   (setq colorsnottochange '(1 2 3 4 5 6 7 8 9))
   (setq Layerstochange '("G-TEST" "G-xref" "G-SAFE"))
        (if (and (not (member laycolor colorsnottochange)) (member layName Layerstochange))
              (vla-put-color each 14)
         )
)
```

You could then use the following script example



```
(load "c:/lisps/layerchanger.lsp")
```

And when paired with scriptpro 2.0 you could run this script on any number of drawings.

This is lisp could also be ran automatically when AutoCAD starts up and it will apply all of your designated layer settings to all of your files.