AUTODESK UNIVERSITY 2014

AutoLISP®: The 21 Frequently Overlooked and Forgotten Functions

Lee Ambrosius – Autodesk, Inc.

SD6751 The AutoLISP programming language has been around for decades, yet there are many functions that often go undiscovered or forgotten by veteran AutoLISP programmers. While there are hundreds of functions in the AutoLISP programming language (thousands, if you count ActiveX), this class will focus on 21 functions that new developers often overlook and that veteran programmers often forget. All of the functions fit into 1 of several different themes, including function execution, object creation and manipulation, and error handling. We will also discuss loading LISP files, accessing variable values, and configuring contextual help.

Learning Objectives

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

- Discover the AutoLISP programming language functions that might make development easier
- Discover the benefits to these functions
- Discover how to implement these functions
- Examine code samples for each function

About the Speaker

Lee is a Principal Learning Content Developer on the AutoCAD® team at Autodesk and has been an AutoCAD user for 18 years in the fields of architecture and facilities management. He has been teaching AutoCAD users for 15 years at both the corporate and college level. Lee is best known for his expertise in programming and customizing AutoCAD-based products, and has 15+ years of experience programming with AutoLISP®, VBA, Microsoft® .NET, and ObjectARX®. Lee has written articles for AUGI® publications and white papers for Autodesk on customization. He is the co-author of the AutoCAD and AutoCAD LT 2015 Bible and the author of the AutoCAD Platform Customization series.

Twitter: *http://twitter.com/leeambrosius*

Email: *lee.ambrosius@autodesk.com*

Blog: *http://hyperpics.blogs.com*

Contents

1 Introduction

The AutoLISP programming language has been around for nearly 3 decades and for much of that time, its core functionality has remained consistent with the exception of ActiveX/COM support being introduced with AutoCAD 2000-based products. While almost everything has remained fairly static, there are many functions that are left undiscovered or become forgotten for a number of reasons. AutoLISP as many have discovered allows for the manipulation of objects in a drawing, adding new objects, and automating simple and complex tasks. This session doesn't focus on why AutoLISP is important, as you already know it is in the work you do and is why you are here today.

During this session, I unearth many of the functions even I forgot about at times when creating new code samples. Many of the functions discussed here fall into 1 of 12 different categories, and those categories are:

- **Work with objects** - *entmakex*, *dumpallproperties*, and *getpropertyvalue*/*setpropertyvalue*
- **Request custom user input** - *grread*
- **Reactors** - *vlr-command-reactor*
- **Access commands** - *initcommandversion* and *vlax-add-cmd*
- **Access symbols** - *vl-bb-ref*/*vl-bb-set*
- **Dynamic evaluation of expressions** - *read* and *eval*
- **Custom Help** - *setfunhelp*
- **Load LSP files** - *vl-load-all* and *autoload*
- **Access Third-Party COM Libraries** - *vlax-import-type-library*
- **File management** - *findfile*, *vl-mkdir*, and *vl-file-copy*
- **Validate data, debug, and handle errors** - *type*, *trace*/*untrace*, *vl-catch-all-apply*, and *atoms_family*

2 Working with Objects

Non-graphical and graphical object creation and manipulation are two of the common uses for AutoLISP. Most AutoLISP programs utilize the `command` function when creating a new object, and in most cases that might work well but there are some limitations with this approach. A few reasons why you might not want to use the `command` function are:

- Increase the backward and forward compatibility of a program
- No command-line version of a command, such as table and multileader styles
- Might need to use multiple commands to complete a task; create an object, place it on a specific layer and more
- Better control over when objects are undone using the Begin and End options of the UNDO command

Instead of using the `command` function to create a new non-graphical or graphical, you can use the `entmakex` function. The syntax for the `entmakex` function is:

```
(entmake '(dotted_pairs))
```

For example, the following draws a line from 0,0,0 to 5,5,0:

```
(entmakex '((0 . "LINE")
            (10 0.0 0.0 0.0) ; Start point
            (11 5.0 5.0 0.0) ; End point
           )
)
```

NOTE: The `entmake` function can be used to create a new object too. The difference between `entmake` and `entmakex` is that `entmakex` returns an entity name (ename) and `entmake` doesn't.

The following demonstrates a function that can add a line, circle, or layer without using a command:

```
; Helper function used to create an object without using the commands
(defun addObj (objType / newObj)
  (setq newObj
        (cond
            ((= (strcase objType) "LINE")
                (entmakex '((0 . "LINE")
                            (10 0.0 0.0 0.0) ; Start point
                            (11 1.0 1.0 1.0) ; End point
                           )
                )
            )
            ((= (strcase objType) "CIRCLE")
                (entmakex '((0 . "CIRCLE")
                            (10 0.0 0.0 0.0) ; Center point
                            (40 . 1.0)       ; Radius
                           )
                )
            )
            ((= (strcase objType) "LAYER")
                (entmakex (list (cons 0 "LAYER")
                                (cons 100 "AcDbSymbolTableRecord")
                                (cons 100 "AcDbLayerTableRecord")
                                (cons 2 "OBJ")        ; Name
                                (cons 70 0)           ; State
                                (cons 62 7)           ; Color
                                (cons 6 "Continuous") ; Linetype
                          )
                )
```

```
                    )
                )
            )
)
```

The following statements demonstrate how to use the custom *addObj* function:

```
; Create a new layer
(addObj "Layer")

; Create a new line
(addObj "Line")

; Create a new circle
(addObj "Circle")
```

Once an object has been added to the drawing, you might modify the object using a command with the `command` function or DXF manipulation. DXF manipulation can be a bit cryptic as a new AutoLISP developer or even as a veteran AutoLISP developer, but is more flexible than using a command at times.

The following shows the code that might be used to update the radius of a circle using the DXF code group value of 40:

```
(defun c:ChangeRadius ( / )
  (setq ent (car (entsel "\nSelect a circle: ")))
  (setq ed (entget ent))

  ; Change the circle's radius to 2
  (setq ed
    (subst (cons 40 2.0)
      (assoc 40 ed)
      ed
    )
  )

  ; Modify and update the object
  (entmod ed)
  (entupd ent)
 (princ)
)
```

Starting with AutoCAD 2011-based products, an alternative set of functions were introduced that provided more flexibility than using commands and required less code than manipulating DXF code group values. These functions are:

- `dumpallproperties` - Outputs the current values of the object's properties.
  Syntax: (`dumpallproperties ename` *`[context]`*)

- `getpropertyvalue` - Gets the current value of an object's property.
  Syntax: (`getpropertyvalue ename propertyname`)
- `setpropertyvalue` - Sets the current value of an object's property.
  Syntax: (`getpropertyvalue ename propertyname value`)

NOTE: The `ispropertyvalue` function returns T or nil if a property is available for the specified object.

```
(entmakex '((0 . "CIRCLE")
            (10 0.0 0.0 0.0) ; Center point
            (40 . 1.0)       ; Radius
           )
)

(dumpallproperties (entlast))
```

Here is part of the output that is created for a circle object with the `dumpallproperties` function:

```
Begin dumping object (class: AcDbCircle)
Annotative (type: bool)  (LocalName: Annotative) = Failed to get value
AnnotativeScale (type: AcString)  (RO)  (LocalName: Annotative scale) = Failed to get value
Area (type: double)  (RO)  (LocalName: Area) = 3.141593
BlockId (type: AcDbObjectId)  (RO) = 7ffffb039f0
CastShadows (type: bool) = 0
Center/X (type: double)  (LocalName: Center X) = 0.000000
Center/Y (type: double)  (LocalName: Center Y) = 0.000000
Center/Z (type: double)  (LocalName: Center Z) = 0.000000
Circumference (type: double)  (LocalName: Circumference) = 6.283185
ClassName (type: AcString)  (RO) =
Closed (type: bool)  (RO)  (LocalName: Closed) = Failed to get value
CollisionType (type: AcDb::CollisionType)  (RO) = 1
Color (type: AcCmColor)  (LocalName: Color) = BYLAYER
Diameter (type: double)  (LocalName: Diameter) = 2.000000
...
Handle (type: AcDbHandle)  (RO) = 1e2
Radius (type: double)  (LocalName: Radius) = 1.000000
...
Thickness (type: double)  (LocalName: Thickness) = 0.000000
Transparency (type: AcCmTransparency)  (LocalName: Transparency) = 0
Visible (type: AcDb::Visibility) = 0
End object dump
```

A revision to the `ChangeRadius` function in the previous code sample using the `setpropertyvalue` function would look something like the following:

```
(defun c:ChangeRadius ( / )
  (setq ent (car (entsel "\nSelect a circle: ")))
  (setpropertyvalue ent "radius" 2)
 (princ)
)
```

The following defines a function that uses the custom `addObj` function and manipulates the properties of the objects that are added:

```
(defun c:newObjects ( / layEnt newObj)
  ; Create a new Layer named Const if it does not exist
  (if (= (setq layEnt (tblobjname "layer" "Const")) nil)
    (progn
      (setq layEnt (addobj "Layer"))

      ; Update the new layers name
      (setpropertyvalue layEnt "Name" "Const")

      ; Set the color of the new layer
      ; Color can be ACI (6), true color (125,125,125),
      ; or color book (PANTONE+ CMYK Coated,PANTONE P 1-3 C)
      (setpropertyvalue layEnt "Color" "231,18,18")

      ; Set the layer so it cannot be plotted
      (setpropertyvalue layEnt "IsPlottable" 0)
    )
  )

  ; Create a new line
  (setq newObj (addObj "Line"))

  ; Set the start and end point
  (setpropertyvalue newObj "StartPoint" '(0.0 0.0 0.0))
  (setpropertyvalue newObj "EndPoint" '(5.0 5.0 0.0))

  ; Change the layer of the new line
  (setpropertyvalue newObj "LayerID" layEnt)

  ; Create a new circle
  (setq newObj (addObj "Circle"))

  ; Set the start and end point
  (setpropertyvalue newObj "Center" '(5.0 5.0 0.0))

  ; Change the layer of the new line
  (setpropertyvalue newObj "LayerID" layEnt)
```

```
 (princ)
)
```

NOTE: The sample code from this section can be found in the *2 - Working with Objects.lsp* file in the dataset.

3 Getting Custom Input

AutoLISP offers a number of functions that can be used to get input from the user, but there are times when you might want to only accept specific values or limit the number of characters that can be entered. The `grread` function can be used to define a custom function that requests the user for input and then based on the return value of the `grread` function, you can choose to continue prompting for input or not. The syntax for the `grread` function is:

(grread *[cursor_tracking] [input_filters [cursor_type]]*)

The following defines two functions that demonstrate the use of the `grread` function:

```
; Prompts for a value between 0-9 and masks the output
(defun c:MyPINCode ( / number)
  (setq number "")
  (prompt "\nEnter a whole number [backspace to clear]: ")

  (setq code (grread))

  (while (and (= 2 (car code))
              (and (/= 13 (cadr code))   ; Enter
                   (/= 32 (cadr code)))) ; Spacebar
    (if (and (>= (cadr code) 48)
             (<= (cadr code) 57))
      (progn
        (setq ch (chr (cadr code)))
        (setq number (strcat number ch))
        (princ "*")
      )
    )

    ; Support backspace
    (if (= (cadr code) 8)
      (progn
        (repeat (strlen number)
          (princ (chr 8))
        )
        (setq number "")
      )
    )

    ; Ask for more input if the user did not press Enter or Space
```

```
    (if (or (/= 13 (cadr code))(/= 32 (cadr code)))
      (setq code (grread))
    )
  )

  (prompt (strcat "\nPIN entered was: " number))
 (princ)
)

; Prompts for a single character
(defun c:GetCharacter ( / code)
  (prompt "\nEnter a single character: ")

  (setq code (grread))

  (if (= 2 (car code))
    (progn
      (prompt (strcat "\nCharacter entered was: " (chr (cadr code))))
      (prompt (strcat "\nASCII code: " (itoa (cadr code))))
    )
    (prompt "\nInput was not from the keyboard.")
  )
 (princ)
)
```

NOTE: The sample code from this section can be found in the *3 - Getting Custom Input.lsp* file in the dataset.

4 Reacting to Commands

AutoCAD monitors a lot of actions that take place in the application, while a drawing is open, and when objects are added to or modified in a drawing. AutoLISP allows you to step in and catch many of these actions with what is known as a *reactor*. Reactors come in a variety of types.

Command reactors are one of the most commonly used reactor types. A command reactor allows you to be notified when a command is started, ends, cancelled, or fails for some reason. For example, you can monitor the use of the HATCH or BHATCH commands and make sure a specific layer or settings are current. You register a command reactor with the `vlr-command-reactor` function.

The syntax for the `vlr-command-reactor` function is:

```
(vlr-command-reactor data '(callback))
```

data specifies the custom values to pass to and have the reactor return so you can filter out one reactor from another. The callback argument is a list that specifies the event to monitor and the AutoLISP function that should be executed.

NOTE: The Visual LISP environment must first be loaded to use a reactor.

The following registers a command reactor that monitors 4 different types of events:

```
; Check to see if our custom command reactors have been loaded into
the current drawing
(if (= hyp-rctCmds nil)
  ; Add the command reactors and the custom callbacks
  (setq hyp-rctCmds (vlr-command-reactor nil
                           '((:vlr-commandCancelled . hyp-cmdAbort)
                             (:vlr-commandEnded . hyp-cmdAbort)
                             (:vlr-commandFailed . hyp-cmdAbort)
                             (:vlr-commandWillStart . hyp-cmdStart)
                            )
                    )
  )
)

; Callback used when the user presses ESCape
; or when the command ends by itself or due to
; a problem
(defun hyp-cmdAbort (param1 param2)
  ; Check to see if our global variable has a value
  ; if it does then set the layer current
  (if (/= hyp-gClayer nil)
    (setvar "clayer" hyp-gClayer)
  )

  ; Clear the global variable
  (setq hyp-gClayer nil)
)

; Callback used when a command is started
(defun hyp-cmdStart (param1 param2)

  ; Store the current layer in a global variable
  (setq hyp-gClayer (getvar "clayer"))

  ; Check to see what command has been started, the
  ; command name is stored in the param2 variable
  (cond
    ; If the QDIM command is active then set the layer
    ; to the Dims layer
    ((= (car param2) "QDIM")(prompt "\nQDIM started"))

    ; If either the HATCH, BHATCH or GRADIENT command
    ; is active then set the current layer to Hatch
    ((or (= (car param2) "HATCH")
       (= (car param2) "BHATCH")
```

```
      (= (car param2) "GRADIENT")
    )
     (progn
       (if (= (tblsearch "layer" "Hatch") nil)
         (progn
           ; Create a reference to AutoCAD
           (setq acadObj (vlax-get-acad-object))

           ; Create a reference to the current drawing
           (setq docObj (vla-get-activedocument acadObj))

           ; Create the Hatch layer
           (setq layerObj (vla-add (vla-get-layers docObj) "Hatch"))

           ; Set the Hatch layer to red
           (vla-put-color layerObj acRed)
         )
       )
       (setvar "clayer" "Hatch")
     )
   )
  )
)
```

NOTE: The sample code from this section can be found in the *4 - Reacting to Commands.lsp* file in the dataset.

5 Working with Commands

Starting with AutoCAD 2009-based programs, commands are now versioned each time a significant change is made in response to the introduction of the Action Recorder. When a command is executed, it executes the most recent version of a command. Older scripts and AutoLISP programs might not execute properly based on the current version of a command. Using the `initcommandversion` function, you can indicate which version of a command should be executed with the next the `command` or `command-s` function.

The syntax for the `initcommandversion` function is:

```
(initcommandversion version)
```

The following demonstrates how to use the `initcommandversion` function:

```
; Executes the COLOR command at the Command prompt
(initcommandversion 1)
(command "._color" "3")

; Executes the COLOR command and display the Color dialog box
(initcommandversion 2)
(command "._color" "3") ; Unknown command 3 error occurs
```

TIP: Command version 2 of the -INSERT command allows the user to specify properties in the Properties palette.

While you can create a function with AutoLISP that acts similar to a command by adding the c: prefix to the functions name, the function doesn't completely act like a standard built-in command. For example, an AutoLISP function defined with c: can be executed transparently while if the function draws new objects it is best not to allow it to be executed transparently. Using the `vlax-add-cmd` function, you can register an AutoLISP function as modal or restrict the use of pickfirst selection behavior.

The syntax for the `vlax-add-cmd` function is:

(vlax-add-cmd *global_name LISP_function [local_name [flags]]*)

The following demonstrates how to use the `vlax-add-cmd` function:

```
; Defines a function that displays the number of objects selected
(defun NumberOfObjects ( / ss)
  (setq ss (ssget))
  (alert (strcat "Number of objects selected: " (itoa (sslength ss))))
)

; Adds a command that is modal
(vlax-add-cmd "NumModal" 'NumberOfObjects
              "NumModal" ACRX_CMD_MODAL)

; Adds a command that is modal and supports pickfirst
(vlax-add-cmd "NumModalPickSet" 'NumberOfObjects
              "NumModalPickSet"
              (+ ACRX_CMD_MODAL ACRX_CMD_USEPICKSET))

; Adds a command that is transparent
(vlax-add-cmd "NumTrans" 'NumberOfObjects
              "NumTrans" ACRX_CMD_TRANSPARENT)

; Removes the command
(vlax-remove-cmd "NumModal")
(vlax-remove-cmd "NumModalPickSet")
(vlax-remove-cmd "NumTrans")
```

NOTE: You remove a command defined using the `vlax-add-cmd` function with the `vlax-remove-cmd` function. To remove the command, you pass the global name of the command to the `vlax-remove-cmd` function.

NOTE: The sample code from this section can be found in the *5 - Working with Commands.lsp* file in the dataset.

6 Accessing Variables across All Drawings

Variables are used to store values for use later and typically are defined with the `setq` function. While not all variables need to be accessed from outside of the current drawing, there are times when you might want to access a value across all drawings that are opened during a session. For example, maybe you want to access the most recently output file from a custom program and instead of using the Windows Registry (Windows) or Plist file (Mac OS X) you store the value in a variable. The `vl-bb-ref` function allows you to access values that have been posted to what is known as the *blackboard*. Values are put on the blackboard using the blackboard with the `vl-bb-set` function.

The syntax for the `vl-bb-ref` and `vl-bb-set` functions is:

```
(vl-bb-ref ‘variable)

(vl-bb-set ‘variable value)
```

The following demonstrates using the `vl-bb-ref` and `vl-bb-set` functions:

```
; Defines a variable named notes-template-file on the blackboard
(vl-bb-set 'notes-template-file "notes.xml")
"notes.xml"

; Gets the value of the variable notes-template-file on the blackboard
(vl-bb-ref 'notes-template-file)
"notes.xml"

; Tries to get the current value of notes-template-file
(eval notes-template-file)
nil

; Sets the notes-template-file with setq
(setq notes-template-file "notes2.xml")
"notes2.xml"

; Tries to get the current value of notes-template-file
(eval notes-template-file)
"notes2.xml"

; Gets the value of the variable notes-template-file on the blackboard
(vl-bb-ref 'notes-template-file)
"notes.xml"
```

NOTE: The sample code from this section can be found in the *6 - Accessing Variables across All Drawings.lsp* file in the dataset.

7 Dynamically Evaluating Functions

In most AutoLISP programs, the expressions that are to be executed are loaded into memory as part of a function. Using the `read` function, you can convert a string to a symbol or expression making it possible to create expressions dynamically on the fly and then evaluate the expression. The syntax for the `read` function is:

```
(read string)
```

For example, the following converts the string "(/ 12 (1+ 2))" to the following expression:

```
(/ 12 (1+ 2))
```

The expression returned by the `read` function can be evaluated with the `eval` function. The syntax for the `eval` function is:

```
(eval atom)
```

For example, the following evaluates the (/ 12 (1+ 2)) and returns the value of 4:

```
(eval (read "(/ 12 (1+ 2))"))

4
```

If a variable is passed to the `eval` function, the value of the variable is returned much like using the ! (exclamation point) at the Command prompt.

```
(eval PI)

3.14159
```

The following demonstrates how to use the `read` and `eval` functions to replace a string value with the actual value of a variable. The variable that is replaced is positioned between two percentage symbols.

```
; Swaps out a variable name with its value in a string
(defun expVar (str / )
  (while (wcmatch str "*%*%*")
    (progn
      (setq start (1+ (vl-string-search "%" str)))
      (setq next (vl-string-search "%" str start))
      (setq var (substr str start (- (+ next 2) start)))

      (setq expVal (vl-princ-to-string
                     (eval (read (vl-string-trim "%" var)))))
```

```
      (if (/= expVal nil)
        (setq str (vl-string-subst expVal var str))
      )
    )
  )
  str
)

; Defines the string to replace and a new variable
(setq *product* (getvar "PRODUCT")
      str2exp "PI=%PI% Product=%*product*%"
)

; Execute the custom function and return the new string
(expVar str2exp)
"PI=3.14159 Program=acad"
```

NOTE: The sample code from this section can be found in the *7 - Dynamically Evaluating Functions.lsp* file in the dataset.

8 Setting up Custom Help

Creating and implementing help isn't something that many developers think about, but is important for users to get the answers as to how your custom functions work. The `setfunhelp` function registers contextual help for a custom function that has the c: prefix. Contextual help means that the help topic associated with your custom function is displayed when the function is executing and the user presses F1. Pressing F1 when your custom function is entered at the Command prompt, but not executed yet will also display the help topic registered with your custom function.

The syntax for the `setfunhelp` function is:

```
; CHM/WinHelp files
(setfunhelp c:function_name [helpfile [topic [flag]]])

; HTML files
(setfunhelp c:function_name [helpfile])
```

The following demonstrates how to use the `setfunhelp` function to add contextual help to the custom function named *c:helloau*:

```
; Custom function
(defun c:HelloAU () (alert "Hello AU2014!"))

; Adds F1 support to a web page
(setfunhelp "c:HelloAU" "http://www.autodesk.com")
```

```
; Adds F1 support to a local HTML file
(setfunhelp "c:HelloAU" (strcat "file:///" (findfile "help.htm")))
(setfunhelp "c:HelloAU" (findfile "help.htm"))

; Adds F1 support to a CHM file
(setfunhelp "c:HelloAU" "C:/Program Files/Autodesk/AutoCAD 2012 -
English/Help/acet.chm" "html/ATTIN")
```

TIP: A custom help topic can also be displayed using the `help` function.

NOTE: The sample code from this section can be found in the *8 - Setting up Custom Help.lsp* file in the dataset.

9 Loading AutoLISP Files

AutoLISP functions and files are often loaded using the *acad.lsp* and *acaddoc.lsp* files. You might even load a LSP file using the `load` function. The using the *acad.lsp* file and the `load` function results in the LSP files to be loaded only into the current drawing. The `vl-load-all` function can be used to load a LSP file into the current drawing or any drawing that is opened during the current session.

The syntax for the `vl-load-all` function is:

(vl-load-all *filename*)

The following statement loads the LSP file named *9 - Loading AutoLISP Files.lsp* into the current and all successive drawing files:

```
(vl-load-all "9 - Loading AutoLISP Files.lsp")
```

Some LSP files can be rather large in size, and you might not want to load all your LSP files into memory at once. The `autoload` function can be used to load an AutoLISP file on demand when a function is executed the first time. By delaying the loading of an AutoLISP file with the `autoload` function it can help to free up system resources.

The syntax for the `autoload` function is:

(autoload *LSP_filename function_list*)

The following shows an example of a LSP file that uses the autoload function to load two other LSP file as needed:

```
; 9 - Loading AutoLISP Files.lsp
(autoload "autoload_ex1.lsp" '("AU_EX_1A" "AU_EX_1B"))
(autoload "autoload_ex2.lsp" '("AU_EX_2"))

; autoload_ex1.lsp
(defun c:AU_EX_1A ()
```

```
  (alert "Autoload_ex1:A")
)

(defun c:AU_EX_1B()
  (alert "Autoload_ex1:B")
)

; autoload_ex2.lsp
(defun c:AU_EX_2 ()
  (alert "Autoload_ex2")
)
```

NOTE: The sample code from this section can be found in the *9 - Loading AutoLISP Files.lsp*, *autoload_ex1.lsp*, and *autoload_ex2.lsp* files in the dataset.

10 Leveraging ActiveX/COM

One of the least used features of the AutoLISP programming language was introduced with AutoCAD 2000-based products, that feature is the ability to utilize ActiveX/COM. Part of the reason why it might not be used as frequently is the lack of understanding of what is possiblely, and secondly it is a bit unnatural since there isn't any really good documentation available. The AutoLISP functions that begin with VLA are implemented as a result of importing the AutoCAD Object library. The `vlax-import-type-library` function is used to expose the methods, properties, and constants of an ActiveX/COM library for use with AutoLISP. The syntax for the `vlax-import-type-library` function is:

```
(vlax-import-type-library :tlb-filename pathname
                          [:methods-prefix mprefix
                           :properties-prefix pprefix
                           :constants-prefix cprefix]
)
```

NOTE: You might need to execute the `vl-load-com` function based on the AutoCAD release you are using.

The following shows two custom functions that implement behavior defined by two ActiveX/COM libraries; AutoCAD ObjectDBX and Microsoft XML libraries:

```
; Load COM environment
(vl-load-com)

; Open drawing in the background and import all dimension styles
(defun c:AccessExternalDrawing ( / acdbObj cntDimstyles acObject
                                   arTemp arDimStyles cntStep)

  ;; Load the ObjectDBX library
```

```
  (if (= acLibImport nil)
    (progn
      (vlax-import-type-library :tlb-filename "C:\\Program
Files\\Common Files\\Autodesk Shared\\axdb20enu.tlb"
                                :methods-prefix "acdbm-"
                                :properties-prefix "acdbp-"
                                :constants-prefix "acdbc-"
      )
      (setq acLibImport T)
    )
  )

  ; Create a reference to the ObjectDBX object
  (setq acdbObj (vlax-create-object "ObjectDBX.AxDbDocument.20"))

  ; Open drawing file
  (acdbm-open acdbObj (findfile "External Drawing.dwg"))

  ; Set the counter for dimension styles to zero
  (setq cntDimstyles 0)

  ; Loop through the dimension styles collection
  (vlax-for acObject (vla-get-Dimstyles acdbObj)

    ; Get the current values in the array and then adjust the size
    (setq arTemp arDimStyles)
    (setq arDimStyles (vlax-make-safearray
                          vlax-vbObject (cons 0 cntDimstyles)))

    ; Step through the old array values and
    ; load them into the new array
    (setq cntStep 0)
    (repeat cntDimstyles
      (vlax-safearray-put-element arDimStyles
            cntStep (vlax-safearray-get-element arTemp cntStep))
      (setq cntStep (1+ cntStep))
    )

    ; Add the dimension style to the array
    ; and increment the counter by 1
    (vlax-safearray-put-element arDimStyles cntDimstyles acObject)
    (setq cntDimstyles (1+ cntDimstyles))
  )
```

```
  ; Create a reference to AutoCAD
  (setq acadObj (vlax-get-acad-object))

  ; Create a reference to the current drawing
  (setq docObj (vla-get-activedocument acadObj))

  ; Create a reference to the database object of the drawing
  (setq dbObj (vla-get-database docObj))

  ; Create a reference to the
  (setq dimstylesColl (vla-get-dimstyles dbObj))

  ; Copy the dimension styles from the drawing
  ; open in memory to the current drawing
  (vla-copyobjects acdbObj arDimStyles dimstylesColl)

  ; Release the ObjectDBX object
  (vlax-release-object acdbObj)
 (princ)
)


; Parses the Notes.xml file
(defun c:ReadXML ( / )
  ; Import the Microsoft XML library
  (if (= xmlLibImport nil)
    (progn
      (vlax-import-type-library :tlb-filename
"c:\\windows\\system32\\msxml6.dll"
                                :methods-prefix "xmlm-"
                                :properties-prefix "xmlp-"
                                :constants-prefix "xmlc-"
      )
      (setq xmlLibImport T)
    )
  )

  ; Create and configure the XML Document object
  (setq oXMLDoc (vlax-create-object "MSXML2.DOMDocument"))
  (xmlp-put-async oXMLDoc :vlax-false)
  (xmlp-put-validateOnParse oXMLDoc :vlax-false)
```

```
  ; Load the XML file
  (xmlm-load oXMLDoc (findfile "notes.xml"))

  ; Gets the Root node of the XML file
  (setq oNodeCatalog (xmlp-get-documentElement oXMLDoc))
  (setq oChildNodes (xmlp-get-ChildNodes oNodeCatalog))

  ; Step through each Note in the XML file
  (setq cntNote 0)
  (while (< cntNote (xmlp-get-length oChildNodes))
    ; Gets and display the ID of the Note element
    (setq oNote (xmlp-get-Item oChildNodes cntNote))
    (prompt (strcat "\nID: "
       (xmlp-get-Text (xmlp-get-Item (xmlp-get-Attributes oNote) 0))))

    ; Gets the children of the Note element
    (setq oNoteChildNodes (xmlp-get-ChildNodes oNote))

    ; Step through each Note element
    (setq cntNoteChild 0)
    (while (< cntNoteChild (xmlp-get-length oNoteChildNodes))
      ; Get the node under the Note element
      (setq oNoteChild (xmlp-get-Item oNoteChildNodes cntNoteChild))
      (prompt (strcat "\n" (xmlp-get-BaseName oNoteChild)
                               ": " (xmlp-get-Text oNoteChild)))

      ; Increment the counter
      (setq cntNoteChild (1+ cntNoteChild))
    )

    (setq cntNote (1+ cntNote))
  )

  ; Release the XML Document object
  (vlax-release-object oXMLDoc)
)
```

NOTE: The sample code from this section can be found in the *10 - Leveraging ActiveX_COM.lsp* file in the dataset.

11 Managing Files

The AutoCAD application relies on a variety of files; those files are typically defined as part of the application settings. The Support File Search Path contains a majority of the files that are used by the AutoCAD program and drawing files that control the way objects are displayed on-screen or plotted. The `findfile` function can be used to locate a file within the AutoCAD support file search paths or validate a file exists at a specific location. The syntax for the `findfile` function is:

```
(findfile pathname)
```

TIP: When it comes to setting up AutoCAD, it can be beneficial to add the main folder of your custom files and use subfolders to organize those files. Then as part of the path passed to the `findfile` function, you can provide a subfolder instead of the full path.

The following shows different ways of validating a file:

```
; Hard coded path for a LSP file, not ideal in programming
(load "C:\\Program Files\\Autodesk\\AutoCAD
2015\\Tutorial\\VisualLISP\\Lesson1\\gpmain.lsp")

; No path to the LSP file; requires a deeper path setup in AutoCAD
(load "gpmain.lsp")

; Partial path to the LSP file
(load (findfile "Tutorial\\VisualLISP\\Lesson1\\gpmain.lsp"))
```

In addition to validating the location of a file with the `findfile` function, you can perform some basic file management tasks with the following AutoLISP functions:

- `vl-mkdir` - Creates a new directory (or folder).
  Syntax: (`vl-mkdir directoryname`)
- `vl-file-copy` - Copies a file from one directory to another.
  Syntax: (`vl-file-copy source_file destination_file` *`[append]`*)

NOTE: There are other file management related functions that AutoLISP supports, you can learn more about those functions in the AutoCAD Online help.

The following shows how you can utilize some of the file management functions of AutoLISP to create a pseudo archival/transmittal program:

```
(defun c:FileDependencies ( / acadObj acDoc acFileDependencies
                              strDestinationDir strFileName
                              strFileExt strFileLocation)
  (setq acadObj (vlax-get-acad-object))
  (setq acDoc (vlax-get-property acadObj 'ActiveDocument))
```

```
(setq acFileDependencies
      (vlax-get-property acDoc 'FileDependencies))

(setq strDestinationDir "c:\\export_project")

(if (= (findfile strDestinationDir) nil)
  (vl-mkdir strDestinationDir)
)

(setq strDestinationDir (strcat strDestinationDir "\\"))

; Loop through the FileDependency collection
(vlax-for acFileDependency acFileDependencies

  ; Get the file name
  (setq strFileName
        (vlax-get-property acFileDependency
                           'FullFileName))

  (setq strFoundPath
        (vlax-get-property acFileDependency
                           'FoundPath))

  ; Get the file path for the file
  (if (/= (vlax-get-property acFileDependency 'FoundPath) "")
    (setq strFileLocation (strcat strFoundPath strFileName))
    (setq strFileLocation strFileName)
  )

  (setq strFileExt (substr (vl-filename-extension strFileLocation)
                           2)
        strFileNameOnly (vl-filename-base strFileLocation)
  )

  ; Check to see if the file is found at the location provided
  (if (/= (findfile strFileLocation) nil)
    (progn
      ; Check to see if the directory exists
      (if (= (findfile (strcat strDestinationDir strFileExt)) nil)
        (createDirectoryPaths
            (strcat strDestinationDir strFileExt)
        )
      )

      ; Copy the file to the directory1
      (vl-file-copy strFileLocation
                    (strcat strDestinationDir strFileExt "\\"
                            strFileNameOnly "." strFileExt)
      )
```

```
      )
    )
  )
 (princ)
)

; Utility function to build directory paths
; Used with the c:FileDependencies function/command definition.
(defun createDirectoryPaths (folderName / strTemp strTemp2 nSlash)
  (setq strTemp ""
     strTemp2 folderName
     nSlash 0
  )

  ; Check to see if the folderName still has characters left in it
  (while (> (strlen folderName) 0)

    ; Check for a slash
    (setq nSlash (vl-string-search "\\" folderName))

    (if (= nSlash 0)
      (setq nSlash (vl-string-search "/" folderName))
      (if (= nSlash nil)(setq nSlash 0))
    )

    ; Get the next directory level to the folderName
    (if (> nSlash 0)
      (progn
        (setq strTemp2 (substr folderName 1 nSlash))
        (setq folderName (substr folderName (+ nSlash 2)))
      )
      (setq strTemp2 folderName
         folderName "")
    )

    ; Check to see if the last character is a slash or not
    (if (and (/= (substr strTemp2 (strlen strTemp2)) "\\")
             (/= (substr strTemp2 (strlen strTemp2)) "/"))
      (setq strTemp2 (strcat strTemp2 "\\"))
    )

    ; Check to see if the directory already exists
    (if (= (findfile (strcat strTemp strTemp2)) nil)
      (vl-mkdir (strcat strTemp strTemp2))
    )

    ; Keep adding to the folder name
    (setq strTemp (strcat strTemp strTemp2))
  )
```

```
 (princ)
)
```

NOTE: The sample code from this section can be found in the *11 - Managing Files.lsp* file in the dataset.

12 Validating Data Types

Prior to using the value of a variable, you should always verify that the value isn't *nil*. Testing for *nil* isn't the only condition you shouldn't test for, you should also test for the type of data that a variable is assigned to avoid potential errors.

The syntax for the `type` function is:

(type *atom*)

The following demonstrates how to use the `type` function to test for a string value:

```
; Funtion that tests a value to see if it is a string
(defun isstring (valTest / )
  (if (= (type valTest) 'STR)
    T
    nil
  )
)

; Example usage
(isstring 1.25)
nil

(isstring '("Hello"))
nil

(isstring "Hello")
nil
```

Even if you validate the values and the types of data used in your programs, things can still go wrong. The `vl-catch-apply-all` function can be used to catch an error from a function and allow execution to resume, giving you a second chance to handle the error. In addition to the `vl-catch-apply-all` function, you can use the `vl-catch-all-error-message` and `vl-catch-all-error-p` functions to get information about the error and check to see if an error occurred.

Here is the syntax of the three functions:

```
(vl-catch-all-apply 'function list)
(vl-catch-all-error-message err-object)
(vl-catch-all-error-p symbol/expression)
```

The following example demonstrates the use of the `vl-catch-all-apply`, `vl-catch-all-error-message` function, and `vl-catch-all-error-p` functions:

```
; Example shows how to work with the Description property of a Layer.
(defun c:layerDescription-WithCatch ( / acDoc acLayer acLayers
                                        strLayerName strLayerDesc)
  ; Check to see if the Visual LISP ActiveX functions have been loaded
  (if (/= (type vlax-get-property) 'SUBR)(vl-load-com))

  ; Get the active drawing and Layers collection
  (setq acDoc (vlax-get-property
                          (vlax-get-acad-object) 'ActiveDocument)
        acLayers (vlax-get-property acDoc 'Layers)
  )

  ; Request the name of the layer to modify
  (while (/= (setq strLayerName
                   (getstring T
                "\nEnter layer name to add description to: ")) "")

    ; Check to see if the Layer exists in the drawing
    (if (not (vl-catch-all-error-p
               (setq err
                     (vl-catch-all-apply 'vla-item
                                 (list acLayers strLayerName)))))
      (progn
        ; Layer exists so ask for a description to add to the Layer
        (setq strLayerDesc
            (getstring T "\nEnter description for layer: "))

        ; If the user just pressed enter then remove the description
        (if (= strLayerDesc nil)(setq strLayerDesc ""))

        ; Change the Description for the Layer
        (vlax-put-property
          (vla-item acLayers strLayerName) 'Description strLayerDesc)
      )
      (progn
        (prompt (strcat "\nerror: " (vl-catch-all-error-message err)))
        (setq strLayerName "")
      )
    )
  )

 (princ)
)
```

The AutoLISP functions `trace` and `untrace` enable and disable the trace flag for a specified function. When a trace is set for a function, you are notified at the Command prompt or in the

Visual LISP Editor Trace window when the function is used, along with the value that the function was passed and the return/result of the function. This can be helpful in identifying why a function might be failing.

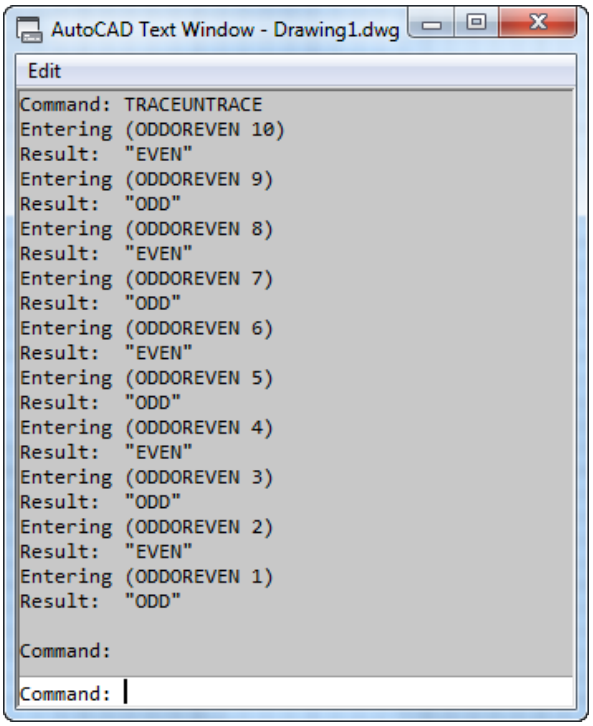
The syntax for the `trace` and `untrace` functions are:

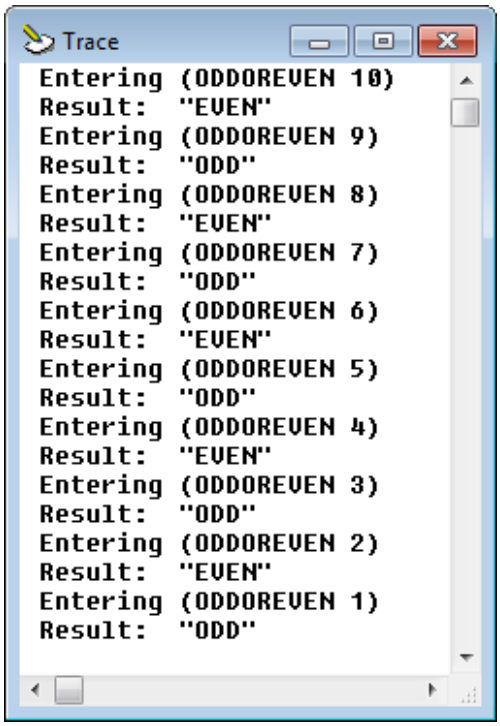
```
(trace function)
(untrace function)
```

The following demonstrates the use of the `trace` and `untrace` functions.

```
(defun OddOrEven (cnt / )
  (if (= (rem cnt 2) 1)
    "ODD"
    "EVEN"
  )
)

(trace OddOrEven)

(defun c:TraceUntrace ( / )
  (setq cnt 10)

  (while (> cnt 0)
    (OddOrEven cnt)

    (setq cnt (1- cnt))
  )
 (princ)
)
```

The following images show the results of tracing the OddOrEven function in the Command prompt or Visual LISP Trace window.





Last but not least is an interesting function named `atoms_family`, not to be confused with the Adams Family. The `atoms_family` function can be used to evaluate the AutoLISP environment. The function returns a list containing the functions and variables defined by or exposed to AutoLISP. Using the results generated, you can manipulate the values in the list to identify variables that might need to be declared as local instead of global for your programs or even to see which functions are defined in the current drawing session.

The syntax for the `atoms_family` function is:

(atoms_family *flag [search_list]*)

The following is example code that demonstrates the use of the `atoms_family` function:

```
; Generates a text file that contains information
; about the AutoLISP environment
(defun c:LISPDump ( / afList nSyms nFuncs nCmds nVars
                      lstSyms lstFuncs lstCmds lstVars
                      item fp)
    (setq afList (atoms-family 1))
    (setq nSyms 0 nFuncs 0 nCmds 0 nVars 0
          lstSyms nil lstFuncs nil lstCmds nil lstVars nil)
    (foreach item aflist
```

```
        (progn
           (cond
               ((= (type (eval (read item))) 'SYM)
                    (progn
                        (setq nSyms (1+ nSyms))
                        (if (/= lstSyms nil)
                            (setq lstSyms (append lstSyms
                                                  (list item)))
                            (setq lstSyms (list item))
                        )
                    )
               )
               ((and (= (type (eval (read item))) 'SUBR)
                     (/= (substr (strcase item) 1 2) "C:"))
                     (progn
                          (setq nFuncs (1+ nFuncs))
                          (if (/= lstFuncs nil)
                              (setq lstFuncs (append lstFuncs
                                                     (list item)))
                              (setq lstFuncs (list item))
                          )
                     )
               )
               ((and (= (type (eval (read item))) 'SUBR)
                     (= (substr (strcase item) 1 2) "C:"))
                     (progn
                          (if (/= (strcase item)
                                  (strcase "c:LISPDump"))
                              (progn
                                  (setq nCmds (1+ nCmds))
                                  (if (/= lstCmds nil)
                                      (setq lstCmds (append lstCmds
                                                      (list item)))
                                      (setq lstCmds (list item))
                                  )
                              )
                          )
                     )
               )
               ((or (= (type (eval (read item))) 'REAL)
                    (= (type (eval (read item))) 'INT)
                    (= (type (eval (read item))) 'LIST)
                    (= (type (eval (read item))) 'PICKSET)
                    (= (type (eval (read item))) 'ENAME)
                    (= (type (eval (read item))) 'VLA-OBJECT)
                    (= (type (eval (read item))) 'FILE)
                    (= (type (eval (read item))) 'VARIANT)
                    (= (type (eval (read item))) 'STR))
                        (progn
```

```
                            (setq nVars (1+ nVars))
                            (if (/= lstVars nil)
                                (setq lstVars (append lstVars
                                                      (list item)))
                                (setq lstVars (list item))
                            )
                        )
                )
            )
        )
    )
    ;; Output details about defined Symbols and Functions
    (prompt (strcat "\nAutoLISP Symbols and Functions count: "
                    "\nSymbols - " (itoa nSyms)
                    "\nFunctions - " (itoa nFuncs)
                    "\nGlobal Variables - " (itoa nVars)
                    "\nCommands - " (itoa nCmds))
    )
    (setq fp (open "c:\\LspDumpFile.log" "w"))
    (write-line "Symbols" fp)
    (setq lstSyms (acad_strlsort lstSyms))
    (foreach item lstSyms
        (write-line item fp)
    )
    (write-line "" fp)
    (write-line "Functions" fp)
    (setq lstFuncs (acad_strlsort lstFuncs))
    (foreach item lstFuncs
        (write-line item fp)
    )
    (write-line "" fp)
    (write-line "Global Variables" fp)
    (setq lstVars (acad_strlsort lstVars))
    (foreach item lstVars
        (write-line item fp)
    )
    (write-line "" fp)
    (write-line "Commands" fp)
    (setq lstCmds (acad_strlsort lstCmds))
    (foreach item lstCmds
        (write-line item fp)
    )
    (close fp)
  (princ)
)
```

Using the previous example code, you could execute the code twice and then use a utility that can see what the differences are between the two files that are generated. For example, execute the code and rename the output file and then load/execute the LSP functions you want

to perform a difference against. Then execute the previous example code again to generate the output file to compare against.

NOTE: The sample code from this section can be found in the *12 - Data_Validation_Debugging.lsp* file in the dataset.

A1 Where to Get More Information

When you are first starting to use a new feature, you will have questions and where you go to find answers might not be clear. The following is a list of resources that you can use to get help:

- **Help System** – The Developer Home page in the AutoCAD Online Help system can be helpful in locating information about AutoLISP and other programming options that AutoCAD supports. To access the AutoCAD Online help, go to: *http://help.autodesk.com/view/ACD/2015/ENU/*.
- **Autodesk Discussion Forums** – The Autodesk forums provide peer-to-peer networking and some interaction with Autodesk moderators. You can ask a question about anything AutoCAD related and get a response from a fellow user or Autodesk employee. To access the discussion forums, go to *http://forums.autodesk.com*, click AutoCAD, and then click one of the links for a subgroup.
- **AUGI Forums** – The AUGI forums provide peer-to-peer networking where you can ask questions about virtually anything in AutoCAD and get a response from a fellow user. Visit AUGI at *http://www.augi.com/*.
- **Industry Events and Classes** – Industry events such as AUGI CAD Camp and Autodesk University are great places to learn about new features in an Autodesk product. Along with industry events, you might also be able to find classes at your local technical college or Autodesk Authorized Training Center (ATC).
- **Internet** – There are tutorials on the Internet to learn many of the customization and programming options that are offered in AutoCAD. Use your favorite search engine, such as Google or Bing and search on the topic of interest.
- **Books** – The AutoLISP book in the AutoCAD Customization Platform series published by Wiley & Sons. The AutoCAD Customization Platform : AutoLISP book contains a wide range of sample code that can be helpful in getting started with AutoLISP or to get more from it

A2 Runner-up Functions

Other functions that I considered for this class were:

- foreach
- lamda
- mapcar
- vl-acad-defun/vl-acad-undefun
- zerop

You can learn more about these functions in the AutoCAD Online Help system.