SD6751 - AutoLISP®: The 21 Frequently Overlooked and Forgotten Functions

Lee Ambrosius – Autodesk, Inc.

Principal Learning Content Developer – IPG

AutoCAD Products – Learning Experience

Where Am I and Who Should Be Here

You are in session:
SD6751 - AutoLISP®: The 21 Frequently Overlooked and Forgotten Functions

You should know:
AutoCAD 2015 (or AutoCAD 2011 and later)

You should want to:

- Learn about obscure AutoLISP functions
- Extend the capabilities of your AutoLISP programs

Key learning objectives

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

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

Who Am I?

My name is Lee Ambrosius

- Principal Learning Content Developer at Autodesk
- AutoCAD user for over 20 years
- Work on the Customization, Developer, and CAD Administration documentation
  - Customizing and programming for over 18 years
  - AutoLISP/DCL, VBA, Managed .NET, and ObjectARX/MFC
- Author and Technical Editor
  - AutoCAD and AutoCAD LT Bible Series
  - AutoCAD Customization Platform Series

Session Rules

A few rules for this session:

- Silent your mobile phone and any other device
- If you have to leave at anytime, please do so quietly
- Hold questions until the end

Thanks for Your Cooperation

Working with Objects

entmakex function

- *command* function isn't the only way to create new objects
- Objects can be created using the *entmakex* function
- The *entmakex* accepts a list of dotted pairs and returns the entity name of the new object

dumpallproperties function

- Accepts an entity name

- Outputs the name and current property values supported by an entity name

For example:

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
Diameter (type: double) (LocalName: Diameter) = 2.000000

getpropertyvalue/setpropertyvalue function

- *getpropertyvalue* function returns the current value of a property for an entity
- *setpropertyvalue* function assigns a value to the property of an entity
- Both functions expect an entity and property name
- Property name and expected value can be obtained with the *dumpallproperties* function

Working with Objects

Demos:

- Creating new objects
- Modifying objects

Sample file:

2 - Working with Objects.lsp

Getting Custom Input

grread function

- *getXxx* functions aren't the only ways of getting input at the Command prompt
- Text and point input can be obtained using the *grread* function
- Input can also be filtered based on the input returned by the *grread* function

Getting Custom Input

Demos:

- Getting numbers only and masking the output
- Prompting for a single value

Sample file:

3 - Getting Custom Input.lsp

Reacting to Commands

vlr-command-reactor function

- *vlr-command-reactor* function monitors the use of commands
- Commands are the most frequent used action
- You can tell when a command
  - Will start
  - Ended or failed
  - Is cancelled

Reacting to Commands

Demo:

- Switching layers before hatching a closed object

Sample file:

4 - Reacting to Commands.lsp

Working with Commands

initcommandversion function

- *initcommandversion* function specifies the version of a command to execute
- Affects the command executed with *command* or *command-s* function
- Introduced in AutoCAD 2009 with Action Record
- If the function isn't used, the recent version of a command is executed

initcommandversion function

Some of the commands affected by *initcommandversion*

- COLOR
- COPY
- EXPLODE
- FILLET
- -INSERT

vlax-add-cmd function

- *vlax-add-cmd* function defines a custom function as a built-in command
- Support for global and local command names
- Be defined as modal or transparent
- Support pickfirst selection or the redrawing of the drawing area

Working with Commands

Demos:

- Specifying a specific version of a command
- Defining a custom function as a built-in command

Sample file:

5 - Working with Commands.lsp

Accessing Variables across All Drawings

vl-bb-set/vl-bb-ref functions

- *vl-bb-set* function specifies a variable on the “blackboard”
- *vl-bb-ref* function gets a variable on the “blackboard”
- Variables on the “blackboard” don’t affect those defined with the *setq* function

Accessing Variables across All Drawings

Demo:

- Working with variables on the “blackboard”

Sample file:

6 - Accessing Variables across All Drawings.lsp

Dynamically Evaluating Functions

read/eval functions

- *read* function converts a string to a symbol or expression
- *eval* function can be used to execute an expression returned by the *read* function
- *eval* function returns the value of a variable

Dynamically Evaluating Functions

Demo:

- Expanding a string

Sample file:

7 - Dynamically Evaluating Functions.lsp

Setting up Custom Help

setfunhelp function

- Registers a topic for the contextual help of a custom function prefixed with c:
- Help access is for your users/reduce support calls
- Help topic can be a:
  - HTML/HTM file
  - CHM file
  - HLP (legacy WinHelp) file

Setting up Custom Help

Demo:

- Implementing contextual help

Sample file:

8 - Setting up Custom Help.lsp

Loading AutoLISP Files

vl-load-all function

- *load* function loads a LSP file into the current drawing
- *vl-load-all* function loads a LSP file into the current and all successive drawings in the current session

autoload function

- *autoload* function sets up demand loading of a LSP file when a custom function is used
- Frees up system resources by not loading all LSP files at startup

Loading AutoLISP Files

Demo:

- Loading a LSP file in all drawings
- Loading a LSP file on demand

Sample files:

9 - Loading AutoLISP Files.lsp, autoload_ex1.lsp, and autoload_ex2.lsp

Leveraging ActiveX/COM

vlax-import-type-library function

- Imports a third-party ActiveX/COM library for use in AutoLISP
- Hundreds and thousands of ActiveX/COM libraries out there
- Some popular third-party libraries are:
  - Microsoft Office
  - ObjectDBX
  - Windows FileSystem object

Leveraging ActiveX/COM

Demo:

- Importing dimension styles from a drawing file
- Reading values from an XML file

Sample file:

10 - Leveraging ActiveX_COM.lsp

Managing Files

findfile function

- Validates a file in a specific location
- Search the AutoCAD support file search paths for a file
- Use partial folder paths to look under all support file search paths

vl-mkdir/vl-file-copy function

- *vl-mkdir* function creates a new folder in the OS
- *vl-file-copy* function copies a file from a source location to a destination location

Managing Files

Demo:

- Creating folders and copying files a drawing is dependent on

Sample file:

11 - Managing Files.lsp

Validating Data Types

type function

- Returns a symbol of the data type returned by an expression or assigned to a variable
- Useful in validating the type of data assigned to a variable before it is used; thereby avoiding errors

Demo:

- Checking for a string value

Sample file:

12 - Data_Validation_Debugging.lsp

vl-catch-all-apply function

- Useful in catching an error that might be generated by a function
- Errors can be handled before getting to the error handler
- Error catch can be evaluated and can be used to determine what your custom function should do next

vl-catch-all-error-p/vl-catch-all-error-message functions

- *vl-catch-all-error-p* function is used to determine whether the value from *vl-catch-all-apply* is an error
- *vl-catch-all-error-message* function returns a text string that helps identify the error that occurred

Validating Data Types

Demo:

- Catching an error

Sample file:

12 - Data_Validation_Debugging.lsp

trace/untrace function

- *trace* function enables the tracing of a function
- When a function is being traced the function's name and the arguments it is being passed
- Useful in tracking down potential errors
- *untrace* removes a trace on a function

Validating Data Types

Demo:

- Tracing a function

Sample file:

12 - Data_Validation_Debugging.lsp

atoms_family function

- Outputs the functions and variables defined in the AutoLISP environment
- The return value of the function is a list
- Using the values in the list, it is possible to compare different states of the AutoLISP Environment

Validating Data Types

Demos:

- Exporting the current AutoLISP environment
- Comparing two exported environments

Sample file:

12 - Data_Validation_Debugging.lsp

Final Thoughts and Questions

Final Thoughts and Questions

Implementing the techniques mentioned in this session should

- provide you with new ways of approaching old problems
- improve or introduce new workflows

Closing Remarks

Thanks for choosing this session and hope you got something out of it.

Do not forget to complete the online evaluation.

If you have any further questions contact me via:

email: lee.ambrosius@autodesk.com
twitter: http://twitter.com/leeambrosius

Enjoy the rest of the conference.