AS227014-L

Advanced AutoLISP Lab

Darren Young
Hermanson Company – Seattle / Portland

Learning Objectives

- Gain hands-on knowledge and use of advanced AutoLISP functions
- Get hands-on knowledge of creating Reactors in AutoLISP
- Gain hands-on knowledge of VBA-style programming with VLA functions in AutoLISP Learning
- Get hands-on use of the Visual LISP Editor

Description

This lab will walk students through an introduction to some advanced AutoLISP concepts and practices. Students will get a taste of reactors, VBA-style programming in AutoLISP, and advanced function and processes.

Speaker(s)

Based in Washington state, Darren Young has been a veteran Autodesk University speaker for well over a decade. His unique ability to leverage multiple every day technologies in interesting ways to solve implicated and laborious tasks has been valued by users around the world. Down to earth and approachable, he's always willing to help his peers anytime of the year even outside of Autodesk University. Darren's background includes a wide variety of disciplines such as Construction, Engineering, Manufacturing, LEAN, Information Technology, Computer Programming, Author and Technical Editor. His lectures and labs are not just a training opportunity for others but a venue which connects him personally with users helping him learn as well.

Lab Assistant(s)

Reysteve Garcia – BIM Developer / Acco Engineered Systems, San Francisco, California
David Ronson – Product Manager / eVolve Software, Austin, Texas
Wade Wessels – BIM Support Specialist / Modern Piping, Cedar Rapids, Iowa

Advanced AutoLISP

- (apply 'function list) – Applies a function on an entire list of data
- (mapcar 'function list {list} …etc…) – Applies a function to each item in a List/Lists
- (lambda (arg1 {arg2}, …etc…) – Defines an anonymous/unnamed function
- (function (lambda (arg1 {arg2}, …etc…) – Protects an anonymous/unnamed function

Recursion

A function defined in a manner in which it also can call itself.

- Useful for processing nested data of unknown nesting levels
  (e.g. Looking for files in folders)
- When a function needs to be repeated a number of times based on it's result
  (e.g Mathematical Factorial)

Reactors

Code which is executed during certain evens (e.g. Actions cause a reaction)

Basic steps for creating reactor code:

1. Write function you want to execute during an event (always takes 2 arguments – reactor that was called, associated data)
2. Find/determine reactor need
3. Fine/determine event reaction needed
4. Determine how reactors get added (persistent/transient, etc.) and mange code as required
5. Verify reactor and callback aren't already loaded
6. Load reactor into environment

Reactor	Event Reaction
`vlr-acdb-reactor`	`:vlr-objectAppended`
	`:vlr-objectUnAppended`
	`:vlr-objectReAppended`
	`:vlr-objectOpenedForModify`
	`:vlr-objectModified`
	`:vlr-objectErased`
	`:vlr-objectUnErased`
`vlr-command-reactor`	`:vlr-unknownCommand`
	`:vlr-commandWillStart`
	`:vlr-commandEnded`
	`:vlr-commandCancelled`
	`:vlr-commandFailed`
`vlr-deepclone-reactor`	`:vlr-beginDeepClone`
	`:vlr-beginDeepCloneXlation`
	`:vlr-abortDeepClone`

	:vlr-endDeepClone
vlr-docmanager-reactor	:vlr-documentCreated
	:vlr-documentToBeDestroyed
	:vlr-documentLockModeWillChange
	:vlr-documentLockModeChangeVetoed
	:vlr-documentLockModeChanged
	:vlr-documentBecameCurrent
	:vlr-documentToBeActivated
	:vlr-documentToBeDeactivated
vlr-dwg-reactor	:vlr-beginClose
	:vlr-databaseConstructed
	:vlr-databaseToBeDestroyed
	:vlr-beginDwgOpen
	:vlr-endDwgOpen
	:vlr-dwgFileOpened
	:vlr-beginSave
	:vlr-saveComplete
vlr-dxf-reactor	:vlr-beginDxfIn
	:vlr-abortDxfIn
	:vlr-dxfInComplete
	:vlr-beginDxfOut
	:vlr-abortDxfOut
	:vlr-dxfOutComplete
vlr-editor-reactor	:vlr-beginClose
	:vlr-beginDxfIn
	:vlr-abortDxfIn
	:vlr-dxfInComplete
	:vlr-beginDxfOut
	:vlr-abortDxfOut
	:vlr-dxfOutComplete
	:vlr-databaseToBeDestroyed
	:vlr-unknownCommand
	:vlr-commandWillStart
	:vlr-commandEnded
	:vlr-commandCancelled
	:vlr-commandFailed
	:vlr-lispWillStart
	:vlr-lispEnded
	:vlr-lispCancelled
	:vlr-beginDwgOpen
	:vlr-endDwgOpen
	:vlr-dwgFileOpened
	:vlr-beginSave
	:vlr-saveComplete
	:vlr-sysVarWillChange
	:vlr-sysVarChanged

vlr-insert-reactor	:vlr-beginInsert
	:vlr-beginInsertM
	:vlr-otherInsert
	:vlr-endInsert
	:vlr-abortInsert
vlr-linker-reactor	:vlr-rxAppLoaded
	:vlr-rxAppUnLoaded
vlr-lisp-reactor	:vlr-lispWillStart
	:vlr-lispEnded
	:vlr-lispCancelled
vlr-miscellaneous-reactor	:vlr-pickfirstModified
	:vlr-layoutSwitched
vlr-mouse-reactor	:vlr-beginDoubleClick
	:vlr-beginRightClick
vlr-object-reactor	:vlr-cancelled
	:vlr-copied
	:vlr-erased
	:vlr-unerased
	:vlr-goodbye
	:vlr-openedForModify
	:vlr-modified
	:vlr-subObjModified
	:vlr-modifyUndone
	:vlr-modifiedXData
	:vlr-unappended
	:vlr-reappended
	:vlr-objectClosed
vlr-sysvar-reactor	:vlr-sysVarWillChange
	:vlr-sysVarChanged
vlr-toolbar-reactor	:vlr-toolbarBitmapSizeWillChange
	:vlr-toolbarBitmapSizeChanged
vlr-undo-reactor	:vlr-undoSubcommandAuto
	:vlr-undoSubcommandControl
	:vlr-undoSubcommandBegin
	:vlr-undoSubcommandEnd
	:vlr-undoSubcommandMark
	:vlr-undoSubcommandBack
	:vlr-undoSubcommandNumber
vlr-wblock-reactor	:VLR-wblockNotice
	:VLR-beginWblockPt
	:VLR-beginWblockId
	:VLR-beginWblock
	:VLR-endWblock
	:VLR-beginWblockObjects
vlr-window-reactor	:vlr-docFrameMovedOrResized
	:vlr-mainFrameMovedOrResized

vlr-xref-reactor	:VLR-beginAttach
	:VLR-otherAttach
	:VLR-abortAttach
	:VLR-endAttach
	:VLR-redirected
	:VLR-comandeered
	:VLR-beginRestore
	:VLR-abortRestore
	:VLR-endRestore
	:VLR-xrefSubcommandBindItem
	:VLR-xrefSubcommandAttachItem
	:VLR-xrefSubcommandOverlayItem
	:VLR-xrefSubcommandDetachItem
	:VLR-xrefSubcommandPathItem
	:VLR-xrefSubcommandReloadItem
	:VLR-xrefSubcommandUnloadItem

ActiveX

ActiveX programming allows Visual Basic (VB/VBA) styles of programming from AutoLisp.

Call **`(vl-load-com)`** to load ActiveX functions into memory.

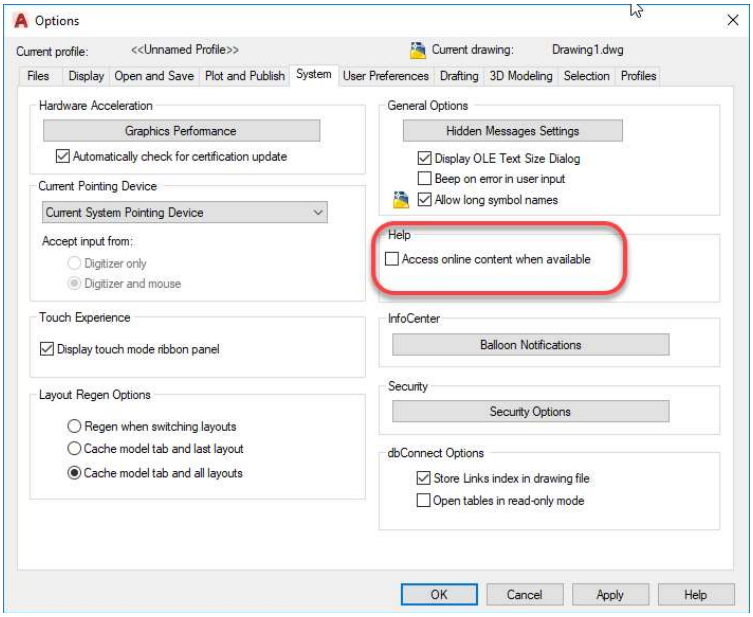
ActiveX programming references objects inside AutoCAD according to an “Object Model”. You can find documentation on this model here…

https://tinyurl.com/Acad2019-VBAModel

For detailed documentation on the ActiveX Object Model, install the AutoCAD Offline Help…

https://tinyurl.com/Acad2019-OfflineHelp

Disable online access to Help using AutoCAD Options…



You’ll now have Access to the ActiveX Reference Guide which contains a lot more information on ActiveX.

IF you want to try VBA (as opposed to AutoLISP) you’ll want to install the VBA Enabler from here…

https://tinyurl.com/AcadVBAInstaller

Determining AutoLISP Function names…

- Creating Objects
  - ActiveX Help says use “AddLine” method
  - LISP function would be “vla-addline”
  - Use Help documentation on ActiveX method to determine required data line start/end points for a line.
- Setting Properties
  - ActiveX Help says “StartPoint” property
  - LISP function would be “vla-put-startpoint” (assuming read/write property)
- Reading Properties
  - ActiveX Help says “Length” property
  - LISP function would be “vla-get-length”
- ActiveX Lisp functions take 1 extra argument besides what ActiveX documentation says
  - ActiveX uses “**Object**.Method”
  - LISP uses (vla-function **<object>** <arg1> <arg2>…)

Use VLAX- functions and constants to help work with ActiveX “Objects” vs traditional AutoLISP types like points, entities, etc. Here are some common ones you’ll use frequently…

Functions…

```
(vlax-ename->vla-object)
(vlax-vla-object->ename)
(vlax-get-acad-object)
(vlax-make-safearray)
(vlax-safearray-fill)
(vlax-make-variant)
(vlax-property-available-p)
(vlax-write-enabled-p)
(vlax-3d-point)
```

Constants…

```
vlax-vbEmpty     (0)
vlax-vbNull      (1)
vlax-vbInteger   (2)
vlax-vbLong      (3)
vlax-vbSingle    (4)
vlax-vbDouble    (5)
vlax-vbString    (8)
vlax-vbObject    (9)
vlax-vbBoolean   (11)
vlax-vbArray     (8192)
```