SD20507 - Deploy and Support AutoLISP Programs Like a Pro

Lee Ambrosius
Principal Learning Experience Designer
Twitter: @LeeAmbrosius

Where Am I and Who Should Be Here

You are in session:

SD20507 - Deploy and Support AutoLISP Programs Like a Pro

You should know:

AutoCAD 2017 (or AutoCAD 2012 and later)

You should want to:

- Implement custom help and improve support
- Deploy custom programs with less stress

Who Am I?

My name is Lee Ambrosius

- Principal Learning Experience Designer at Autodesk
- Work on the Customization, Developer, and CAD Administration documentation
- Customizing and programming AutoCAD for about two decades
- Author of the AutoCAD Customization Platform book series published by Wiley

My job in a nutshell:

I document the present and past AutoCAD releases for the future

What You Will Learn Today

By the end of this session, you will know how to:

- Create and implement custom help topics
- Support multiple languages
- Deploy programs and define plug-in bundles
- Trust and digitally sign AutoLISP program files

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

Ready, Set, and Run…

Overview

- My assumption:
  Everyone, or most, here already deploy and support AutoLISP programs to some extent.

- My hope:
  Show you new approaches that can
  - Lead to improvements with the user experience you deliver
  - Make things easier for you

Overview

- Who knew writing custom programs was just the beginning?
- After writing a custom program, you most likely wanted to share it
- Sharing is commonly referred to as "deploying"
- Deploying a custom program, as you most likely learned
  - Isn't like sending an email
  - It takes courage, as everyone has an opinion
  - Results in having less time in the day to do other things

Overview

- Deployment is not
  - Linear
  - Same for all

Overview

- Deployment can be affected by:
  - Company size (individual, small, medium, large)
  - Location (local or remote)
  - Make-up of user base (techie vs non-techie; multinational)

- Support is often initially overlooked when deploy custom programs:
  - Knowledge transfer and training
  - Deployment/installing
  - Troubleshooting

Overview

Knowledge

- Training sessions (group or 1-1)
- Newsletters
- Custom help that is integrated or stand-alone

Deployment

- Use global command & options
- Locating program files
- Loading custom program files
- Trust locations

Troubleshoot

- Test, Debug, Repeat
- Patch program files
- Log, catch, handle errors
- Trace functions

Implement Custom Help

Implement Custom Help

- Comes in all different sizes and shapes
  - Prompt and error messages
  - Listing of exposed commands or functions
  - Information about command options or expected values
  - Concepts topics; When would I?
  - Tutorials; How do I?

Implement Custom Help

- Command prompts should be
  - Short
  - Conform to the AutoCAD standards
  - Complex and multiple options might be best broken into multiple prompts

- Error messages should be
  - Short and informative
  - Long explanations should be in the help documentation and not the custom program
  - Disruptive when appropriate; *soft* vs *hard* error messages

Implement Custom Help

- Explanatory documentation can be
  - Provided online and/or offline
  - Integrated into the AutoCAD workflow
  - Comprise of
    - Loose web (HTML, JS, CSS) and image files
    - Compiled Help (CHM) files
    - WinHelp (HLP) files *(Obsolete format)*

- CHM files can be created with Microsoft HTML Help Workshop

Implement Custom Help

- These functions can be used to integrate help documentation in the command workflow:
  - `HELP`
  - `SETFUNHELP`

- Help file format being displayed by `HELP` and `SETFUNHELP` determines whether the help is opened in:
  - Main product help window
    -or-
  - In its own application window

Implement Custom Help

- Other documentation formats can be displayed with the `STARTAPP` function
  - ASCII text (TXT) file
  - Rich Text Format (RTF) file
  - Microsoft Word (DOC/DOCx) document file
  - Portable Document Format (PDF) file

Demo:

2 - help setfunhelp and startapp.lsp

Support Multiple Languages

Support Multiple Languages

- Global design firms and teams have their set of challenges
  - Time zone differences
  - Skillsets
  - Local and industry knowledge
  - Spoken/written language

- Spoken/written language can be a barrier when supporting custom programs

Support Multiple Languages

- These items affect the support for multiple languages
  - Prompt strings, keywords, and error messages
  - Dialog boxes implemented using DCL
  - `COMMAND` function and scripts
  - Commands defined with the `DEFUN` and `VLAX-ADD-CMD` functions
  - Macros in a loaded CUI/CUIx file
  - Help documentation

Support Multiple Languages

- Users have a better experience when using custom programs localized in their native language

- Prompts, error messages, and documentation can be localized using
  - Machine translation
  - Manually by a linguistic

- Various ways to store and display localized strings can vary
  - In the source code
  - In an external data file

Support Multiple Languages

- Need to identify the product language which is done via the *product key*

- Product key is
  - Stored in the Windows Registry
  - Obtained with the `VLAX-PRODUCT-KEY` function

- Last three letters of the product key identify product language

Support Multiple Languages

```
;; AutoCAD 2017 - English
(vlax-product-key)
"Software\\Autodesk\\AutoCAD\\R21.0\\ACAD-0001:409"

;; AutoCAD 2017 - French
(vlax-product-key)
"Software\\Autodesk\\AutoCAD\\R21.0\\ACAD-0001:40C"
```

Demo:

3 - localize text string example.lsp

Support Multiple Languages

- Commands have two names
  - Local
  - Global
- Global command names are
  - Same as the English language commands
  - Access by prefixing a command name with an (_) underscore
  - Use with the `COMMAND` function, scripts, and command macros
- Use `GETCNAME` function to identify a global command name

Support Multiple Languages

```
;; AutoCAD 2017 - English
Command: (getcname "LINE")
"_LINE"
Command: (getcname "LIGNE")
nil

;; AutoCAD 2017 - French
Commande: (getcname "LINE")
nil
Commande: (getcname "LIGNE")
"_LINE"
```

Support Multiple Languages

- `DEFUN` function defines commands with the **same** local and global name
- `VLAX-ADD-CMD` function can be used to define commands with the **different** local and global name
- Commands defined with `VLAX-ADD-CMD` can be used with the `COMMAND` function

Demo:

3 - defun and vlax-add-cmd example.lsp

Support Multiple Languages

- Similar to command names, options have global names
  - Same as the English language option names
  - Access by prefixing an option name with an (_) underscore
  - Use with the `COMMAND` function, scripts, and command macros
- Options of standard AutoCAD commands support global names
- Option names defined with `INITGET` are defined as both local and global, but with the same name
- Can define different local and global names with `INITGET`

Support Multiple Languages

```
;; English keywords example
(initget "Blue White Red Green _Blue White Red Green")
(getkword "\nSpecify color [Blue/White/Red/Green]: ")

;; French keywords example
(initget "blEu blAnc Rouge Vert _Blue White Red Green")
(getkword "\nSpécifiez la couleur [blEu/blAnc/Rouge/Vert]: ")
```

Demo:

3 - initget.lsp

Deploy and Load AutoLISP Program Files

Deploy AutoLISP Program Files

- When deploying custom program files, consider the following:
  - Where will the custom programs be stored and loaded from?
    - Local
    - Network
  - Who will be using the custom programs?
    - Internal
    - External
  - What is the expertise level of the users?
    - Techie
    - Basic computer skills

Deploy AutoLISP Program Files

- Custom programs can be deployed by:
  - Manually copying files to a drive
    - Local
    - Network
  - Automating the copying of files
    - Group policies or script
    - Synchronizing with Box, DropBox, Google Drive
  - Custom installer

Load AutoLISP Program Files

- Custom programs can be loaded into AutoCAD:
  - Manually with the APPLOAD command
  - Automatically using the/a
    - Startup Suite in the APPLOAD command
    - *acad.lsp* and *acaddoc.lsp* files
    - LSP Files node in a CUI/CUIx file
    - MNL file with the same name as a loaded CUI/CUIx file
    - LSP file with `LOAD` and `AUTOLOAD` function statements
    - Plug-in bundle

Specify Support File Search Paths

Specify Support File Search Paths

- Used to help AutoCAD locate program and resource files

- Can be specified with the
  - Options dialog box
  - ACAD environment variable
  - `SupportPath` property of the `AcadPreferencesFiles` object in the AutoCAD ActiveX library
  - AutoCAD installation deployment
  - Plug-in bundle

Specify Support File Search Paths

```
;; Usage (appendSupportPath "c:\\my programs")
(defun appendSupportPath (folderName / curACADPaths)
  (if (vl-file-directory-p folderName)
    (progn
      (setq curACADPaths (getenv "ACAD"))
      (setenv "ACAD" (strcat curACADPaths folderName ";"))
    )
  )
)
```

Demo:

5 - support paths.lsp

Trust Executable Locations

Trust Executable Locations

- Used to identify the locations in which AutoCAD can safely load program (executable) files
- Feature initially added in AutoCAD 2013 SP1
- Some of the program files AutoCAD considers executables are:
  - LSP, FAS, VLX, MNL
  - ARX, DBX, CRX
  - DVB
  - .NET assemblies

Trust Executable Locations

- Added initially in AutoCAD 2013 SP1
- Default trusted folders and subfolders on Windows
  - *C:\Program Files\*
  - *C:\Program Files (x86)\*
- Default trusted folders and subfolders on Mac
  - *~\Applications\*
- Locations should be read-only

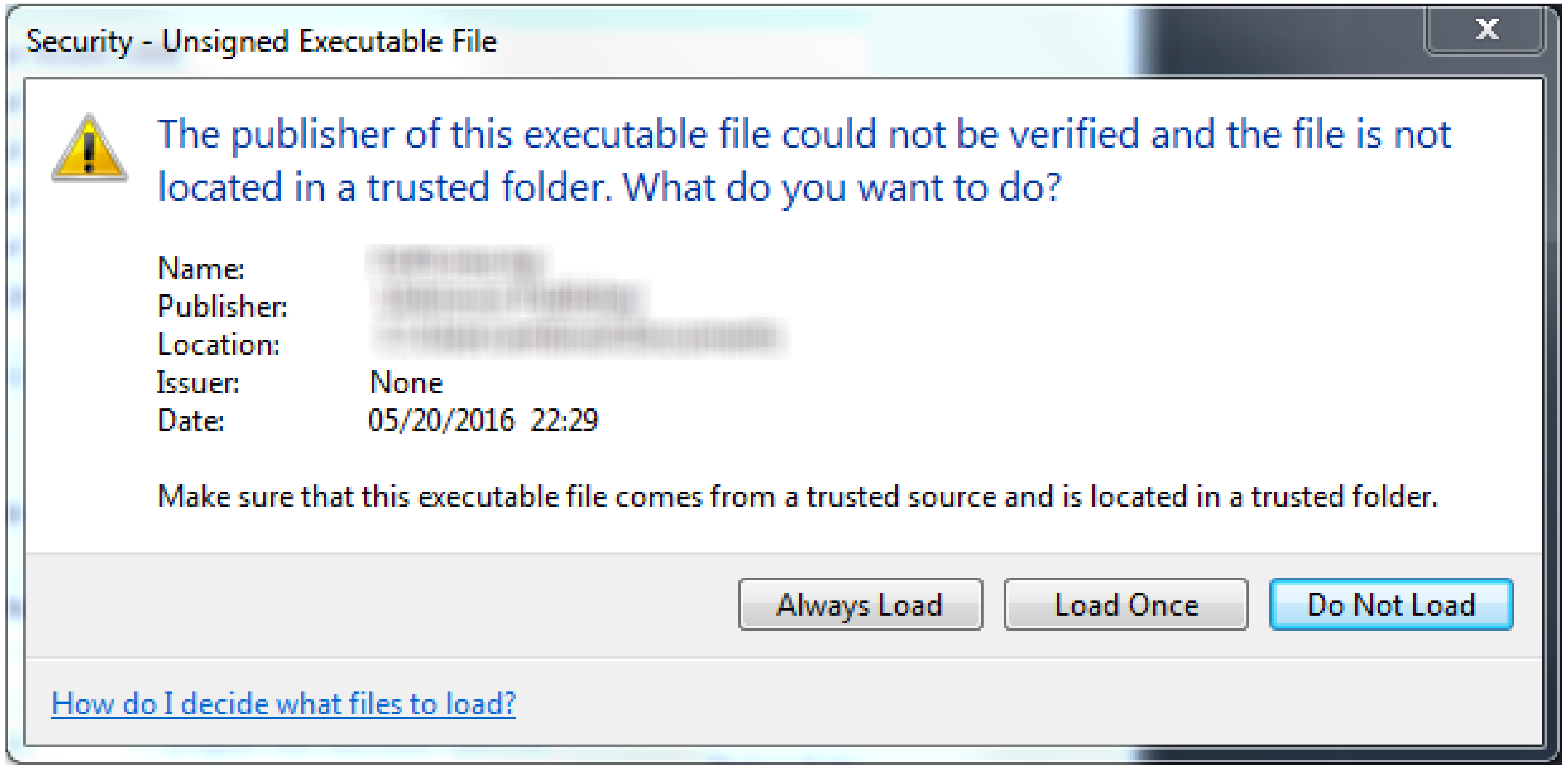
Trust Executable Locations

- Locations can be specified with the
  - Options dialog box
  - TRUSTEDPATHS system variable
  - AutoCAD installation deployment

- SECURELOAD system variable affects the use of trusted locations
  - Recommended to not change the default value

Trust Executable Locations

- User is warned when a file is loaded outside a trusted location



Security - Unsigned Executable File

The publisher of this executable file could not be verified and the file is not located in a trusted folder. What do you want to do?

Name:	[illegible]
Publisher:	[illegible]
Location:	[illegible]
Issuer:	None
Date:	05/20/2016 22:29

Make sure that this executable file comes from a trusted source and is located in a trusted folder.

Trust Executable Locations

```
;; Usage (appendTrustedLocation "C:\\My Programs\\LSPs")
;; Usage (appendTrustedLocation "C:\\My Programs\\LSPs\\..")
(defun appendTrustedLocation (folderName / curTrustedPaths)
  (setq curTrustedPaths (getvar "trustedpaths"))
    ...
)
```

Demo:

6 - trusted locations.lsp

Compile and Protect AutoLISP Files

Compile and Protect AutoLISP Files

- AutoLISP program files don’t need to be compiled
- Compiling or protecting AutoLISP program files does deter people from modifying and copying the source code, but not the file itself
- Program files can be protected using these utilities:
  - Kelvinate (kelvinate.exe)
  - Protect (protect.exe)
  - Visual LISP IDE (VLIDE command)

Compile and Protect AutoLISP Files

- It is recommended to not use Kelvinate and Protect; they are legacy utilities that are not as good as the Visual LISP IDE

- Visual LISP IDE can compile a program file into two formats:
  - VLX – Can contain one or more program and resource files in a single compiled file; supported on Windows only
  - FAS – Represents a single compiled program file

Digitally Sign AutoLISP Program Files

Digitally Sign AutoLISP Program Files

- Digitally signing AutoLISP program files help the user know that the files came from a reputable vendor
- A digitally signed file doesn't necessary mean the file is safe
- To digitally sign a file, you need a/the:
  - Digital certificate from a Certificate Authority (CA)
  - Attach Digital Signatures utility

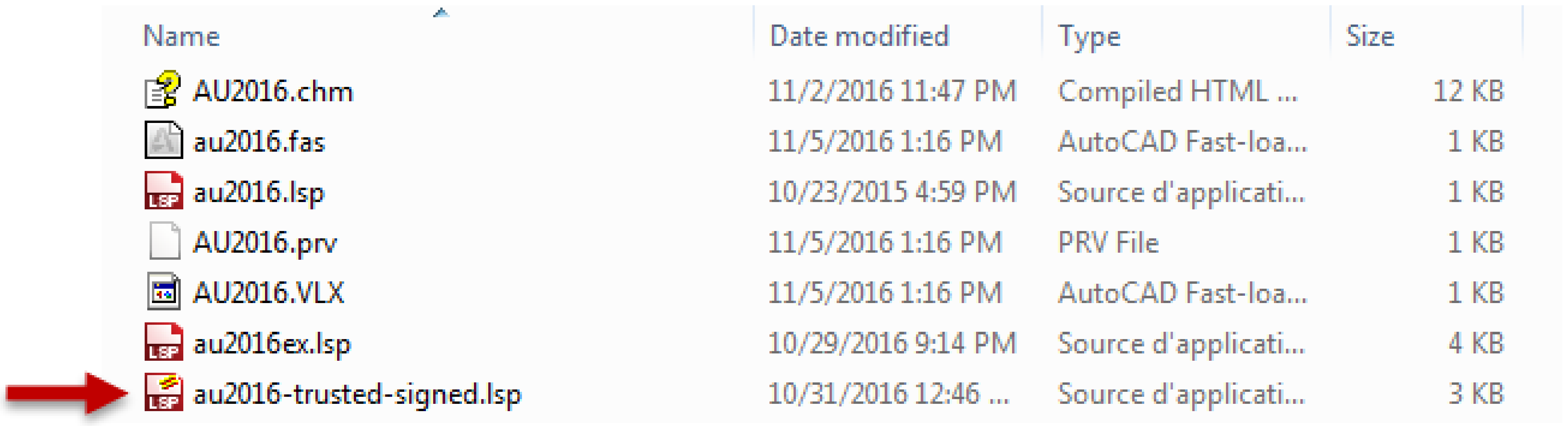
Digitally Sign AutoLISP Program Files

- Not all digital certificates are created equal
  - Most developers use Code Signing Certificates
  - Some use Personal Authentication Certificate

- Some of the common CAs are:
  - DocuSign
  - Comodo
  - GlobalSign
  - IdenTrust

Digitally Sign AutoLISP Program Files

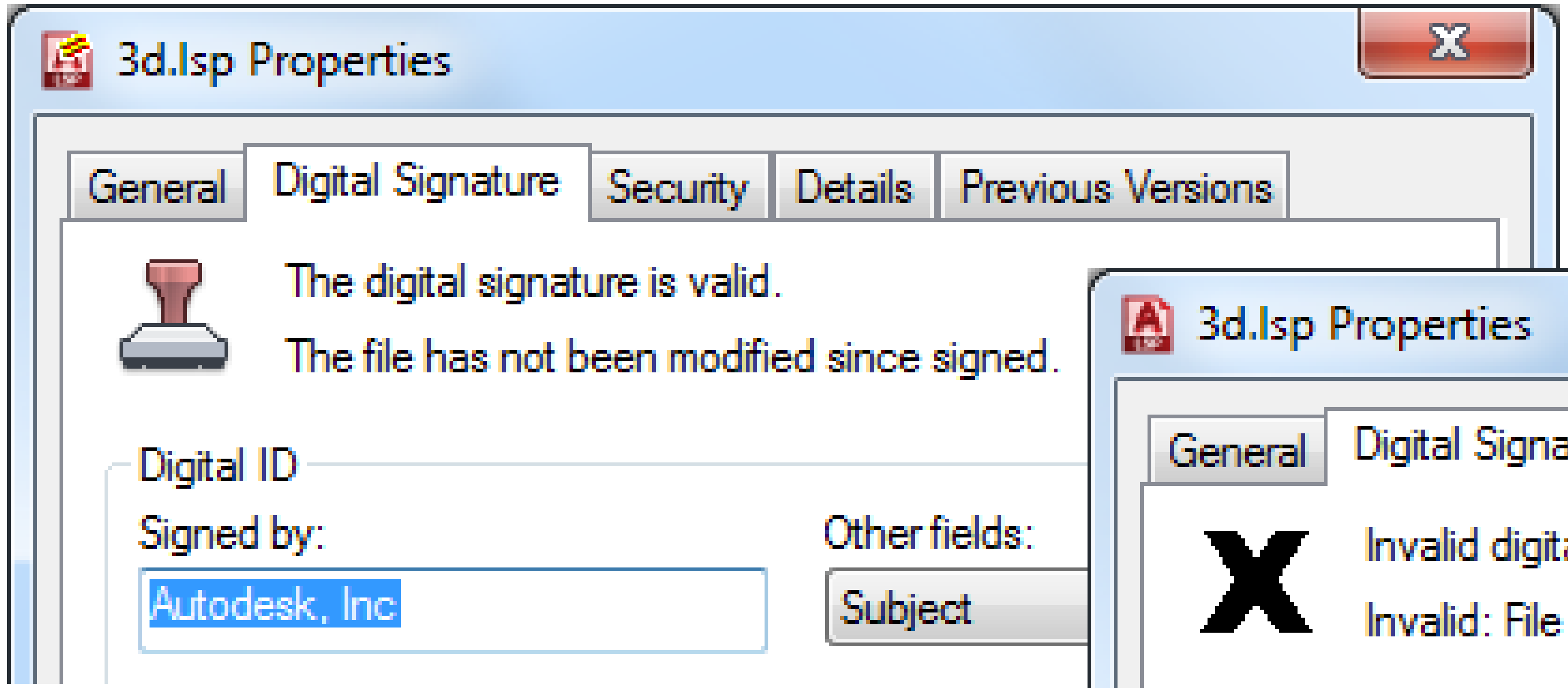
- A digitally signed file can be identified in Windows Explorer or File Explorer



Name	Date modified	Type	Size
AU2016.chm	11/2/2016 11:47 PM	Compiled HTML ...	12 KB
au2016.fas	11/5/2016 1:16 PM	AutoCAD Fast-loa...	1 KB
au2016.lsp	10/23/2015 4:59 PM	Source d'applicati...	1 KB
AU2016.prv	11/5/2016 1:16 PM	PRV File	1 KB
AU2016.VLX	11/5/2016 1:16 PM	AutoCAD Fast-loa...	1 KB
au2016ex.lsp	10/29/2016 9:14 PM	Source d'applicati...	4 KB
au2016-trusted-signed.lsp	10/31/2016 12:46 ...	Source d'applicati...	3 KB

Digitally Sign AutoLISP Program Files

- Properties of a digital signature can be viewed by right-clicking a file and choosing Properties



3d.lsp Properties

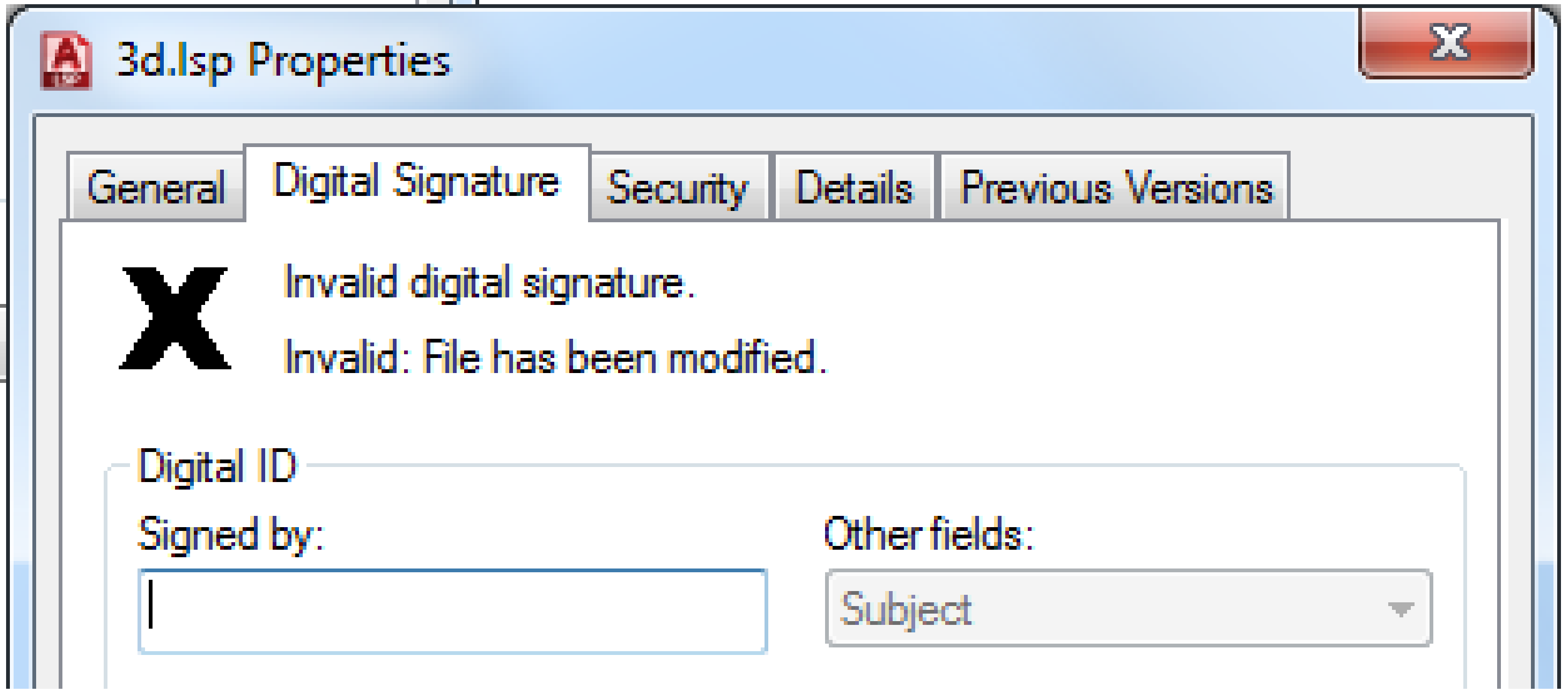
The digital signature is valid.

The file has not been modified since signed.

Digital ID

Signed by: Autodesk, Inc

Other fields: Subject



3d.lsp Properties

Invalid digital signature.

Invalid: File has been modified.

Digital ID

Signed by:

Other fields: Subject

Build Plug-in Bundles for AutoLISP Programs

Build Plug-in Bundles for AutoLISP Programs

- Plug-in bundles:
  - Provide a consistent way to deploy and load LSP files
  - A file and folder structure that contains an XML file named *PackageContents.xml*
- *PackageContents.xml* is
  - Placed in the root folder of a bundle
  - Describes the files in the bundle to AutoCAD and defines how they should be loaded

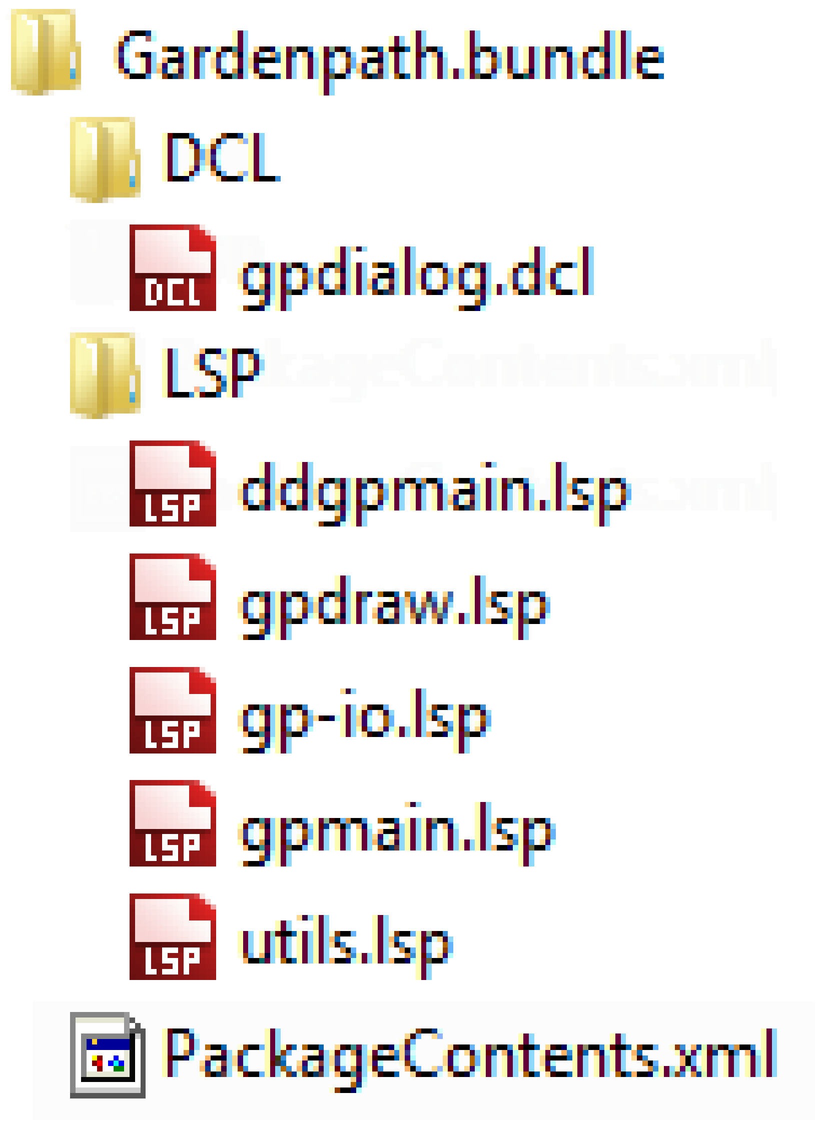
Build Plug-in Bundles for AutoLISP Programs

- Plug-in bundles can help:
  - Control the files that should be loaded by product release
  - Limit the operating systems the custom programs can be loaded
  - Support multiple languages
  - Specify support file search and tool palette paths
  - Implement custom help; CHM or loose HTM/HTML files
  - Set the values of a Windows Registry keys
  - Set the values of system and/or environment variables

Build Plug-in Bundles for AutoLISP Programs

- Example structure of a bundle named GardenPath:

```
Gardenpath.bundle
  |-> DCL
       |-> gpdialog.dcl
  |-> LSP
       |-> ddgpmain.lsp
       |-> gpdraw.lsp
       |-> gp-io.lsp
       |-> gpmain.lsp
       |-> utils.lsp
  |-> PackageContents.xml
```



Gardenpath.bundle
DCL
gpdialog.dcl
LSP
ddgpmain.lsp
gpdraw.lsp
gp-io.lsp
gpmain.lsp
utils.lsp
PackageContents.xml

Build Plug-in Bundles for AutoLISP Programs

- Basic example of a *PackageContents.xml* file:

```
<?xml version="1.0" encoding="utf-8"?>
<ApplicationPackage
    SchemaVersion="1.0"
    AppVersion="1.0"
    Name="AU2016 IT20496-L"
    Description="AU2016 Example for session IT20496-L."
    Author="HyperPics, LLC"
    ProductCode="{45F619FE-E286-4C4E-8134-B50E8DFC23E3}"
>
```

Build Plug-in Bundles for AutoLISP Programs

```
    <CompanyDetails
       Name="HyperPics, LLC"
       Url="http://www.hyperpics.com"
    />
<Components Description="Windows and Mac OS operating systems">
   <RuntimeRequirements
      OS="Win32|Win64|Mac"
      SeriesMin="R19.0"
      Platform="AutoCAD*"
   />
```

Build Plug-in Bundles for AutoLISP Programs

```
      <ComponentEntry Description="Your custom file"
          AppName="AU2016Examples"
          Version="1.0"
          ModuleName="./au2016.lsp">
       </ComponentEntry>
     </Components>
   </ApplicationPackage>
```

Build Plug-in Bundles for AutoLISP Programs

- Access the AutoCAD Online Help system for more information on the *PackageContents.xml* file.

- **Note:** The ProductCode value (GUID) must be unique for each bundle. - *http://www.guidgenerator.com/*

- A bundle is deployed by copying all the files and folders of a bundle to one of these folders:
  - All Users Profile folder
  - User Profile folder

Build Plug-in Bundles for AutoLISP Programs

Trusted and recommended locations

- Windows 7 and later:
  %PROGRAMFILES%\Autodesk\ApplicationPlugins
  %PROGRAMFILES(x86) %\Autodesk\ApplicationPlugins
- Mac OS X:
  ~/Applications/Autodesk/ApplicationAddins

Build Plug-in Bundles for AutoLISP Programs

Other supported locations, but they are not trusted by default

- Windows 7 and later:
  %ALLUSERSPROFILE%\Autodesk\ApplicationPlugins
  %APPDATA%\Autodesk\ApplicationPlugins
- Mac OS X:
  ~/Autodesk/ApplicationAddins

Demo:

SD20507.bundle

SD20507 - Advanced.bundle

Troubleshoot and Debug AutoLISP Files

Basic Debugging

- Core AutoLISP does not provide specific debugging functions
- During execution these functions can display information
  - `ALERT`
  - `PRINC`
  - `PROMPT`

Basic Debugging

- In addition to `PRINC`, these functions can write values/messages out to a file
  - `PRIN1`
  - `PRINT`

Demo:

10 - debug - basic.lsp

10 - debug - custom.lsp

Tracing Functions

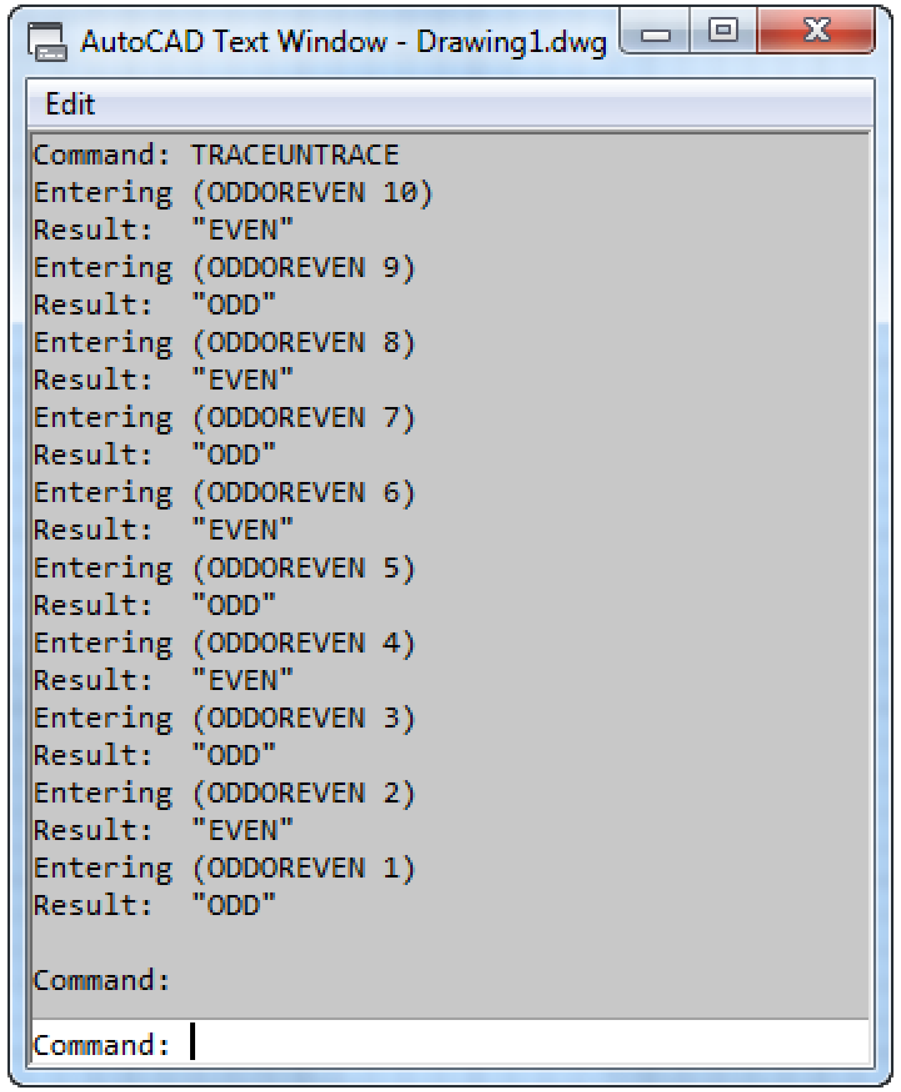
- Core AutoLISP provides a few functions to monitor the usage of a function
  - `TRACE`
  - `UNTRACE`
- When tracing is enabled, you can see:
  - The values passed to the function
  - Results from the function

Tracing Functions

- Results of tracing a function named OddOrEven

Demo:

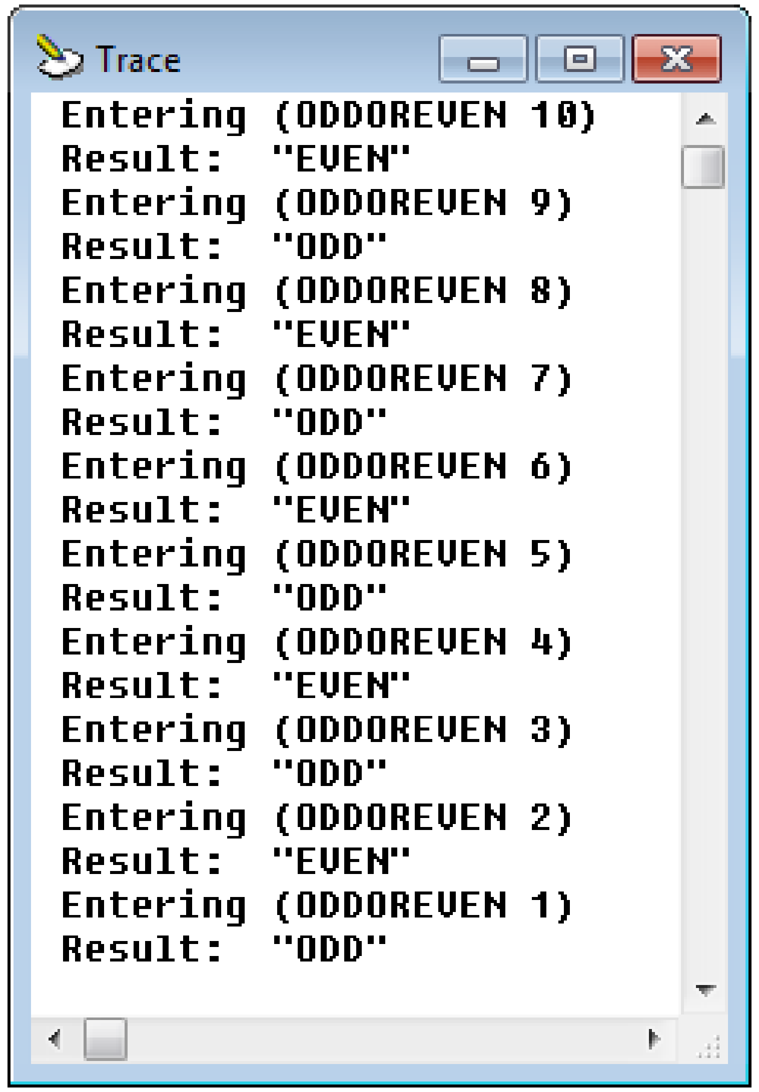
10 - trace and untrace.lsp



```
Command: TRACEUNTRACE
Entering (ODDOREVEN 10)
Result:  "EVEN"
Entering (ODDOREVEN 9)
Result:  "ODD"
Entering (ODDOREVEN 8)
Result:  "EVEN"
Entering (ODDOREVEN 7)
Result:  "ODD"
Entering (ODDOREVEN 6)
Result:  "EVEN"
Entering (ODDOREVEN 5)
Result:  "ODD"
Entering (ODDOREVEN 4)
Result:  "EVEN"
Entering (ODDOREVEN 3)
Result:  "ODD"
Entering (ODDOREVEN 2)
Result:  "EVEN"
Entering (ODDOREVEN 1)
Result:  "ODD"

Command:

Command:
```



```
Entering (ODDOREVEN 10)
Result:  "EVEN"
Entering (ODDOREVEN 9)
Result:  "ODD"
Entering (ODDOREVEN 8)
Result:  "EVEN"
Entering (ODDOREVEN 7)
Result:  "ODD"
Entering (ODDOREVEN 6)
Result:  "EVEN"
Entering (ODDOREVEN 5)
Result:  "ODD"
Entering (ODDOREVEN 4)
Result:  "EVEN"
Entering (ODDOREVEN 3)
Result:  "ODD"
Entering (ODDOREVEN 2)
Result:  "EVEN"
Entering (ODDOREVEN 1)
Result:  "ODD"
```

Catching Errors

- Errors are common in programming, it is how you handle them that is key
- Error handlers should be designed to handle the errors you cannot recover from
- `IF` and `COND` functions when used with operators are an important part of performing conditional tests, but are not always enough alone

Catching Errors

- These functions are used to catch an error that might be caused by a function after it is evaluated
  - `VL-CATCH-ALL-APPLY`
  - `VL-CATCH-ALL-ERROR-P`
  - `VL-CATCH-ALL-ERROR-MESSAGE`

Demo:

10 - catch error.lsp

Defining Custom Error Handlers

- Custom error handlers are essential to a great user experience and are often under utilized
- These functions are used to implement custom error handlers
  - `*ERROR*`
  - `*PUSH-ERROR-USING-COMMAND*`
  - `*PUSH-ERROR-USING-STACK*`
  - `*POP-ERROR-MODE*`

Defining Custom Error Handlers

- The exit and quit functions can be called from your AutoLISP program to force it to return to the Command prompt
- VLX projects with separate namespaces can return a message or value from the VLX error handler to the *error* handler using:
  - `VL-EXIT-WITH-ERROR`
  - `VL-EXIT-WITH-VALUE`

Demo:

10 - error handling.lsp

10 - VLX-exit-with.lsp

Grouping and Rolling Back Changes

- The UNDO command allows the grouping of multiple calls to `COMMAND` function into a single operation
- Without groupings, each command is undone one at a time if the U command is used by the user
- All operations that recorded as part of an Undo record are rolled back with a single U command
- Groupings can be helpful if your AutoLISP program fails part way through execution

Grouping and Rolling Back Changes

- Use the following options of the UNDO command to begin and end a grouping
  - `BEgin`
  - `End`

Demo:

10 - undo grouping.lsp

Final Thoughts and Questions

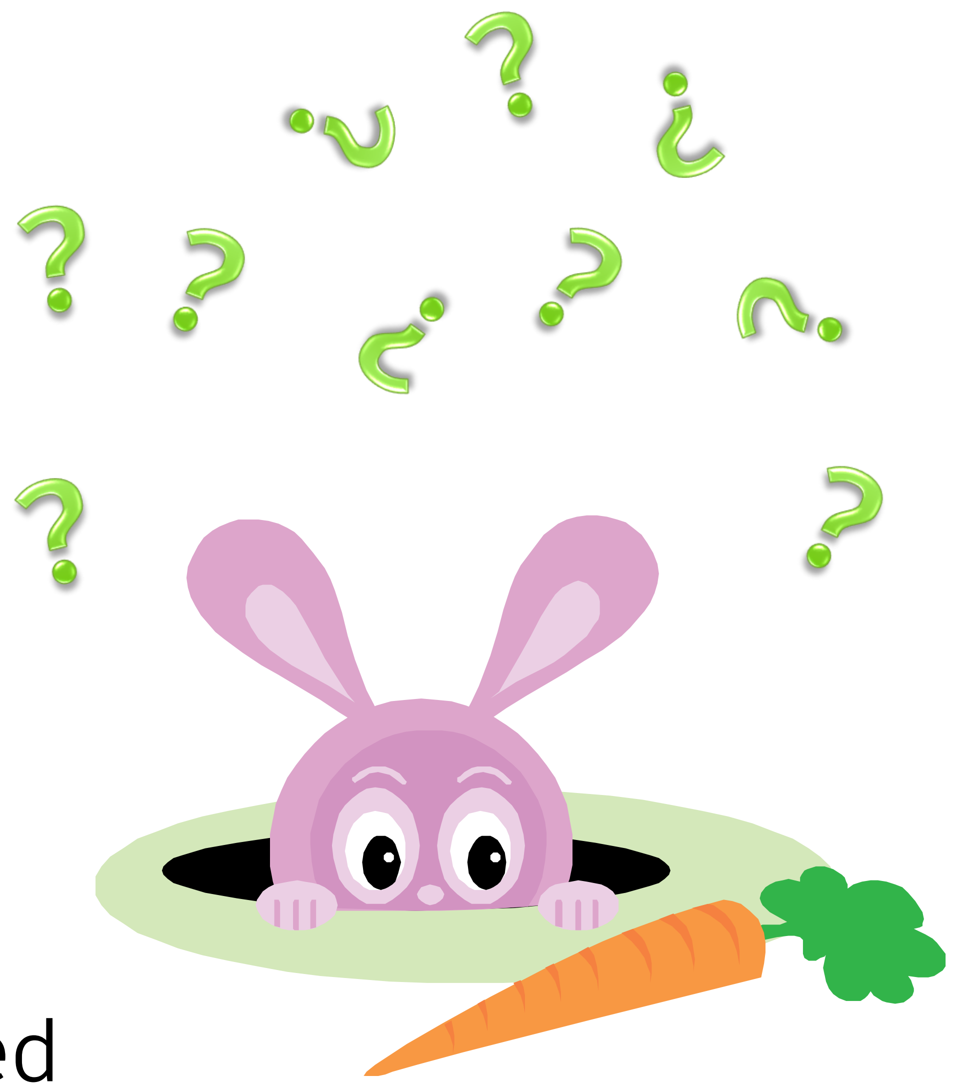
Final Thoughts and Questions

Scripting and programming can:

- Enhance productivity
- Improve or introduce new workflows

Programming has many similarities to the rabbit hole in Lewis Caroll’s *Alice’s Adventures in Wonderland*. Both:

- Are virtually endless
- Hold many mysteries waiting to be discovered



Closing Remarks

Thanks for choosing this session.

Don’t forget to complete this session’s online evaluation.

If you have any further questions, contact me via:

email: lee.ambrosius@autodesk.com
twitter: @leeAmbrosius