AUTODESK UNIVERSITY

SD500025 - Bridging the Gap: Extending AutoLISP with .NET

Lee Ambrosius – Autodesk, Inc.

Principal Learning Experience Designer | @leeambrosius

Who’s this Session For

- Those that want to learn how to:
  - Program with Managed .NET
  - Extend the functionality of AutoLISP programs with .NET
    - Create custom commands and AutoLISP functions
    - Create and develop modern dialog boxes
- What you should already know:
  - AutoCAD 2022 (or AutoCAD 2016 and later)
  - AutoLISP

About the Speaker

- My name is Lee Ambrosius
  - Principal Learning Experience Designer at Autodesk, Inc.
    - Technical writer and data analyst
    - Have You Tried and My Insights for AutoCAD
    - Customization, Developer, and CAD Administration documentation
  - 25 years of customization and programming experience
  - Author of the AutoCAD Customization Platform book series published by Wiley & Sons
- In a nutshell:
  - I document the past and present AutoCAD releases for the future

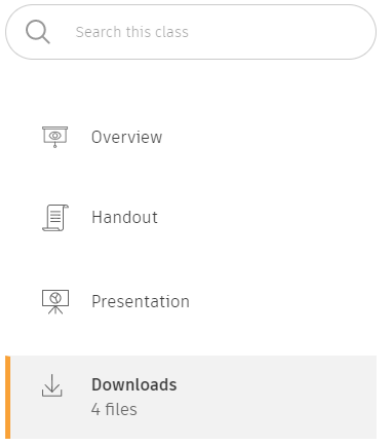
Things You Should Know Before Proceeding

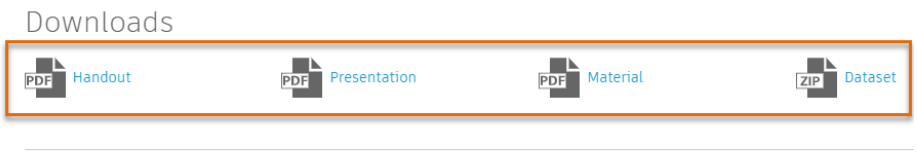
What You Need to Get Started

- For this session, you will need/want:
  - AutoCAD 2022 (or AutoCAD 2016 and later)
  - Experience with AutoLISP programming
  - Materials for this session from the AU website
    - Handout
    - Additional Materials

Setting Up for this Session

- Materials for this session can be obtained by:
  1. Going to the Autodesk University website and searching on this session's ID of **SD500025**.
  2. In the search results, click the entry for this session.
  3. On the session page, click Downloads and then download
     a. Handout
     b. Material





Introduction

What You Will Learn Today

- At the end of this session, you will have learned to:
  - Build and load a .NET assembly
  - Create a command or AutoLISP function
  - Request input from users
  - Create and display a user form

What You Need Before Getting Started

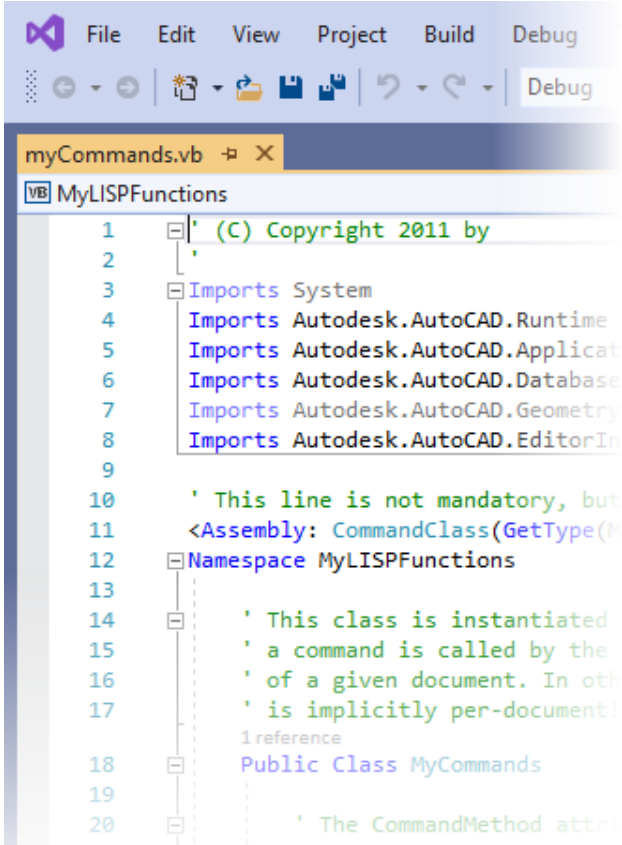
- Development Environment:
  - Visual Studio 2019
  - Visual Studio 2019 Community Edition
- ObjectARX Software Development Kit (SDK)
- AutoCAD 2022 .NET Wizard
- AutoCAD Managed .NET Developer’s Guide

Application Compatibility

- Before starting, you should be aware of the following:
  - Determine which AutoCAD releases and OSs you need to support
    - Affects the libraries needed
    - Affects the .NET Framework needed
    - May need to build for different releases
- .NET is not compatible with AutoCAD for Mac
  - However, ObjectARX can be used with Objective C

Create a VB .NET Project

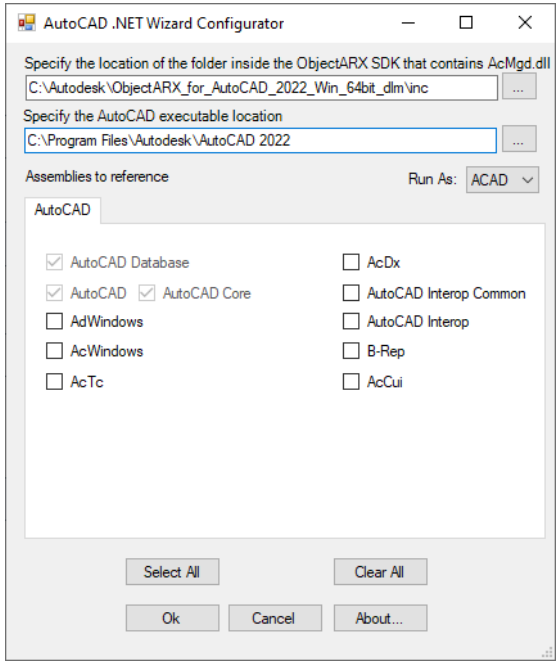
Create a VB .NET Project

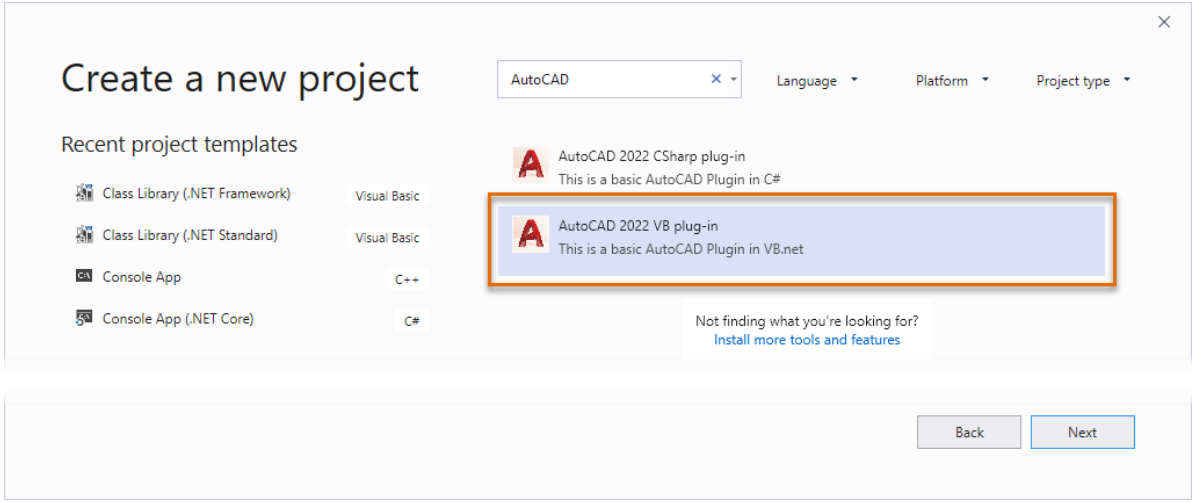


- Need to install:
  - Visual Studio 2019 (or Community Edition)
  - ObjectARX Software Development Kit (SDK)
  - AutoCAD 2022 .NET Wizard
- Create a new project based these templates:
  - VB Class Library (.NET Framework)
  - AutoCAD 2022 VB plug-in

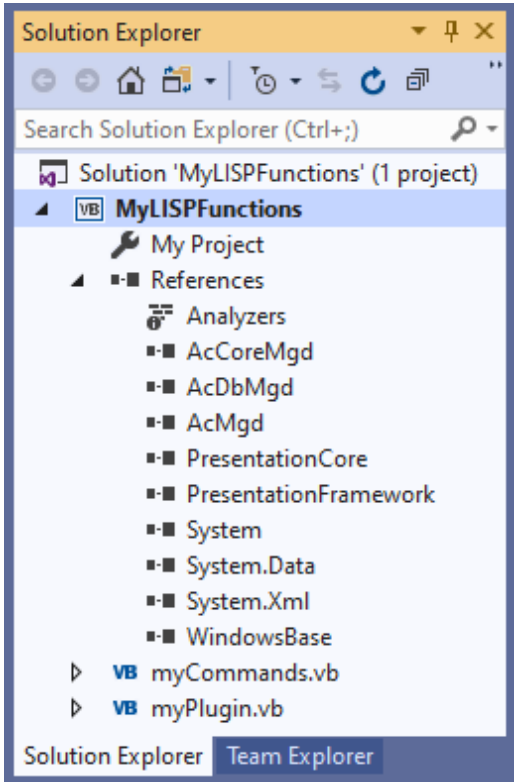
Create a VB .NET Project

- AutoCAD 2022 VB plug-in simplifies the process



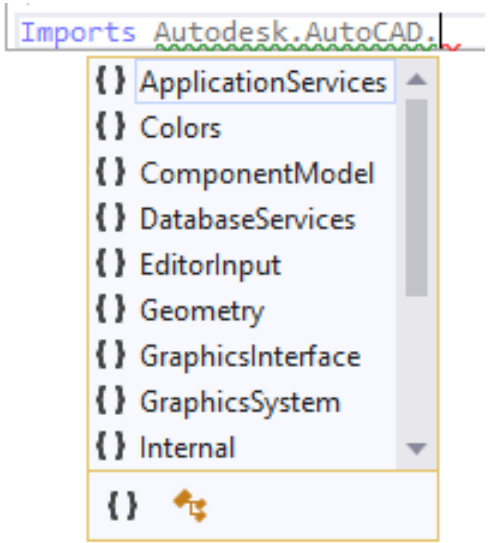


Create a VB .NET Project



- Main assemblies you will need to work with:
  - *AcCoreMgd.dll*
  - *AcDbMgd.dll*
  - *AcMgd.dll*
- Access to the AutoCAD ActiveX/COM libraries:
  - *Autodesk.AutoCAD.Interop.dll*
  - *Autodesk.AutoCAD.Interop.Common.dll*

Create a VB .NET Project



- Assemblies are divided into namespaces to organize methods and properties
- Common namespaces are:
  - Runtime
  - ApplicationServices
  - DatabaseServices
  - Geometry
  - EditorInput

Define a Command

Define a Command

- Create a Public Sub(routine)
- Use the **CommandMethod** attribute and provide the necessary parameters
  - Group name
  - Global command name
  - Local command name
  - Command flags

```
<CommandMethod("MyGroup", "HelloAU", "BonjourAU", CommandFlags.Modal)>
Public Sub HelloAU()
    Dim docEditor As Editor =
           Application.DocumentManager.
           MdiActiveDocument.Editor
    docEditor.WriteMessage("Welcome to AU 2021!")
End Sub
```

Build and Load a .NET Assembly

Build and Load a .NET Assembly

- Project must be built into a DLL for:
  - Debug
  - Release
- Choose a Solution Configuration and click Build menu > Build Solution
- Load a DLL into AutoCAD with the `NETLOAD` command
- DLL can be loaded with AutoLISP:
  - ```
    (command "._NETLOAD"
             "C:/MyTools/MyLISPFunctions.dll")
    ```
    ```

Create and Expose a Function

Create and Expose a Function

- AutoLISP functions are defined similar to custom commands
- Create a Public Function that accepts a single `ResultBuffer` data type
- Use the `LispFunction` attribute
- Function should always return a value of one of these two types
  - `ResultBuffer`
  - `TypedValue`

```
<LispFunction("StringReturn")> _
Public Function StringReturn(ByVal rb As ResultBuffer)

    Return New TypedValue(LispDataType.Text, "My Value")
End Function
```

Create and Expose a Function

- Check to see if the value passed to the function is a/n
  - Standard data type
  - `TypedValue` or array of `TypedValue`

```
If Not rb = Nothing Then
    For Each val As TypedValue In rb
        docEditor.WriteMessage(vbLf & "Type: " & val.TypeCode.ToString())
        If IsNothing(val.Value) = False Then
            docEditor.WriteMessage(vbLf & "Value: " & val.Value.ToString() & vbLf)
        Else
           docEditor.WriteMessage(vbLf & "Value: nil" & vbLf)
        End If
    Next
End If
```

Create and Expose a Function

- `ResultBuffer` in .NET is similar to a List in AutoLISP
- Contains any of the common AutoLISP data types
- Represent Lists and Dotted Pairs
- Each item of a `ResultBuffer` is a `TypedValue` data type
  - TypeCode property indicates data type
  - TypeValue property contains the set value

```
Dim rbRt As New ResultBuffer
rbRt.Add(New TypedValue(LispDataType.Int16, 0))
rbRt.Add(New TypedValue(LispDataType.DottedPair))
rbRt.Add(New TypedValue(LispDataType.Text, "INSERT"))
Return rbRt
```

Access AutoLISP User-defined Variables

Access AutoLISP User-defined Variables

- AutoLISP variables defined with `SETQ` can be accessed with:
  - `GetLispSymbol`
  - `SetLispSymbol`
- `GetLispSymbol` returns the value assigned to a variable
  - Use `GetType` function to determine data type assigned
- `SetLispSymbol` assigns an `Object` data type to a variable

```
Dim lspVal As Object = doc.GetLispSymbol("foo")
Dim typeValue As New TypedValue(LispDataType.Nil)

If IsNothing(lspVal) = False Then
    typeValue = New TypedValue(LispDataType.Text,
                               lspVal.GetType().Name)
End If
```

Request User Input

Request User Input

- User input is handled with `Get*` methods similar to those in AutoLISP
- Each `Get*` method requires the use of a:
  - `PromptOptions*` object controls the method's behavior
  - `PromptResult*` object contains the return value or status
- `Get*` methods are members of the `Editor` object

```
Dim docEditor As Editor =
       Application.DocumentManager.MdiActiveDocument.Editor

Dim pPointOpts As New PromptPointOptions(vbLf +
                    "Specify a point or [Layer/Undo]: ",
                    "Layer Undo")

Dim pPointResult As PromptPointResult =
                    docEditor.GetPoint(pPointOpts)
```

Request User Input

- Biggest advantage of utilizing the `Get*` methods from the .NET API is

  Pressing ESC doesn't terminate your program.

- `PromptResult` allows you to check the status of the Get* function

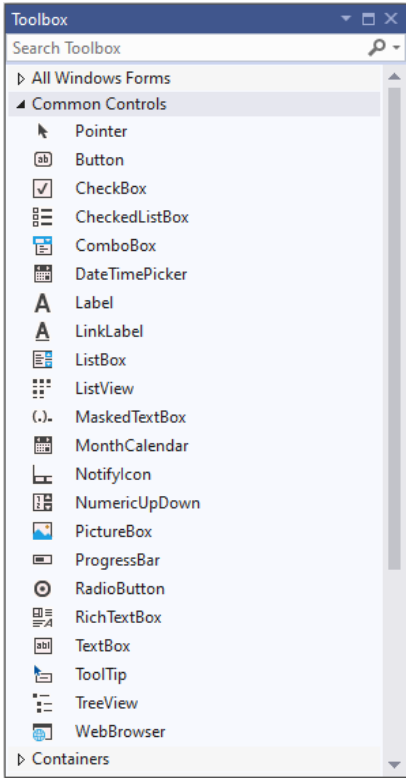
```
If pPointResult.Status = PromptStatus.OK Then
    typeValue = New TypedValue(LispDataType.Point3d, pPointResult.Value)

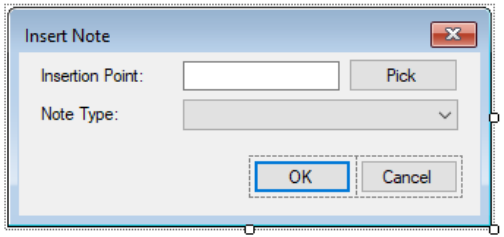
  ' User entered a keyword
ElseIf pPointResult.Status = PromptStatus.Keyword Then
    typeValue = New TypedValue(LispDataType.Text,
                               pPointResult.StringResult)

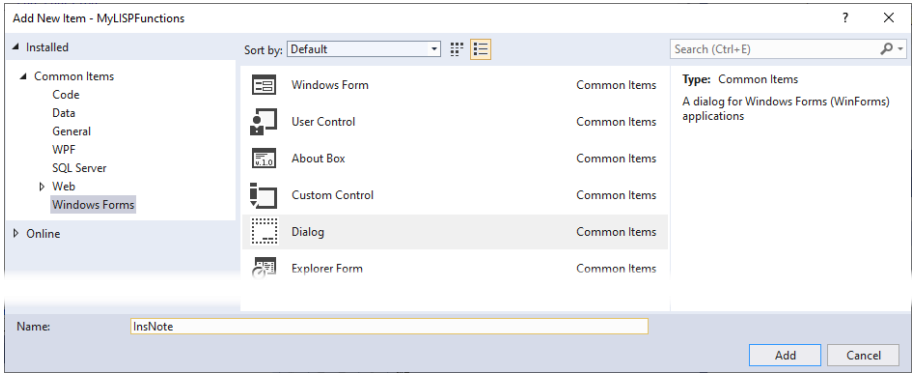
  ' User cancelled the input
ElseIf pPointResult.Status = PromptStatus.Cancel Then
    MsgBox("Input cancelled")
End If
```

Create and Display a Dialog Box

Create and Display a Dialog Box







- .NET forms and dialog boxes:
  - Simpler than DCL
  - Modernized controls and experiences

Final Thoughts

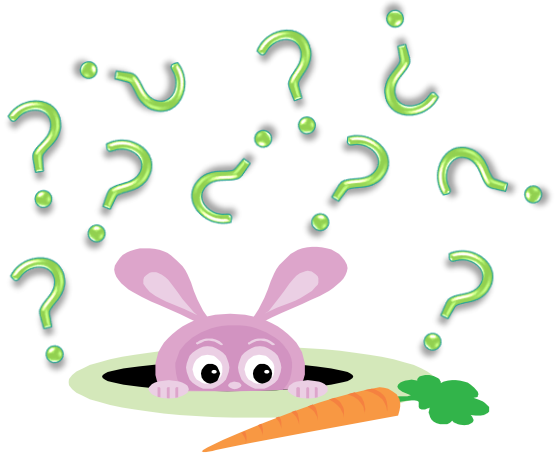
Final Thoughts

- Extending the functionality of AutoLISP:
  - Take advantage of the AutoCAD Managed .NET API
  - Utilize modernized dialog boxes and palettes
- Programming has many similarities to Wonderland in *Lewis Caroll's Alice's Adventures*
  - Both
    - Are virtually endless
    - Hold many mysteries just waiting to be discovered

Final Thoughts

- If you have any further questions,
  - Leave a comment on this session's AU page
  - Feel free to contact me via
    - **email:** lee.ambrosius@autodesk.com
    - **twitter:** @leeambrosius

- Thanks for watching this session!



AUTODESK UNIVERSITY