SD500025

Bridging the Gap: Extending AutoLISP with .NET

Lee Ambrosius
Autodesk, Inc.

Learning Objectives

- Learn how to build and load a .NET project
- Learn how to create a command or AutoLISP function
- Learn about requesting user input
- Learn how to create and display a user form

Description

AutoLISP is a very powerful programming language, but it doesn't offer the same knockout punch that VB.NET/C# does when combined with the AutoCAD .NET API. This class will explore the fundamentals of VB.NET and examine how you can use it to enhance existing AutoLISP routines. During this class, you will learn how to create basic commands for AutoCAD software, and discover functions that you can use with your existing AutoLISP routines. We will show examples of how to work with AutoCAD objects, get input from the user, create a basic form, and pass values between AutoLISP and .NET, plus other general programming concepts. For this class, you should know some AutoLISP, but it will be a beginner-level class for VB.NET and the AutoCAD .NET API.

Speaker(s)

Lee Ambrosius is a Principal Learning Experience Designer at Autodesk, Inc., for the AutoCAD® and AutoCAD LT products on Windows and Mac. He works primarily on the customization, developer, and CAD administration documentation along with the user documentation. Lee has presented at Autodesk University for about 15 years on a range of topics, from general AutoCAD customization to programming with the ObjectARX technology. He has authored several AutoCAD-related books, with his most recent project being *AutoCAD Platform Customization: User Interface, AutoLISP, VBA, and Beyond*. When Lee isn't writing, you can find him roaming various AutoCAD community forums, posting articles on his or the AutoCAD blog, or tweeting information regarding the AutoCAD product.

Twitter: @leeambrosius
Email: lee.ambrosius@autodesk.com
Blog: http://hyperpics.blogs.com

1 Introduction

AutoCAD is an ever changing platform, not only by what it allows a user to do out of the box but what it offers under the hood for those that want to extend it. AutoLISP is the oldest programming option for extending AutoCAD, and really has not changed at its core too much since it was first implemented. There have been a few significant extensions of AutoLISP since it was first introduced, and those were:

- DCL which added the ability to create and display dialog boxes.
- Visual LISP which added an internal editor (or IDE - Integrated Development Environment) with AutoCAD 2000. It was offered as a separate download for AutoCAD R14.

Beyond those two enhancements, it has stayed pretty constant which in the development world is usually not a good sign. While AutoLISP does not seem to be going anywhere in a hurry, it is really not growing much in functionality either. There have been a few native functions added over the past few releases for key features, but not quite to the same level that Visual LISP extended the functionality of AutoLISP.

This session explains how to leverage VB.NET to extend the functionality of AutoLISP by creating custom native functions in secure libraries. While using VLA- and VLAX functions are great, they are also limited in functionality based on if features are exposed in the ActiveX/COM libraries and not all new features are supported through the ActiveX/COM APIs. .NET allows you to access features in the AutoCAD Managed .NET libraries, ActiveX/COM and also functions from ObjectARX if needed.

2 What You Need Before Getting Started

Before you start working with VB.NET and the AutoCAD Managed .NET API, you should obtain the following:

- **ObjectARX Software Development Kit (SDK) –** The ObjectARX SDK contains code samples, project templates, Managed .NET library files, and the AutoCAD Managed .NET Reference Guide. - *https://www.autodesk.com/objectarx*, under the Licensing section, click Download and follow the onscreen instructions.
- **AutoCAD 2022 .NET Wizard –** The AutoCAD 2022 .NET Wizard helps to define the necessary project settings and libraries references that you will need to work with. - *https://www.autodesk.com/developautocad*, click *AutoCAD 2022 .Net Wizards.zip* and save the file to your local drive. Extract the files and run *AutoCAD 2022 dotNET Wizards.msi*.
- **Development Environment –** Visual Studio 2019 is what you will need if you plan on developing applications for AutoCAD 2021 or AutoCAD 2022. If you are working with an earlier release of AutoCAD, you will want to obtain an earlier release of Visual Studio. See the Which Edition of Microsoft Visual Studio to Use (https://help.autodesk.com/view/OARX/2022/ENU/?guid=GUID-450FD531-B6F6-4BAE-9A8C-8230AAC48CB4) topic in the AutoCAD Developer documentation to know which version of Visual Studio and the .NET Framework you might need to use.

  If you do not have Visual Studio available, you can download Visual Studio 2019 Community from *https://visualstudio.microsoft.com/vs/community/*.

- **AutoCAD Managed .NET Developer's Guide –** The .NET Developer's Guide contains information on how to work with the AutoCAD Managed .NET API, and there many code samples that show the different aspects of the .NET API.

  To access the online help, go to: https://www.autodesk.com/autocad-net-developers-guide

Important Terms

The following are important terms in VB.NET that will be mentioned during this session that you might not be aware of:

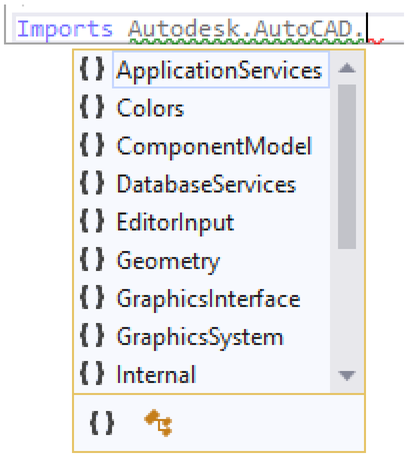
Assemblies – A compiled application that can be loaded into AutoCAD or as part of a project to access parts of the AutoCAD .NET or another API. The AutoCAD .NET API is made up of three primary assemblies, which are:

- **AcDbMgd.dll.** Use when working with objects in a drawing file.
- **AcMgd.dll.** Use when working with the AutoCAD application.
- **AcCui.dll.** Use when working with customization (CUIx) files.

Namespace – A component of a loaded assembly. Namespaces are used to access and organize classes and methods of an API.

- Autodesk is the parent namespace in the AutoCAD .NET API, which contains several other important namespaces. AutoCAD is located under the Autodesk namespace which contains all the namespaces related to AutoCAD as you might have guessed.

  The namespaces located under AutoCAD is where you start seeing the actual structure of the libraries related to AutoCAD.



```
Imports Autodesk.AutoCAD.
```

  The following are the most frequently used namespaces and the ones used in these handouts:

  - **Runtime.** Required to define commands and AutoLISP functions.
  - **ApplicationServices.** Required to work with the application and access open drawings.
  - **DatabaseServices.** Required to work with objects in a drawing.

- **Geometry.** Required to work with data types and methods related to coordinates and other math related tasks used to define geometry.
- **EditorInput.** Required to display messages at the Command prompt and request input from a user.

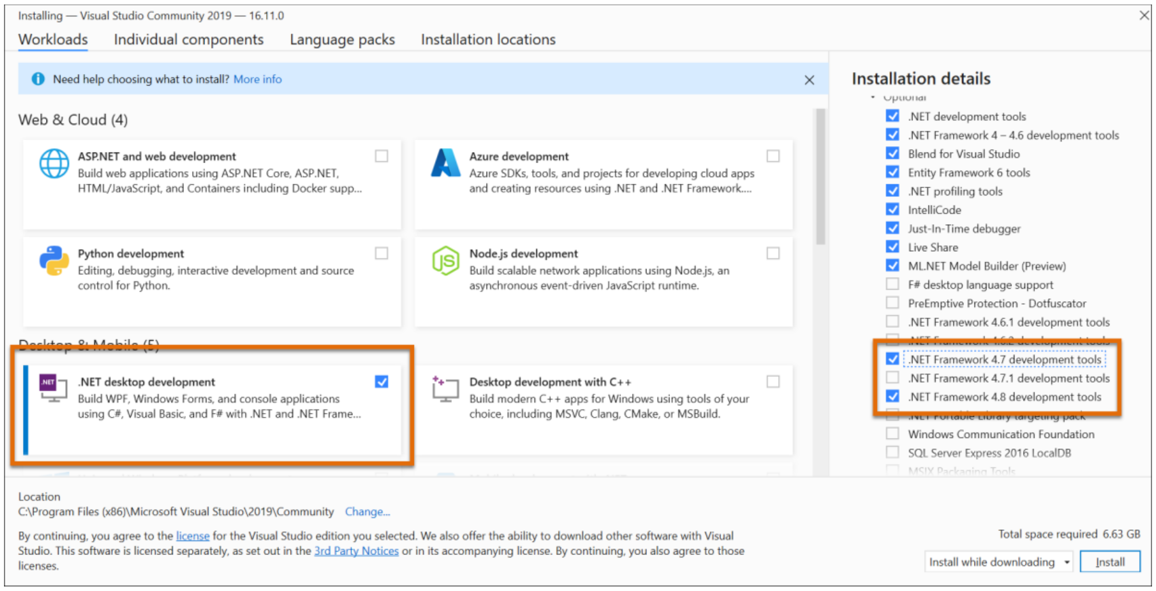
Note: You use the Imports keyword to enable the use of a namespace in your project.

Download and Install Visual Studio 2019 Community Edition

If you don't have a license to Visual Studio 2019, you can download and use Visual Studio 2019 Community for free. Folow these steps to download and install Visual Studio Community:

1. Browse to the Visual Studio Community page on the Microsoft website.

   Go to https://visualstudio.microsoft.com/vs/community/.
2. Click Download Visual Studio.
3. After you download the Visual Studio Installer, click Open File or double-click the file in the Downloads folder.
4. If you are prompted with "Do you want to allow this app to make changes to your device?" message, click Yes.
5. In the Visual Studio Installer message box, click Continue.
6. In the Installing dialog box, under Desktop & Mobile, check .NET Desktop Development.



Installing — Visual Studio Community 2019 — 16.11.0

Need help choosing what to install? More info

Web & Cloud (4)

ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...

Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....

Python development
Editing, debugging, interactive development and source control for Python.

Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

.NET desktop development ✓
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

Desktop development with C++
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

Installation details

Optional
- ✓ .NET development tools
- ✓ .NET Framework 4 – 4.6 development tools
- ✓ Blend for Visual Studio
- ✓ Entity Framework 6 tools
- ✓ .NET profiling tools
- ✓ IntelliCode
- ✓ Just-In-Time debugger
- ✓ Live Share
- ✓ ML.NET Model Builder (Preview)
- ☐ F# desktop language support
- ☐ PreEmptive Protection - Dotfuscator
- ☐ .NET Framework 4.6.1 development tools
- ☐ .NET Framework 4.6.2 development tools
- ✓ .NET Framework 4.7 development tools
- ☐ .NET Framework 4.7.1 development tools
- ✓ .NET Framework 4.8 development tools
- ☐ .NET Portable Library targeting pack
- ☐ Windows Communication Foundation
- ☐ SQL Server Express 2016 LocalDB
- ☐ MSIX Packaging Tools

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community Change...

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 6.63 GB

7. In the Installation Details area, check the following additional components:
   - .NET Framework 4.7
   - .NET Framework 4.8
8. Click Install.
9. When the installer finishes, sign into your Microsoft Account (or create a new one) or click Not Now, Maybe Later. to start Visual Studio.

10. In the Visual Studio, Start with a Familiar Environment dialog box, click Start Visual Studio.
11. Close Visual Studio and the Visual Studio Installer, and then continue to the next sections to install the ObjectARX SDK and the AutoCAD 2022 .NET Wizard.

Download and Install the ObjectARX SDK

You will need to install the ObjectARX 2022 SDK to access the AutoCAD .NET libraries, and the AutoCAD .NET Developer's and Reference Guides. The following steps explain how to download and install the ObvjectARX 2022 SDK:

1. Browse to the ObjectARX section of the AutoCAD Developer's Center website.

   Go to https://www.autodesk.com/developer-network/platform-technologies/autocad/objectarx-license-download.
2. Scroll down and fillout the information for the License & Download Agreement.
3. Click "I Agree" and then click Download.
4. On the AutoCAD Object Downloads page, click the link for your target release of AutoCAD.
5. Once the file has been downloaded, double-click the EXE file and click OK to accept the defaults.

   By default, the SDK files are installed to:

   C:\Autodesk\ObjectARX_for_AutoCAD_2022_Win_64bit_dlm

Download and Install the AutoCAD 2022 .NET Wizard

The AutoCAD 2022 .NET Wizard simplifies the process of creating a new VB .NET project. The exercises in this handout utilize the wizard, so you will want to install it. The following steps explain how to dopwnload and install the AutoCAD 2022 .NET Wizard:

1. Browse to the AutoCAD Developer's Center website.

   Go to https://www.autodesk.com/developer-network/platform-technologies/autocad.
2. Scroll down to the Tools section and click the AutoCAD 2022 DotNet Wizard link.
3. On the GitHub page, click Download.

   The Download can also be found here: https://github.com/ADN-DevTech/AutoCAD-Net-Wizards/blob/ForAutoCAD2022/AutoCADNetWizardsInstaller/AutoCAD_2022_dotnet_wizards.zip
4. Once the file has been downloaded, extract the contents of the ZIP file.

   The contents don't need to be extracted to a specific folder.
5. Open the folder in which you extracted the ZIP file and doble-click the *AutoCADNetWizards.msi* file.
6. If the Windows Protected Your PC message box appears, click More Info and then click Run Anyway.
7. Follow the on-screen prompts and accept all the default values.

3 Compatibility

Projects created with .NET and the AutoCAD Managed .NET API are built for specific releases because of the binary compatibility of the .NET API libraries. So a .NET project created for AutoCAD 2021 might not work with AutoCAD 2020. However, a project built for AutoCAD 2020 may run in AutoCAD 2021 and AutoCAD 2022.

Keep the following in mind when compatibility might be a concern:

- Use the .NET Framework supported by the earliest target release of AutoCAD. AutoCAD will let you know if the .NET application is compatible or not when you attempt to load it. If it is not compatible, you will see a series of error messages at the command line.

  The .NET Framework 4.8 release is recommended for AutoCAD 2021 and AutoCAD 2022. See the Which Edition of Microsoft Visual Studio to Use (https://help.autodesk.com/view/OARX/2022/ENU/?guid=GUID-450FD531-B6F6-4BAE-9A8C-8230AAC48CB4) topic in the AutoCAD Developer documentation to know which .NET Framework release you should use for the target release of AutoCAD.
- Use the .NET and ActiveX/COM libraries for the earliest target release of AutoCAD that you want to support. This will help you to avoid the use of new functions that might only exist in the latest APIs.
- Typically, when upgrading a project from an earlier release, you should only need to reference the new .NET library files and rebuild the application.
- Make sure you reference the correct libraries based on the target OS release. If you are building applications for AutoCAD 2019 and earlier, you will need to have two different projects, one for 32-bit and another for 64-bit applications based on the liubraries you are using.

4 Building a VB.NET Project

Before you can get started with defining your custom functions, you will want to download the sample files for this session and need to create a new project in Visual Studio 2019 or Visual Basic 2019 Community Edition based on your preference.

The exercises in this handout require you to download and setup a folder for this session. These steps explain how to download and extract the contents of this session's sample files:

Download and Setup the Dataset Folder

1. Browse to this session's page on the Autodesk University website.

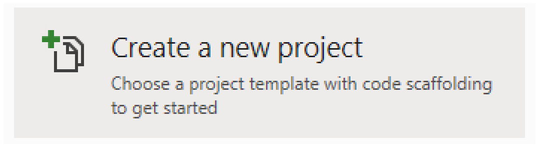
   Go to https://www.autodesk.com/autodesk-university/conference/overview and search on the session ID **SD500025**.
2. In the search results, click the entry for this session.
3. On the session's page, click Downloads and then click Material.

   The dataset should start downloading to your local drive.
4. Once the dataset has been downloaded, extract the contents of the ZIP file to a folder structure similar to the following (just make sure to use the same location throughout):

   C:\Datasets\SD500025

Create a New VB.NET Project with the .NET Wizard

1. Start Microsoft Visual Studio 2019. (Click Start menu > Visual Studio 2019)
2. In Microsoft Visual Studio, click Create a New Project (or click File menu > New Project).

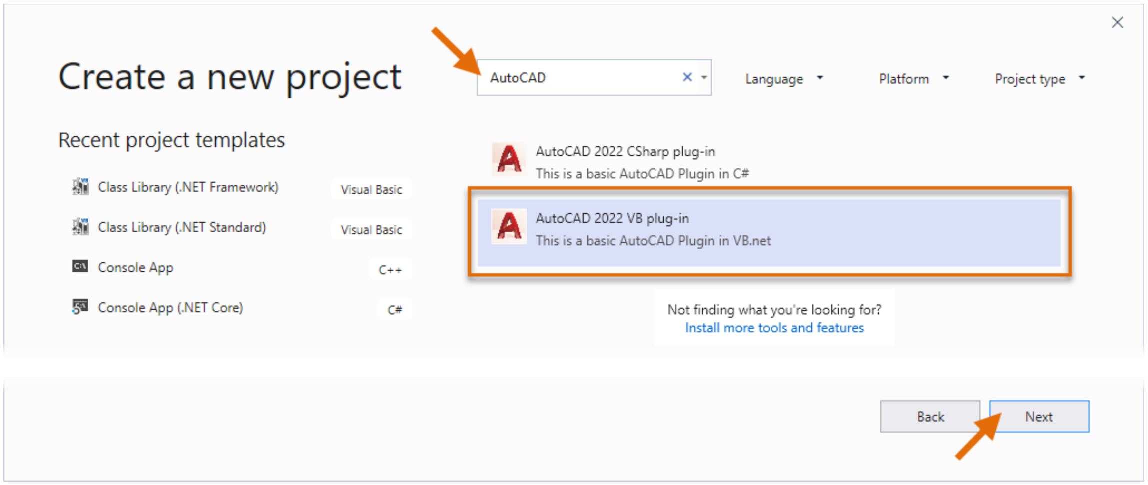

Create a new project
Choose a project template with code scaffolding to get started

3. In the Create a New Project dialog box, in the Search text box, type **AutoCAD**.

   If no matching results are found, clear the Search text box and search for AutoCAD 2022 VB plug-in. If it isn't found, try one of the following:

   - Go to Appendix1: AutoCAD .NET Wizard Not Showing in Visual Studio and try some of the steps there to resolve the problem.
   - Go to Appendix 2: Create a new VB .NET Project without the .NET Wizard and create a project from scratch. It will take a few more steps, but the .NET Wizard isn't required though it simplifies the setup process.

4. Choose AutoCAD 2022 VB plug-in and click Next.


Create a new project
AutoCAD
Language
Platform
Project type
Recent project templates
Class Library (.NET Framework) Visual Basic
Class Library (.NET Standard) Visual Basic
Console App C++
Console App (.NET Core) C#
AutoCAD 2022 CSharp plug-in
This is a basic AutoCAD Plugin in C#
AutoCAD 2022 VB plug-in
This is a basic AutoCAD Plugin in VB.net
Not finding what you're looking for?
Install more tools and features
Back
Next

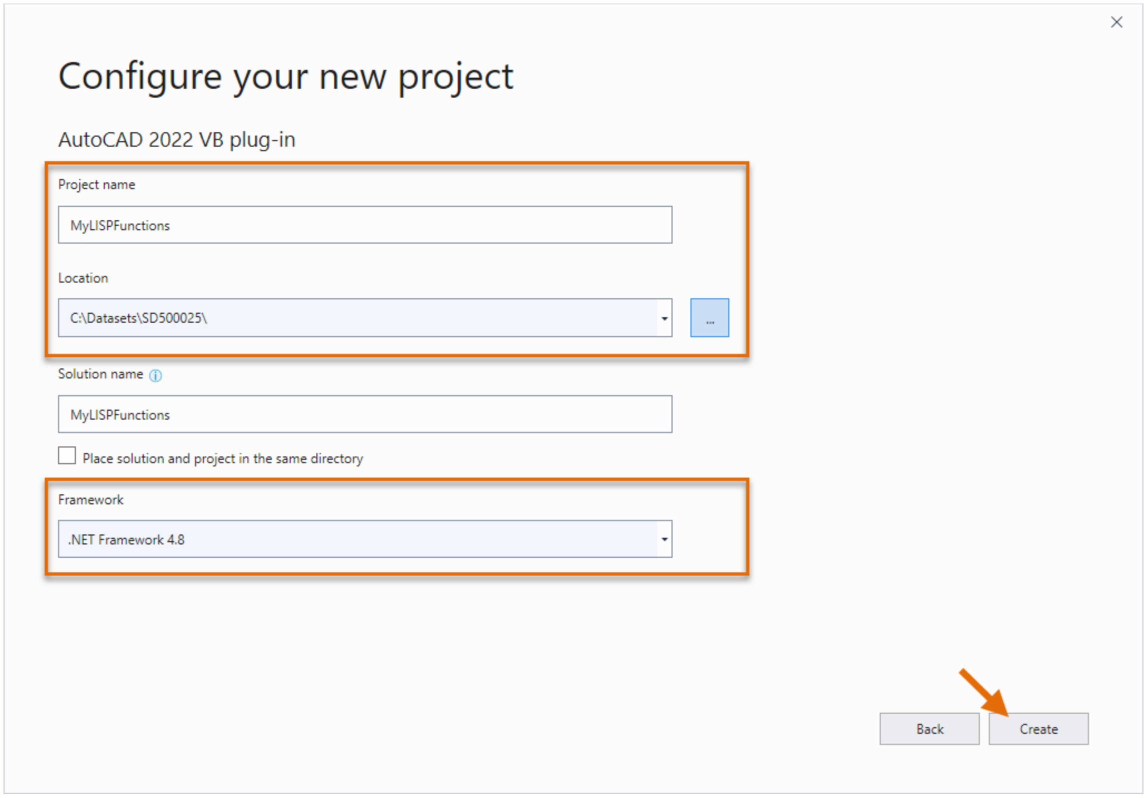
5. In the Configure Your New Project dialog box, in the Project Name text box, type **MyLISPFunctions**.
6. Under Location, click the Ellipsis […] button.
7. In the Project Location dialog box, browse to the location of the dataset for this session. Double click the Dataset folder and click Select Folder.

   The location might look something like:

   C:\Datasets\SD500025

8. Click the Framework drop-down list and choose .NET Framework 4.8.

   If you don't have 4.8 installed, you can choose .NET Framework 4.7 as well.



Configure your new project

AutoCAD 2022 VB plug-in

Project name: MyLISPFunctions

Location: C:\Datasets\SD500025\

Solution name: MyLISPFunctions

☐ Place solution and project in the same directory

Framework: .NET Framework 4.8

9. Click Create.
10. In the AutoCAD .NET Wizard Configurator dialog box, click the first Ellipsis [...] button and browse the location of the *Inc* folder in the ObjectARX SDK.

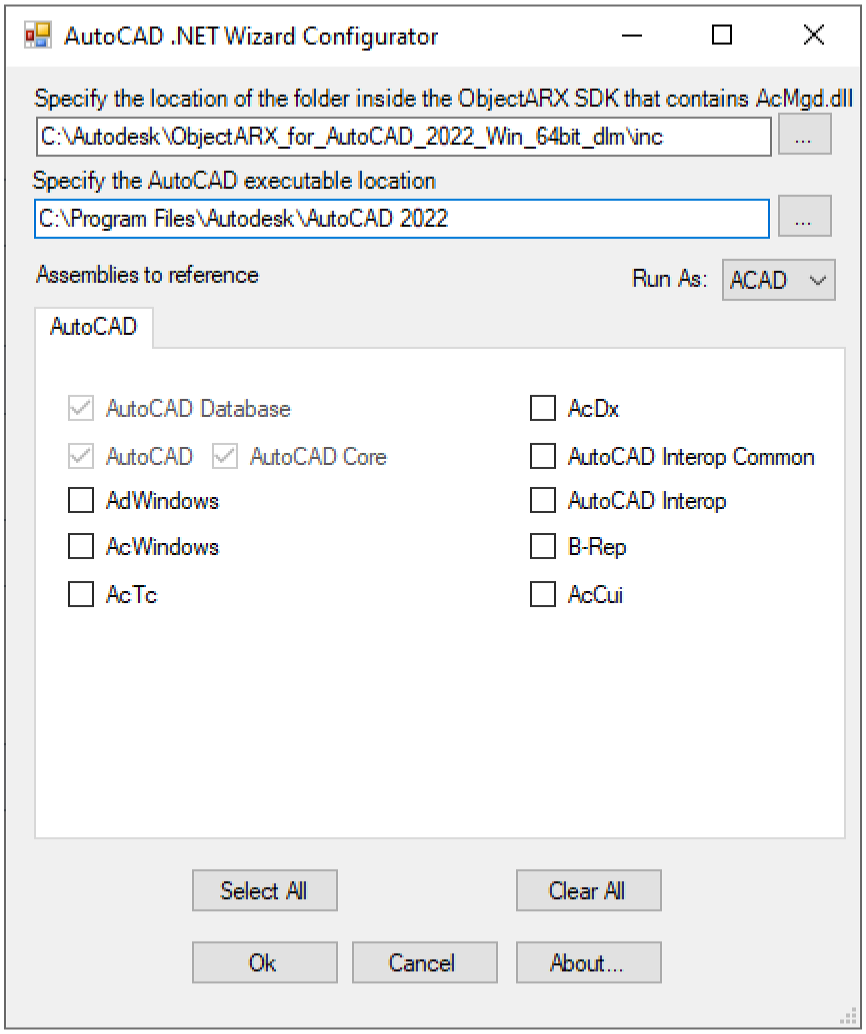
    By default, the folder is at:

    C:\Autodesk\ObjectARX_for_AutoCAD_2022_Win_64bit_dlm\inc

11. Click the second Ellipsis […] button and browse to the folder of the *acad.exe* file.

    By default, AutoCAD 2022 is installed in the folder:

    C:\Program Files\Autodesk\AutoCAD 2022



AutoCAD .NET Wizard Configurator

Specify the location of the folder inside the ObjectARX SDK that contains AcMgd.dll

C:\Autodesk\ObjectARX_for_AutoCAD_2022_Win_64bit_dlm\inc

Specify the AutoCAD executable location

C:\Program Files\Autodesk\AutoCAD 2022

Assemblies to reference

Run As: ACAD

AutoCAD

- [✓] AutoCAD Database
- [✓] AutoCAD
- [✓] AutoCAD Core
- [ ] AdWindows
- [ ] AcWindows
- [ ] AcTc
- [ ] AcDx
- [ ] AutoCAD Interop Common
- [ ] AutoCAD Interop
- [ ] B-Rep
- [ ] AcCui

12. Optionally, check AutoCAD Interop Common and AutoCAD Interop if you plan on using the AutoCAD ActiveX/COM API with your .NET application.

    Note: You can't use or access the VLA-OBJECT data type with your custom AutoLISP functions.

13. Click OK.
14. Click File menu > Save All.

Using the AutoCAD .NET Wizard simplifies the learning curve of setting up a new VB .NET project, but it also hides a few things that you should know as well when you are first getting started. Appendix 2: Create a New VB .NET Project without the .NET Wizard later in this handout explains how to create a VB.NET project from scratch without using the AutoCAD .NET Wizard.

5 Create a Command

Commands are the most common way of exposing functionality defined in a VB.NET project to a user and for use in an AutoLISP program. To define a command, you prefix a `Public` method with the `CommandMethod` attribute. This lets AutoCAD know that the following method should be defined in AutoCAD as a command when it is loaded.

Any `Public` method in a VB.NET project can be used to define a command, but it should not be defined to accept any parameters. Parameters are used to pass values into a method, much like the variables defined before the backslash of a `defun` expression in AutoLISP. If your command requires input, you must prompt the user for the appropriate values at the Command prompt or use a dialog box.

Add the Code to Create a Command

These steps explain how to define a command with the global name of HELLOAU and a local name of BONJOURAU. The command prints the text “Welcome to AU 2021!” to the Command window.

1. In Visual Studio, in the Solution Explorer, double-click *myCommands.vb*.

   The file will open in an editor window, and if you scroll through the file you will notice there are several boiler plate templates for commands and functions. This was added as part of the AutoCAD 2022 VB plug-in template.

   If you don’t have *myCommands.vb* in your project, see Step 17under the Appendix 2: Create a new VB .NET Project without the .NET Wizard section of this handout.

2. Scroll to the bottom of the file and click in the line above `End Class` and then press Enter.

3. On the new blank line, enter the following code:

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

   MyGroup – Group name used to organize related commands. Defined groups can be seen with the Commands option of the ARX command in AutoCAD.

   HelloAU – Global command name.

   BonjourAU – Local/regional command name.

   CommandFlags.Modal – Command flags used to control the behavior of the command.

4. Click the File menu > Save All to save the changes made to the project.

6 Build and Load a .NET Assembly

After you have defined the methods that represent the new commands or functions to be made available in AutoCAD, you must build (often referred to as compile) your project into a .NET assembly – or DLL. Once a DLL is been built, you use the NETLOAD command to load it into AutoCAD.

Note: DLLs loaded into AutoCAD cannot be unloaded from the current session. However, they are also not persistent between sessions unless they are loaded through the Windows Registry.

For information on loading a .NET DLL using the Windows Registry, see the Develop Applications with VB.NET and C# > Distribute Your Application topic (*https://help.autodesk.com/view/OARX/2022/ENU/?guid=GUID-70D60274-57E0-4B22-8D0C-3C7F212A7CAF*) in the AutoCAD .NET Developer's Guide.

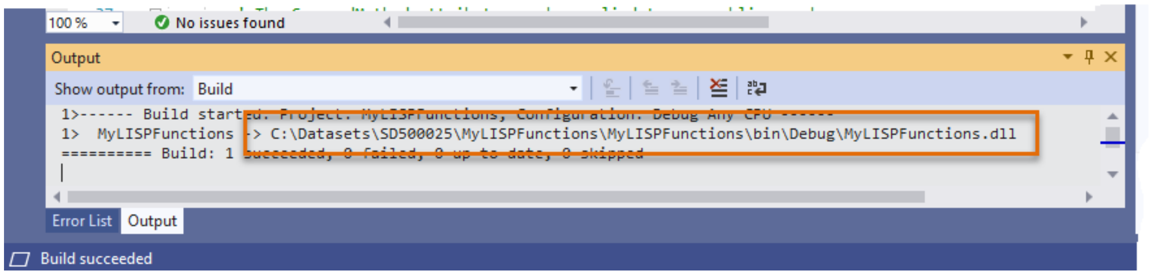
Build and Load a .NET Assembly

You build a DLL from your project using the Build menu within Microsoft Visual Studio. The following steps explain how to build and load your DLL into AutoCAD, and then start the HelloAU command.

Note: For information on how to debug and build your project for release, see *Appendix 1: Debugging and Building a VB .NET Project for Release*.

1. With your project open in Visual Studio, click Build menu > Build MyLISPFunctions.

   The build status and location of your DLL are displayed in the Output window.

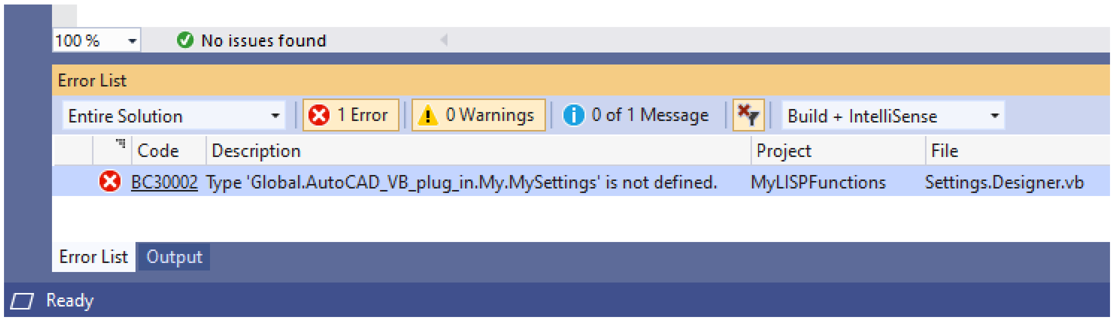


```
1>------ Build started: Project: MyLISPFunctions, Configuration: Debug Any CPU ------
1>  MyLISPFunctions -> C:\Datasets\SD500025\MyLISPFunctions\MyLISPFunctions\bin\Debug\MyLISPFunctions.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
```

   If the Output window is not displayed, click View menu > Output.

2. If any errors are displayed in the Output window, make the required changes to the code and try to build the project again.

   For example, if you see the following in the Error List, double-click the entry to open the file in the editor window.



	Code	Description	Project	File
✖	BC30002	Type 'Global.AutoCAD_VB_plug_in.My.MySettings' is not defined.	MyLISPFunctions	Settings.Designer.vb

In the editor window for this particular error, replace the two instances of:

```
Global.AutoCAD_VB_plug_in.My.MySettings
```

With the following:

```
Global.MyLISPFunctions.My.MySettings
```

Note: This might happen if the AutoCAD .NET Wizard doesn't properly update the files in the template while creating the project.

3. If you had to fix any errors in Step 2, rebuild the project again following Step 1.
4. Launch AutoCAD.
5. In AutoCAD, at the Command prompt, enter **netload**.
6. In the Choose .NET Assembly dialog box, browse to the location of the DLL displayed in the Output window and select it. Click Open.

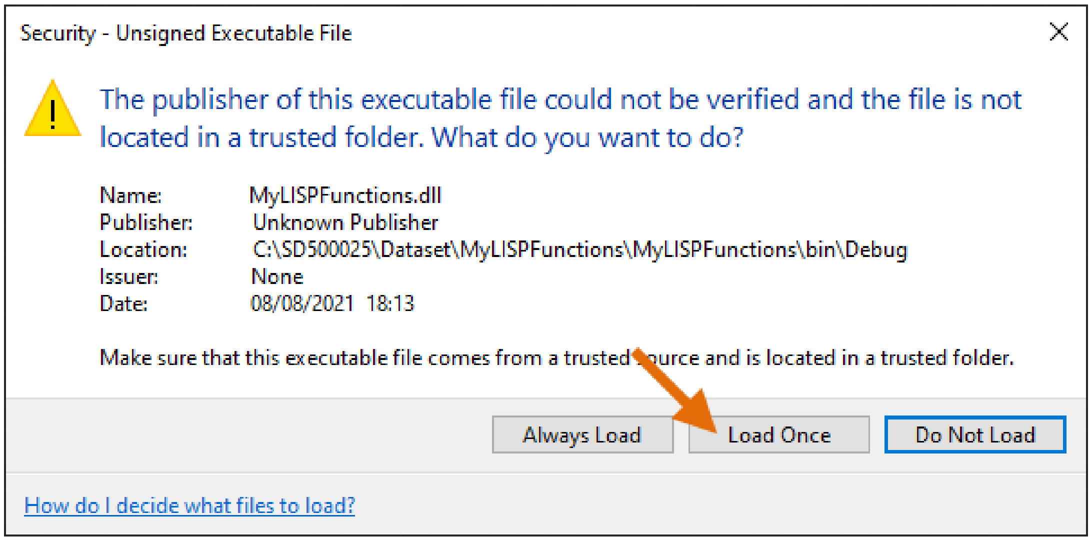
   By default, your DLL should be built to:

   C:\ Dataset\SD500025\MyLISPFunctions\MyLISPFunctions\bin\Debug\

   Tip: You can also use the Command function with the NETLOAD command if you want to load the DLL using AutoLISP or a macro. There are no AutoLISP functions specific for loading a .NET DLL like there is for AutoLISP (`LOAD`) or ObjectARX (`ARXLOAD`) files. An example of using NETLOAD at the command line is:

   ```
   (command "netload" "C:/MyUtilities/MyLISPFunctions.dll")
   ```

7. If the Security – Unsigned Executable File dialog box is displayed, click Load Once.

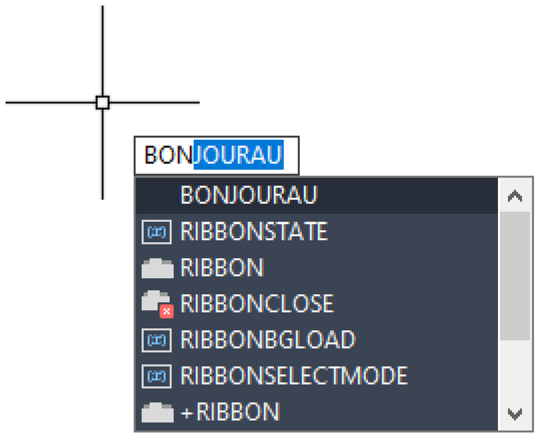

Security - Unsigned Executable File

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?

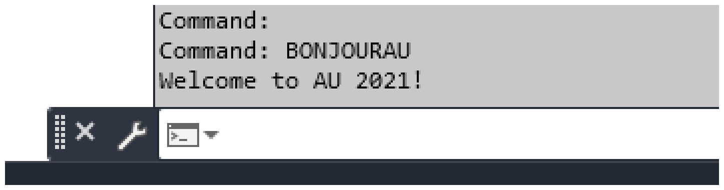
Name: MyLISPFunctions.dll
Publisher: Unknown Publisher
Location: C:\SD500025\Dataset\MyLISPFunctions\MyLISPFunctions\bin\Debug
Issuer: None
Date: 08/08/2021 18:13

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

8. At the Command prompt, enter **bonjourau**.



The message “Welcome to AU 2021!” is displayed in the Command window.



```
Command:
Command: BONJOURAU
Welcome to AU 2021!
```

9. At the Command prompt, enter **helloau**.

   You may notice the command didn’t appear in the AutoComplete list but the message was output to the Command window as expected. It doesn’t appear in the AutoComplete list because it is our global command name.

10. At the Command prompt, enter _**helloau**.

    This time, you should have noticed that it appeared in the AutoComplete list

11. At the Command prompt, enter **(command "bonjourau")**.

    The command runs as expect with the `COMMAND` function, just like a standard AutoCAD command would such as LINE or CIRCLE.

7 Define an AutoLISP Function

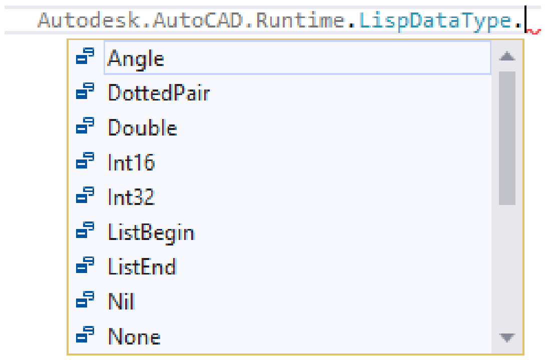
Now that the basics are out of the way, it's time to get down to the fun stuff and that is defining a custom AutoLISP function. Defining an AutoLISP function is similar to creating a custom command, but instead of the `CommandMethod` attribute you use the `LispFunction` attribute. The method you create in .NET for your custom AutoLISP function must be declared as `Public` and accept a single parameter of the `ResultBuffer` type only. `ResultBuffer` is a data type that is specific to AutoCAD and is similar to a list in AutoLISP.

```
' This method can have any name
<LispFunction("MyLispFunction", "MyLispFunctionLocal")> _
Public Function MyLispFunction(ByVal args As ResultBuffer)
    ' Put your command code here

    ' Return a value to the AutoCAD Lisp Interpreter
  Return 1
End Function
```

While it might look like the custom AutoLISP function only allows for a single parameter, that isn't true as the `ResultBuffer` represents an array of values; so it can be empty or have one or more values. For example, the following usages of the previous function are all valid:

```
(MyLispFunction)
(MyLispFunction "One")
(MyLispFunction "One" "Two")
(MyLispFunction '("One" "Two"))
```

The values of a `ResultBuffer` correspond to one of the data types or data structures that AutoLISP supports. These data types and structures are listed in the `Autodesk.AutoCAD.Runtime.LispDataType` enumerator.


```
Autodesk.AutoCAD.Runtime.LispDataType.
```

Values of a `ResultBuffer` are defined as the `TypedValue` data type. When you create a new or receive a `TypedValue`, it contains two properties. The first property is known as the *Type Code* which is commonly one of the constants in the `LispDataType` enumerator.

The second property is the actual value you assigned to the `TypedValue`. When you define a custom AutoLISP function, it should always return a value of the `TypedValue` or `ResultBuffer` type. Use a `TypedValue` when you want to return a single value like a string, integer, or `nil` or a `ResultBuffer` when you want to return a list of values.

For example, to return the text "String" from an AutoLISP function you might use:

```
Return New TypedValue(LispDataType.Text, "String")
```

To return a list containing an integer value of 0 and the string “INSERT” you might use:

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

To return a list containing a dotted pair of (0 . "INSERT") you might use:

```
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
```

To return a list containing two nested lists of ((0 . "INSERT") (0 5 0)) you might use:

```
Dim rbRt As New ResultBuffer
rbRt.Add(New TypedValue(LispDataType.ListBegin))
rbRt.Add(New TypedValue(LispDataType.ListBegin))
rbRt.Add(New TypedValue(LispDataType.Int16, 0))
rbRt.Add(New TypedValue(LispDataType.DottedPair))
rbRt.Add(New TypedValue(LispDataType.Text, "INSERT"))
rbRt.Add(New TypedValue(LispDataType.ListEnd))
rbRt.Add(New TypedValue(LispDataType.Point3d, New Point3d(0, 5, 0)))
rbRt.Add(New TypedValue(LispDataType.ListEnd))
Return rbRt
```

Add the Code to Define an AutoLISP Function

These steps explain how to define a custom AutoLISP function named `printargs`. The function will accept no arguments, or one or more arguments. When a value is provided in an argument, its type code and value are printed in the Command window.

1. In Visual Studio, in the Solution Explorer, double-click *myCommands.vb*.
2. Scroll to the bottom of the file and click in front of the `End Class` line and press Enter twice.
3. On the new blank line just above `End Class`, enter the following code:

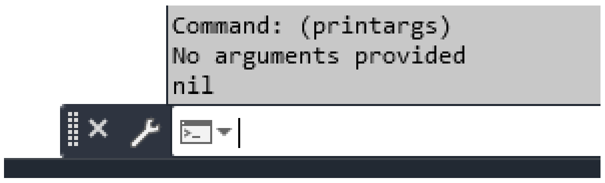
```
' Defines an AutoLISP function that prints the
' type code and value for each argument
' Usage: (printargs '((0 . "INSERT") (0 5 0)))
<LispFunction("PrintArgs")>
Public Function PrintArgs(ByVal rb As ResultBuffer)
    ' Gets the AutoCAD editor
    Dim docEditor As Editor =
        Application.DocumentManager.MdiActiveDocument.Editor
```

```
     ' Checks to see if any values were passed into the function
     If Not rb = Nothing Then
         ' Step through each value passed
         For Each val As TypedValue In rb
             ' Output the type and value
             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
     Else
         docEditor.WriteMessage(vbLf & "No arguments provided" & vbLf)
     End If

     ' Return nil
     Return New TypedValue(LispDataType.Nil)
 End Function
```

4. Click File > Save All to save the changes made to the project.
5. Build and load the DLL into AutoCAD.

   Note: If the DLL is loaded into AutoCAD already, you will need to close and restart AutoCAD so you can build the DLL.
6. When the DLL has been loaded into AutoCAD, at the Command prompt, enter **(printargs)**.

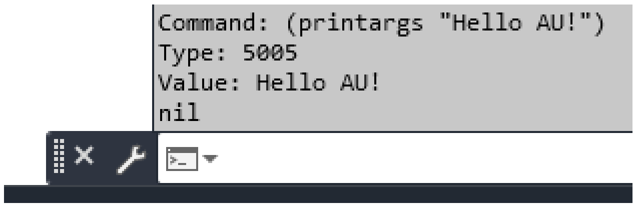
   The text message "No arguments provided" appears in the Command window.



```
Command: (printargs)
No arguments provided
nil
```

7. At the Command prompt, enter **(printargs "Hello AU!")**.

   The argument is identified as LispDataType.Text (5005) with a value of "Hello AU!".



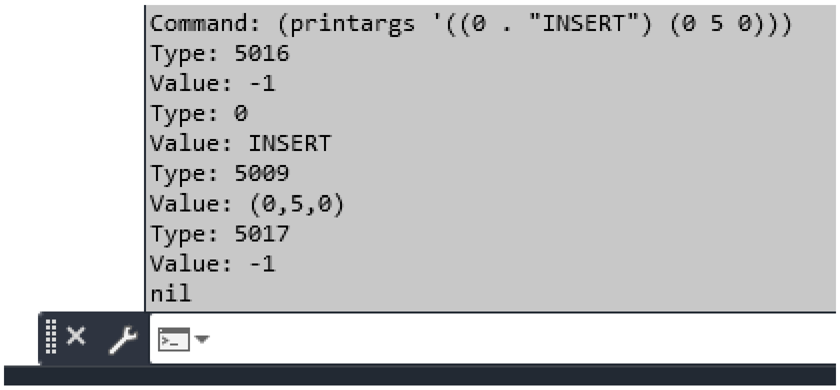
```
Command: (printargs "Hello AU!")
Type: 5005
Value: Hello AU!
nil
```

8. At the Command prompt, enter **(printargs '((0 . "INSERT") (0 5 0)))**.

The arguments are identified as the following:

- LispDataType.ListBegin (5016) with a value of -1
- 0 with a value of "INSERT"

  Note: When a dotted pair is provided, the first value in the pair is its type code.
- LispDataType.Point3d (5009) with a value of (0,5,0)
- LispDataType.ListEnd (5017) with a value of -1



```
Command: (printargs '((0 . "INSERT") (0 5 0)))
Type: 5016
Value: -1
Type: 0
Value: INSERT
Type: 5009
Value: (0,5,0)
Type: 5017
Value: -1
nil
```

Note: The `printargs` function always returns `nil`, which is like many other AutoLISP functions. The returned `nil` can be captured with `(princ)` so it doesn't output to the Command window.

Working with Dotted Pair

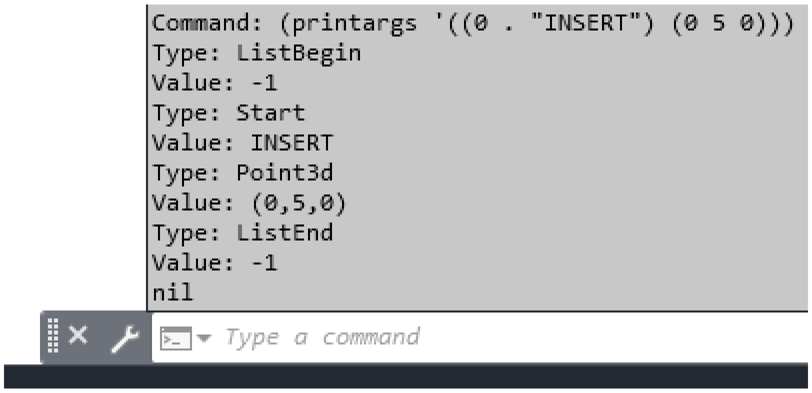
Dotted pairs are commonly used to represent DXF data of an object, but that isn't always the case. When defining a dotted pair, don't use a type code greater than 5000. Type code values greater than 5000 are reserved for AutoLISP data types. Type codes typically used to represent DXF data are represented by the `DxfCode` enumerator constants.

```
If val.TypeCode >= 5000 Then
    Dim lspDataType As LispDataType = val.TypeCode
    docEditor.WriteMessage(vbLf & "Type: " &
                           lspDataType.ToString())
Else
    Dim dxfCodeType As DxfCode = val.TypeCode
    docEditor.WriteMessage(vbLf & "Type: " &
                           dxfCodeType.ToString())
End If
```

The above code is a modification of the following line in the printarg function.

```
docEditor.WriteMessage(vbLf & "Type: " &
                       val.TypeCode.ToString())
```

Instead of outputting the integer value of the type code assigned to the `TypedValue`, it uses the actual enumerators to get the name of the constant value.



```
Command: (printargs '((0 . "INSERT") (0 5 0)))
Type: ListBegin
Value: -1
Type: Start
Value: INSERT
Type: Point3d
Value: (0,5,0)
Type: ListEnd
Value: -1
nil
```

Other Examples of AutoLISP Functions

The following code example is of a custom AutoLISP function that expects two arguments which are of the string data types. The function validates it was passed 2 and only arguments and then assigns each argument to a string variable if possible before displaying the values to the Command window. The function returns a value of `LispDataType.Nil`.

```
' Defines an AutoLISP function named Employee
' Usage: (employee "Lee Ambrosius" "12345")
<LispFunction("Employee")>
Public Function Employee(ByVal rb As ResultBuffer)
    ' Gets the AutoCAD editor
    Dim docEditor As Editor =
        Application.DocumentManager.MdiActiveDocument.Editor

    ' Checks to see if any values were passed into the AutoLISP function
    If Not rb = Nothing Then
        If rb.AsArray().Length <= 2 Then
            Dim name As String = "", id As String = ""
            Dim count As Integer = 0

            ' Step through each value passed
            For Each val As TypedValue In rb
                ' Check to see if the value passed was a string
                If (val.TypeCode = LispDataType.Text) Then
                    ' If a string, assign it to the correct variable
                    Select Case count
                        Case 0
                            name = val.Value.ToString()
                        Case 1
                            id = val.Value.ToString()
                    End Select
                Else
                    ' Display error message if an argument wasn't string
                    docEditor.WriteMessage(vbLf &
                                   "; error: argument not of STR type" &
                                   vbLf)
                    ' Return nil
                    Return New TypedValue(LispDataType.Nil)
                End If

                ' Increments the counter by 1
                count = count + 1
            Next

            ' Output the two string values
            docEditor.WriteMessage(vbLf & "Name: " & name &
                                   vbLf & "ID: " & id & vbLf)
        Else
            ' Display error message when more than 2 values are passed in
            docEditor.WriteMessage(vbLf &
                                   "; error: too many arguments" & vbLf)
        End If
    Else
        ' Display error message when no values are passed in
        docEditor.WriteMessage(vbLf &
                               "; error: too few arguments" & vbLf)
    End If

    ' Return nil
    Return New TypedValue(LispDataType.Nil)
End Function
```

When the above function is loaded, try the following AutoLISP statements to see how errors are handled:

```
(employee "Lee Ambrosius" "12345")
(employee "Lee Ambrosius" 12345)
(employee)
(employee "Lee" "Ambrosius" "12345")
```

The following example demonstrates how you can return a string from an AutoLISP function.

```
' Defines an AutoLISP function that returns a string
' Usage: (stringreturn)
<LispFunction("StringReturn")> _
Public Function StringReturn(ByVal rb As ResultBuffer)
    ' Return a string value from the function
    Return New TypedValue(LispDataType.Text, "My Value")
End Function
```

Note: The above examples, the completed project in the dataset files for this session also contains a function named FuncReturn. The function demonstrates how to return many of the common data types to AutoLISP from VB.NET.

Work with AutoLISP User-Defined Variables

Along with working with values passed into an AutoLISP function, you can get and set the values of an AutoLISP user-defined variable. Accessing AutoLISP user-defined variables allows you to use global variables across AutoLISP programs and custom functions defined with VB .NET.

The way you work with AutoLISP user-defined variables is through the use of the following methods:

> `GetLispSymbol` – Gets the value assigned to an AutoLISP user-defined variable. The method accepts the name of the variable for which to get its value and returns an `Object` that contains the value of the variable. The `GetType` method of the `Object` can be used to determine the type of data that the `Object` represents.
>
> `SetLispSymbol` – Assigns a value to an AutoLISP user-defined variable. This method takes the name of the variable for which you want to assign a value and an `Object` or `TypedValue` type that contains the value to be assigned to the variable.

Note: If you are using AutoCAD 2012 or earlier, to get and set the value of AutoLISP user-defined variable, you will have to import the `acedGetSym` and `acedPutSym` methods from the *acad.exe* or *accore.dll* file. See the Using the `acedGetSym` and `acedPutSym` Methods section later in this handout and the completed project available as part of this session's materials for examples.

Get and Set User-Defined AutoLISP Variables

These steps explain how to define two custom AutoLISP functions named `fetchfoo` and `setfoo`. The functions don't require any arguments. `setfoo` assigns a string to the AutoLISP user-defined variable *foo*, while `fetchfoo` gets the data type of the value assigned to the AutoLISP user-defined variable *foo*. Both of the functions return the value assigned to a `TypedValue` type.

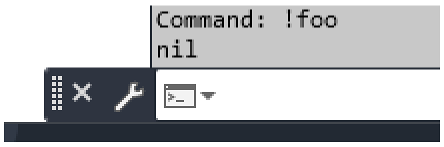
1. In Visual Studio, in the Solution Explorer, double-click *myCommands.vb*.
2. Scroll to the bottom of the file and click in front of the `End Class` line and press Enter twice.
3. On the new blank line just above `End Class`, enter the following code:

```
' Defines or updates the FOO variable
' Usage: (setfoo)
<LispFunction("SetFoo")>
Public Function SetFoo(ByVal rb As ResultBuffer)
    ' Gets the AutoCAD editor and current document
    Dim docEditor As Editor =
        Application.DocumentManager.MdiActiveDocument.Editor
    Dim doc As Document =
        Application.DocumentManager.MdiActiveDocument

    Dim typeValue As New TypedValue(LispDataType.Nil)
    If rb = Nothing Then
        typeValue = New TypedValue(LispDataType.Text, "Default Value")
        docEditor.WriteMessage(vbLf + "String assigned to FOO" + vbLf)
    End If

    ' Assign the value to FOO
    doc.SetLispSymbol("foo", typeValue)

    Return typeValue
End Function

' Returns the data type assigned to the FOO variable
' Usage: (fetchfoo)
<LispFunction("FetchFoo")>
Public Function FetchFoo(ByVal rb As ResultBuffer)
    ' Gets the current document
    Dim doc As Document =
        Application.DocumentManager.MdiActiveDocument

    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

    Return typeValue
End Function
```

4. Click File > Save All to save the changes made to the project.
5. Build and load the DLL into AutoCAD.

   Note: If the DLL is loaded into AutoCAD already, you will need to close and restart AutoCAD so you can build the DLL.
6. When the DLL has been loaded into AutoCAD, at the Command prompt, enter **!foo**.

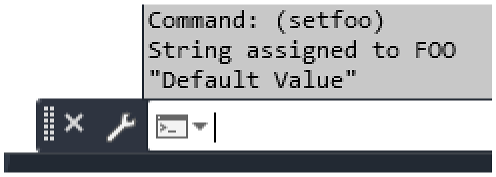
   `nil` should be returned and appear in the Command window since the variable hasn't been defined or initialzed yet.



```
Command: !foo
nil
```

7. At the Command prompt, enter **(setfoo)**.

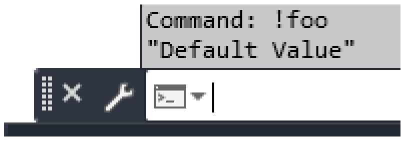
   The message "String assigned to FOO" appears in the Command window along with the text string of "Default Value" which is the value returned by the function and assigned to the FOO variable.



```
Command: (setfoo)
String assigned to FOO
"Default Value"
```

8. At the Command prompt, enter **!foo**.

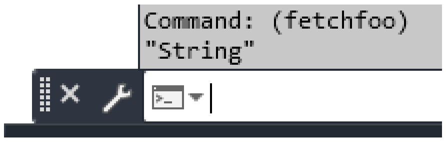
   `"Default Value"` is returned and appears in the Command window.



```
Command: !foo
"Default Value"
```

9. At the Command prompt, enter **(fetchfoo)**.

   `"String"` is returned and appears in the Command window because the `foo` variable is assigned a value of the data type String.



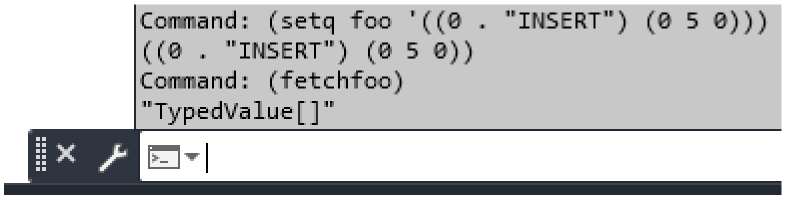
```
Command: (fetchfoo)
"String"
```

10. At the Command prompt, enter **(setq foo '((0 . "INSERT") (0 5 0)))**.

    A list of a dotted pair and a 3D point is assigned to the `foo` variable.

11. At the Command prompt, enter **(fetchfoo)**.

    `"TypedValue[]"` is returned and appears in the Command window because the `foo` variable is assigned an array of `TypedValue` types.



```
Command: (setq foo '((0 . "INSERT") (0 5 0)))
((0 . "INSERT") (0 5 0))
Command: (fetchfoo)
"TypedValue[]"
```

8 Request User Input

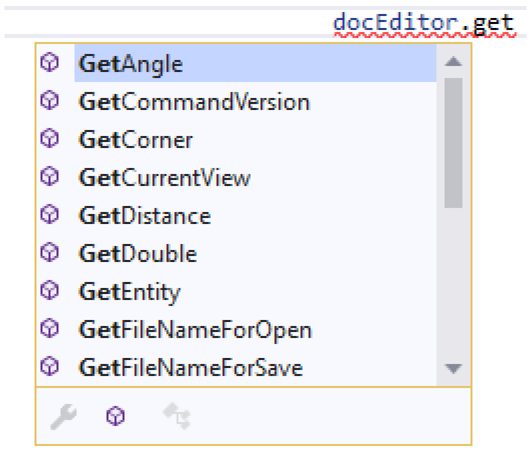
Requesting user input at the Command prompt is an important part of most custom applications in AutoCAD. The AutoCAD Managed .NET API gives you access to many of the same input functions found in AutoLISP, except you have much greater control over their behavior. With the user input functions in Managed .NET, you can determine if the user finished entering input, used a keyword, or cancelled the input method.

There are three parts to the input methods in VB .NET; setup, request, and test. First you setup the prompt and keywords, and any of the available input options you want to use. Then you request input from the user before finally testing the status of the input provided.

Input methods are part of the `Editor` class in the AutoCAD Managed .NET API. You can reference the current editor by using the following code snippet in your VB .NET application.

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

After you get the current editor, you can use the `Editor` object to determine which input methods are available. Hint, much like AutoLISP, they start with 'Get'.

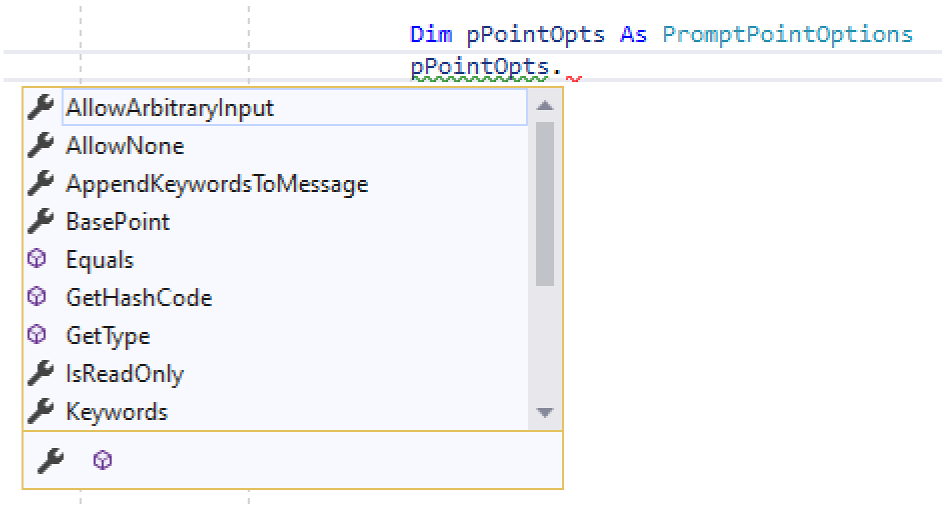


```
docEditor.get
```

GetAngle
GetCommandVersion
GetCorner
GetCurrentView
GetDistance
GetDouble
GetEntity
GetFileNameForOpen
GetFileNameForSave

After you identify the user input method you want to use, you can determine everything you need to setup the method for use. All of the `Get*` input methods require the use of a *PromptOptions* class. *PromptOptions* is not an actual class found in the AutoCAD Managed .NET API, but is a way to reference a grouping of different classes that are related to prompts and options for the `Get*` input methods.

For example, you use the `GetPoint` method to prompt a user for a point. The `GetPoint` method requires you to pass it a value of the `PromptPointOptions` type. When you assign a variable as the `PromptPointOptions` type, you can specify the prompt string and any keywords that you might want to use. You can also change these and other behavior properties after the variable is defined. In AutoLISP, you would use the `Initget` function to do this as well as specify a base point in the case of the `GetPoint` function itself.

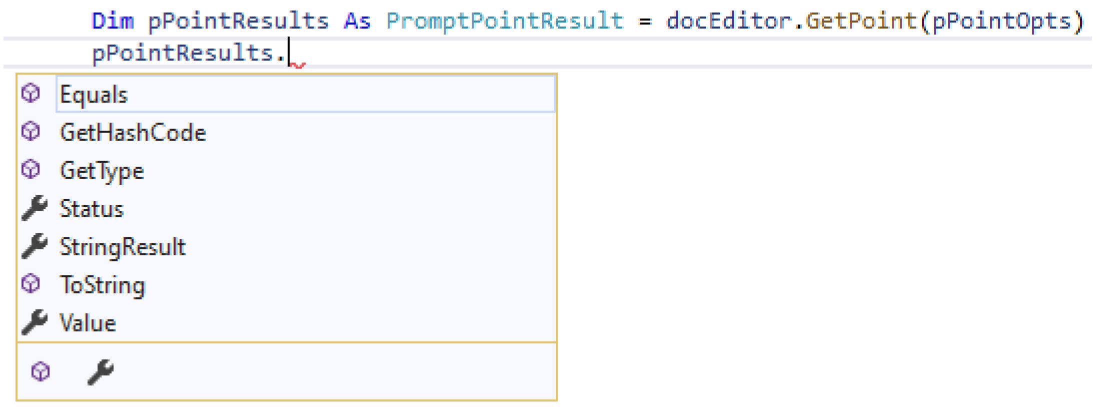


```
Dim pPointOpts As PromptPointOptions
pPointOpts.
```

After you define the appropriate properties for the *PromptOptions* type, you then use the object with the correct input method.

```
Dim pPointOpts As New PromptPointOptions(vbLf + _
                                        "Specify insertion point: ")

docEditor.GetPoint(pPointOpts)
```

While the above code snippet is valid, it is not helpful as the input that the user provides is not stored for you to use. As previously mentioned, each input method requires you to specify a variable defined as the appropriate *PromptOptions* type to control the behavior of the input method. There is also a specific set of classes that are available for capturing the input specified by the user. These are referred to as the *PromptResult* classes.



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

The *PromptResult* classes allow you to check the status of a `Get*` input method as well as to get the value the user specified. Commonly, you are going to check the `Status` property against the following constants:

- `PromptStatus.OK` - User provided valid input
- `PromptStatus.Cancel` - User pressed Esc and cancelled the current input
- `PromptStatus.Keyword` - User entered one of the available keywords

Each *PromptResult* class allows you to get the value entered or when there is a keyword, the full text string of the keyword value.

Add a Basic Request for User Input to an AutoLISP Function

These steps explain how to define a custom AutoLISP function named `pointorkword`. The function prompts the user for a point using the `GetPoint` method. It also demonstrates how you can get the point specified, the keyword entered, or check whether the user pressed Esc. The function returns a value assigned to a `TypedValue` type.

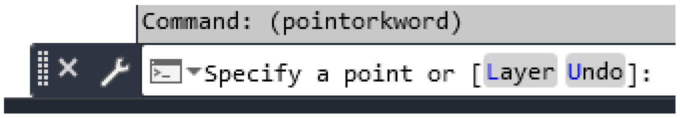
1. In Visual Studio, double-click *myCommands.vb* from the Solution Explorer.
2. Scroll to the bottom of the file and click in front of the `End Class` line and press Enter twice.
3. On the new blank line just above `End Class`, enter the following code:

```
' Defines an AutoLISP function that prompts for a point or keyword
' Usage: (pointorkword)
<LispFunction("PointOrKWord")>
Public Function PointOrKWord(ByVal rb As ResultBuffer)
    Dim docEditor As Editor =
        Application.DocumentManager.MdiActiveDocument.Editor

    Dim typeValue As New TypedValue(LispDataType.Nil)

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

    Dim pPointResult As PromptPointResult =
        docEditor.GetPoint(pPointOpts)

    ' User specified a point
    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

    Return typeValue
End Function
```

4. Click File > Save All to save the changes made to the project.

5. Build and load the DLL into AutoCAD.
6. When the DLL has been loaded into AutoCAD, at the Command prompt, enter **(setq rtVal (pointorkword))**.

   The prompt message `Specify a point or [Layer Undo]:` is displayed in the Command.

   ```
   Command: (pointorkword)
   Specify a point or [Layer Undo]:
   ```

7. At the Command prompt, pick a point in the drawing.

   The point specified in the drawing is assigned to the *rtVal* variable.
8. At the Command prompt, enter **(setq rtVal (pointorkword))** and then enter **L**.

   The Layer keyword is assigned to the *rtVal* variable as a string.
9. At the Command prompt, enter the following AutoLISP statements:

   ```
   (defun c:TestInput ()
       (setq pt1 (getpoint "Specify a point with AutoLISP: "))
       (setq pt2 (pointorkword))
       (alert "AutoLISP continued.")
   )
   ```

10. With the TestInput command defined using AutoLISP, at the Command prompt, enter **testinput**.
11. At the `Specify a point with AutoLISP` prompt, press ESC.

    You should notice the TestInput command was cancelled.
12. At the Command prompt, enter **testinput** again.
13. At the `Specify a point with AutoLISP` prompt, specify a point in the drawing this time.
14. At the `Specify a point or [Layer Undo]:` prompt, press ESC.

    You should notice the TestInput command isn't cancelled this time and a message box with the text `Input cancelled` was displayed. This message box is from the `pointorkword` function.
15. Click OK to dismiss the message box with the text `Input cancelled`.
16. Click OK to dismiss the alert message with the text `AutoLISP continued` which is from the AutoLISP defined command TestInput.

    As shown with the user input methods in the AutoCAD .NET API, you have greater control over how your custom programs can continue or end execution.

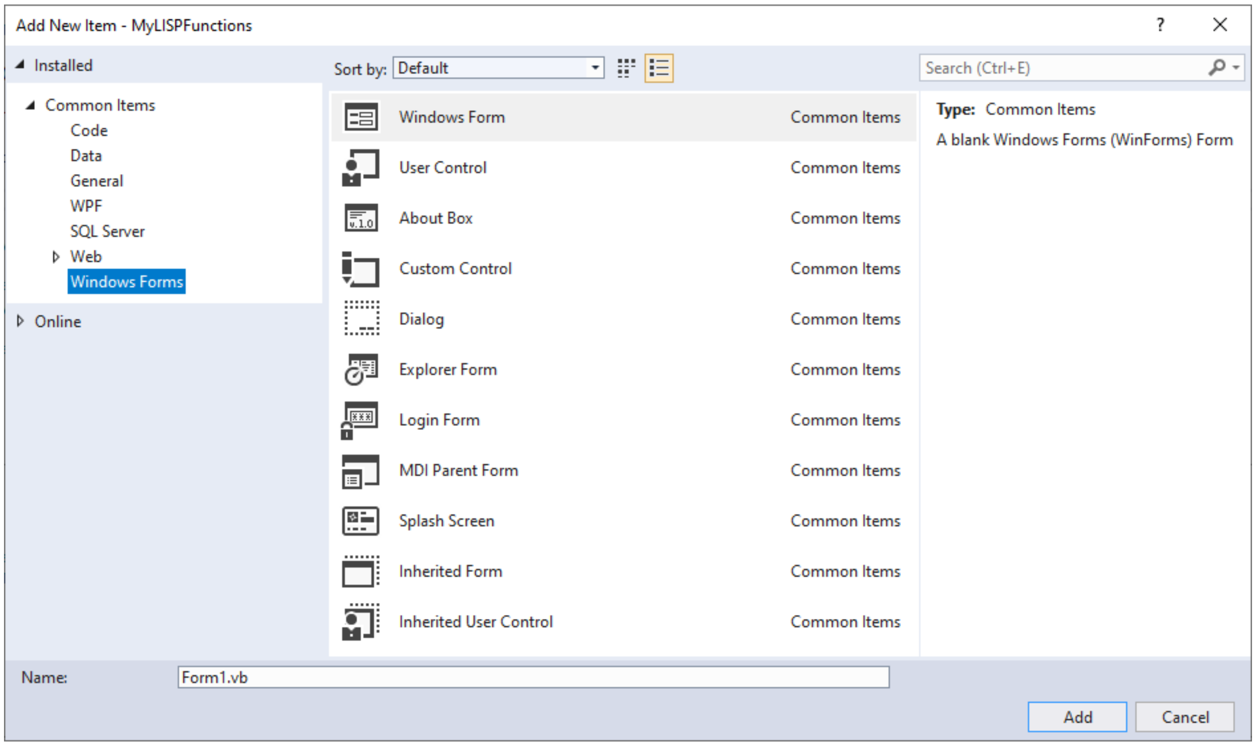
9 Create and Display a Dialog Box

AutoLISP allows you to create and use DCL to create basic dialog boxes, but in many ways it ends up being more of an art than actual science in getting the DCL dialog box to display correctly. VB .NET provides an integrated dialog box editor that can be combined with what you have learned about creating custom AutoLISP functions already up to this point.

You can pass values to and get values from the dialog box just like you have already seen in previous examples of custom functions, but you will want to make sure that you are working with lists in AutoLISP and TypedValue arrays or `ResultBuffers` in VB .NET. If you use lists, you can pass in any number of values you want to use for the dialog box, and also return many values back to your existing AutoLISP applications.

While it takes a little bit of effort to learn how to work with dialog boxes in VB .NET, the effort pays off in the end with a modern dialog box that can take advantage of a wide range of controls that are not available in DCL.

When you create a dialog box in VB .NET, you can start with one of the available templates such as About Box, Splash Screen, or Explorer Form or just a basic dialog box. You access the list of available dialog box related templates by clicking Project menu > Add New Item.


Add New Item - MyLISPFunctions

Installed
- Common Items
  - Code
  - Data
  - General
  - WPF
  - SQL Server
  - Web
  - Windows Forms
- Online

Sort by: Default

Search (Ctrl+E)

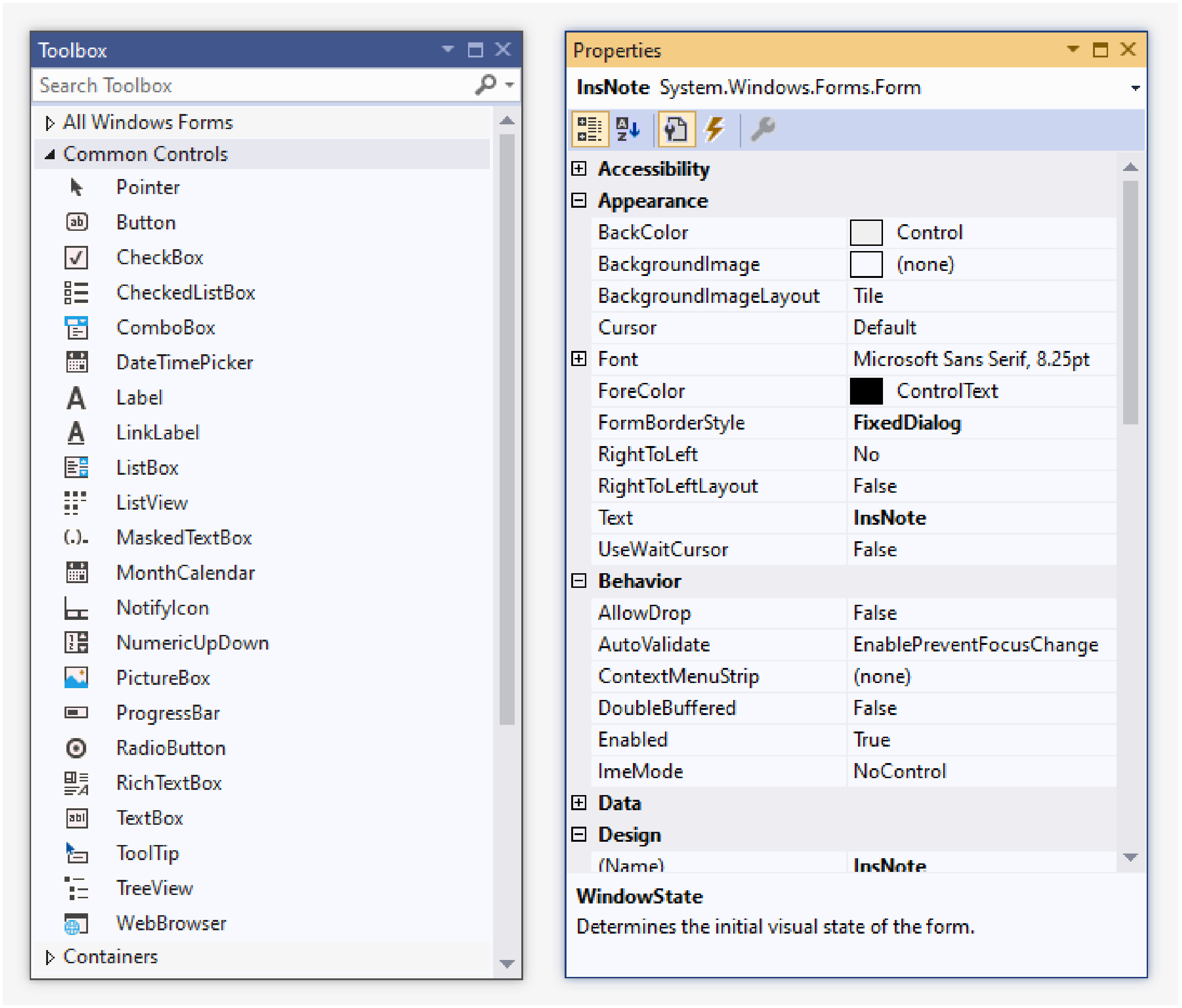
Windows Form	Common Items
User Control	Common Items
About Box	Common Items
Custom Control	Common Items
Dialog	Common Items
Explorer Form	Common Items
Login Form	Common Items
MDI Parent Form	Common Items
Splash Screen	Common Items
Inherited Form	Common Items
Inherited User Control	Common Items

Type: Common Items

A blank Windows Forms (WinForms) Form

Name: Form1.vb

The Dialog template is going to be the one that you will find yourself using commonly. After you add a dialog box to your project, you use the Toolbox to add controls to your dialog box and the Properties window to edit the properties of the dialog box or a selected control. The Properties window functions much like the Properties palette in AutoCAD.



InsNote System.Windows.Forms.Form

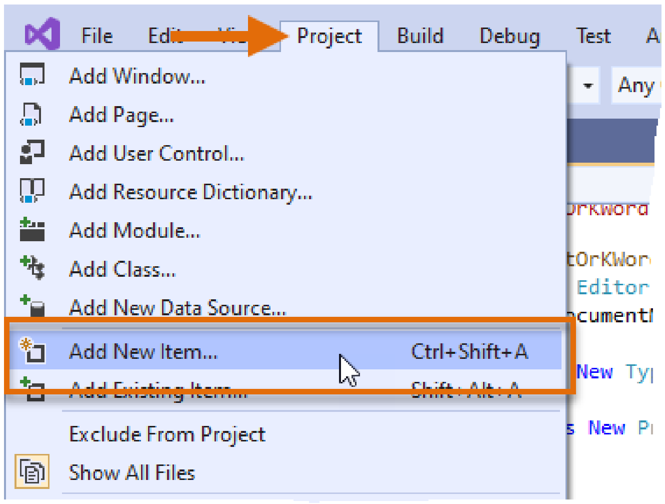
Accessibility	
Appearance	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Cursor	Default
Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
FormBorderStyle	**FixedDialog**
RightToLeft	No
RightToLeftLayout	False
Text	**InsNote**
UseWaitCursor	False
Behavior	
AllowDrop	False
AutoValidate	EnablePreventFocusChange
ContextMenuStrip	(none)
DoubleBuffered	False
Enabled	True
ImeMode	NoControl
Data	
Design	
(Name)	**InsNote**

WindowState
Determines the initial visual state of the form.

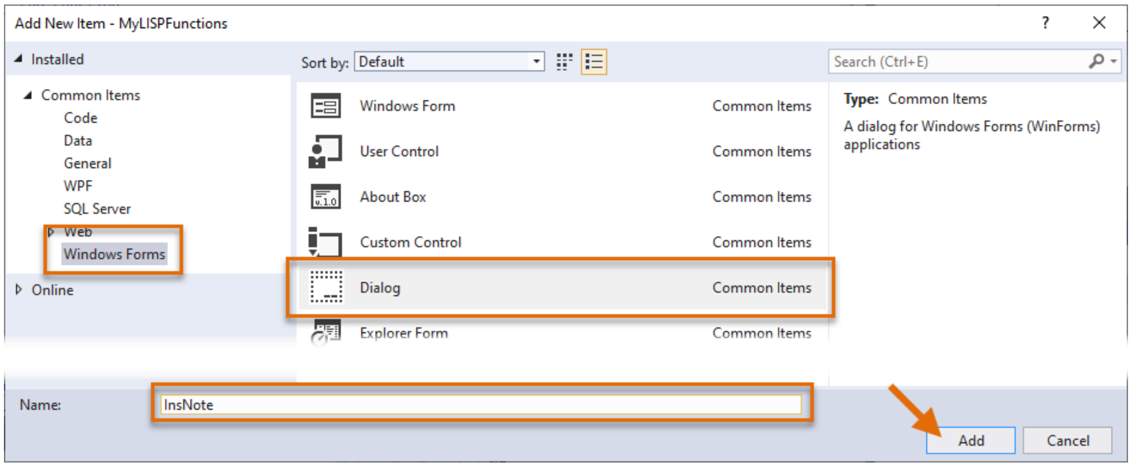
Create a Form in VB.NET

The following steps explain how to create a basic dialog box using the Dialog template. This dialog box will allow the user to choose from several standard notes to insert into the drawing. You will add several labels, a text box, a command button, and a drop-down list. By default, the Dialog template provides you with the standard OK and Cancel buttons.

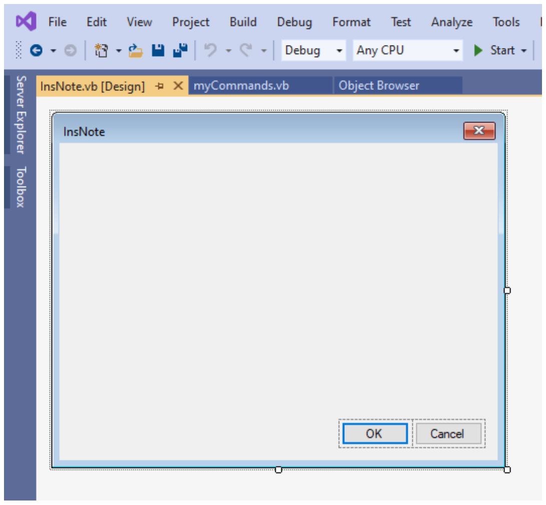
1. In Visual Studio, with the MYLISPFunctions open, click Project > Add New Item (or right-click the MYLISPFunctions node in the Solution Explorer and choose Add > New Item).



2. In the Add New Item dialog box, Installed Components, expand Common Items and choose Windows Forms.
3. In the Items list, select Dialog.

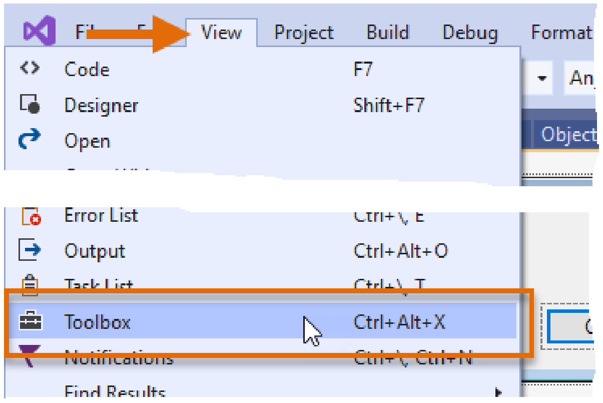

Add New Item - MyLISPFunctions
Installed
Common Items
Code
Data
General
WPF
SQL Server
Web
Windows Forms
Online
Sort by: Default
Search (Ctrl+E)
Windows Form Common Items
User Control Common Items
About Box Common Items
Custom Control Common Items
Dialog Common Items
Explorer Form Common Items
Type: Common Items
A dialog for Windows Forms (WinForms) applications
Name: InsNote
Add
Cancel

4. In the Name text box, select the default text and enter **InsNote**. Click Add.



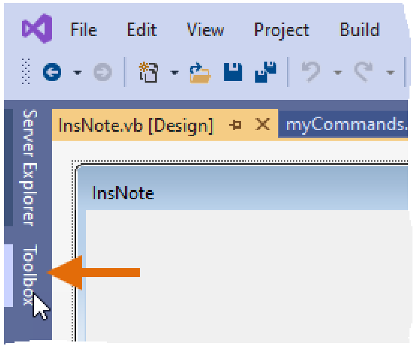
The new dialog box is displayed in the Form editor.

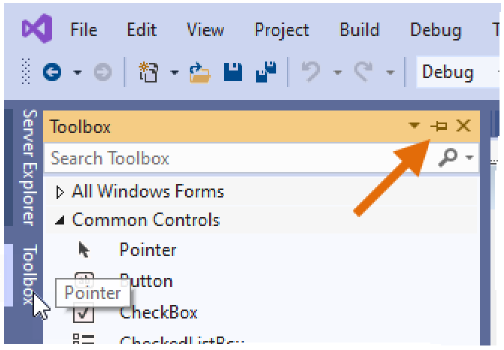
5. Click View menu > Toolbox to display the Toolbox window.



By default, the Toolbox is displayed along the left side of the application window.

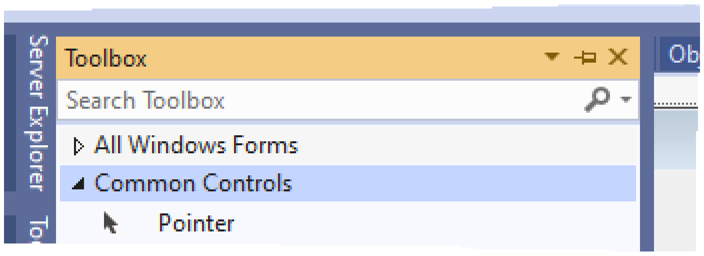
6. Click the Toolbox label to expand it.





If you click the Pushpin that is displayed in the titlebar of the Toolbox when expanded, the Toolbox will remain expanded until you click the Pushpin again.

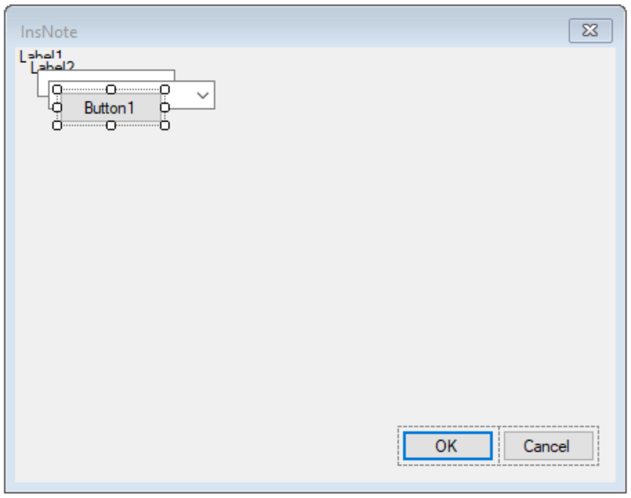
7. In the Toolbox, click the ► next to All Windows Forms if it is currently expanded.
8. Click ► next to Common Controls if it is currently collapsed.



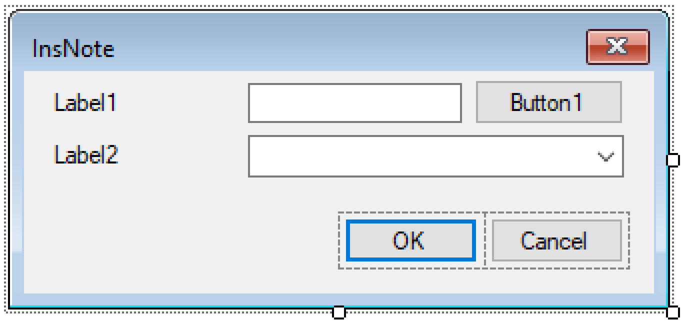
9. Double-click the Label control twice.

   Two new Labels are added to the dialog.
10. Then double-click the TextBox, ComboBox, and Button controls once each.

    Your form should look similar to the following image.



11. Click the Pointer tool, and then arrange and resize the controls and dialog box so it looks similar to the following image.



When the Pointer tool is active, you can manipulate a control in one of the following ways:

- Click a control or the dialog box to display its resizing handles (or grips). Click and drag the handle to resize the control in the direction you drag.
- Click and drag a control to move it on the dialog box.

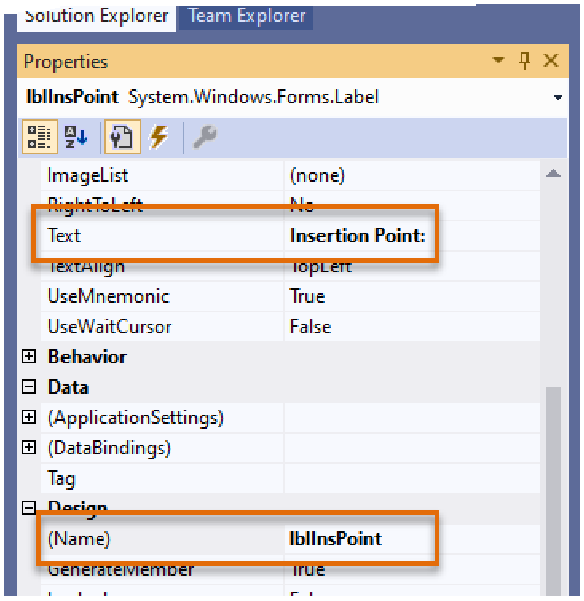
Note: It is best to position the controls and then resize the dialog box, the OK and Cancel buttons will reposition themselves when resizing the dialog box.

12. Select Label1 on your dialog box and use the Properties window to change the control's properties.

Note: If the properties window isn't open, click View > Properties Window.

Change the following properties for the Label1 control:

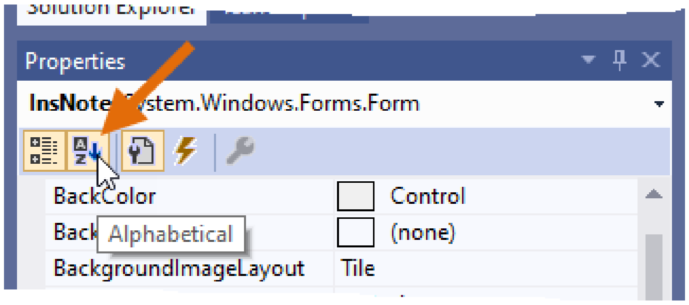
- *Name:* lblInsPoint
- *Text:* Insertion Point:



lblInsPoint System.Windows.Forms.Label

ImageList	(none)
RightToLeft	No
Text	Insertion Point:
TextAlign	TopLeft
UseMnemonic	True
UseWaitCursor	False
Behavior	
Data	
(ApplicationSettings)	
(DataBindings)	
Tag	
Design	
(Name)	lblInsPoint
GenerateMember	True

Tip: If you struggle with finding a property using the Categorized display of the Properties window, click Alphabetical to sort the properties by name.



InsNote System.Windows.Forms.Form

BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile

13. Change the properties for the other controls to the following:

Label2
- *Name:* lblNoteType
- *Text:* Note Type:

ComboBox1
- *Name:* cmbNoteTypes
- *Text:*
- *DropDownStyle:* DropDownList

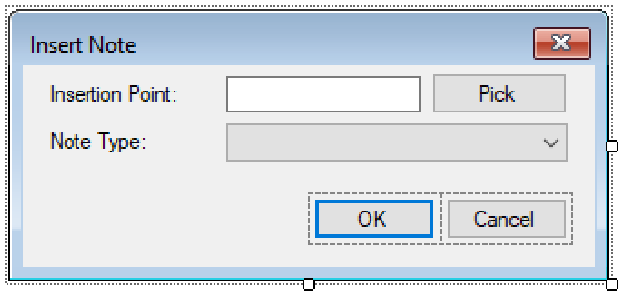
Button1
- *Name:* btnInsPoint
- *Text:* Pick

TextBox1
- *Name:* txtInsPoint
- *Text:*
- *Enabled:* False

When the dialog box is displayed, the user will be able to interact with all controls except the labels and TextBox1 which was disabled after changing the *Enabled* property to False. By default, labels usually don't allow for user interaction as they are commonly used to provide information to the user.

14. Click on the dialog box and then change these following properties:
    - *Name:* frmInsNote
    - *Text:* Insert Note
    - *StartPosition:* CenterScreen

Your form should now look similar to the following after you have made all of the property changes.

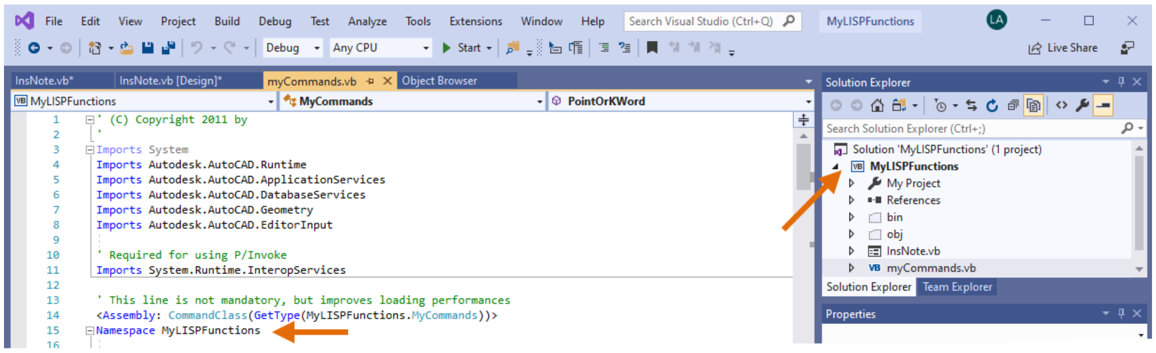


15. Right-click over the dialog box in the form editor and click View Code.

16. At the very top of the Code Window, add the following `Imports` statements just below the Imports System.Windows.Forms statement. These will provide you with access to the methods defined in the *MyCommands.vb* file and parts of the AutoCAD .NET API.

```
Imports MyLISPFunctions.MyLISPFunctions
Imports Autodesk.AutoCAD
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices.Application
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput
```

Note: `MyLISPFunctions.MyLISPFunctions` represents <Solution name>.<Namespace>. If you used a different name for the solution or namespace, make sure to adjust as needed.

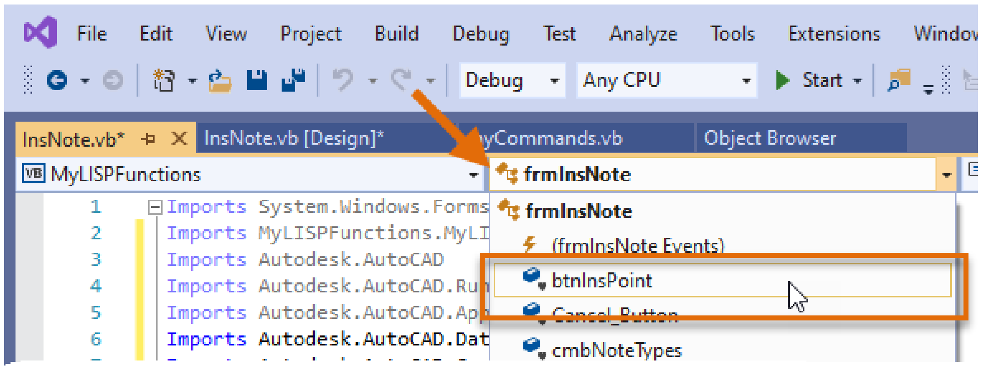


```
' (C) Copyright 2011 by
'
Imports System
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

' Required for using P/Invoke
Imports System.Runtime.InteropServices

' This line is not mandatory, but improves loading performances
<Assembly: CommandClass(GetType(MyLISPFunctions.MyCommands))>
Namespace MyLISPFunctions
```

17. Just below the `Public Class frmInsNote` statement, enter the following code:

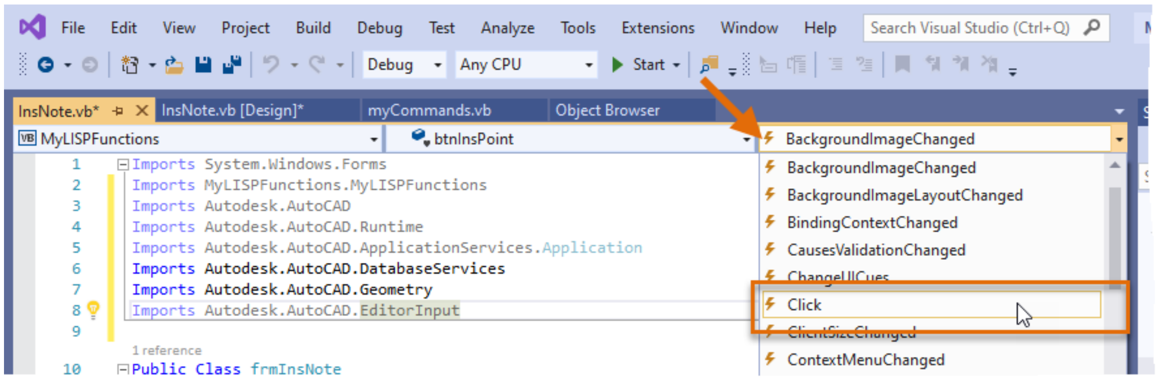
```
Private Const c_InsPoint As Integer = 10
Private Const c_NoteType As Integer = 2
Private g_InsPoint As Point3d
Public DialogValues As ResultBuffer
Private g_NoteType As String
```

18. In the Code Window, Class Name drop-down list, choose `btnInsPoint`.



```
Imports System.Windows.Forms
Imports MyLISPFunctions.MyLI
Imports Autodesk.AutoCAD
Imports Autodesk.AutoCAD.Ru
Imports Autodesk.AutoCAD.Ap
Imports Autodesk.AutoCAD.Dat
```

19. From the Method Name drop-down list, choose `Click`.



```
Imports System.Windows.Forms
Imports MyLISPFunctions.MyLISPFunctions
Imports Autodesk.AutoCAD
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices.Application
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

1 reference
Public Class frmInsNote
```

20. Click inside the newly added `btnInsPoint_Click` method near the bottom of the Code Window. Then enter the following code for the `btnInsPoint_Click` method:

```
Private Sub btnInsPoint_Click(ByVal sender As Object, _
                              ByVal e As EventArgs) _
                              Handles btnInsPoint.Click
   Dim docEditor As EditorInput.Editor
   docEditor = DocumentManager.MdiActiveDocument.Editor

   ' Hide the dialog box to interact with the drawing window
   Dim docEditorInt As EditorInput.EditorUserInteraction
   docEditorInt = docEditor.StartUserInteraction(Me)

   Try
       Dim pPointOpts As New PromptPointOptions(vbLf + _
                                 "Specify insertion point: ")

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

       If pPointResult.Status = PromptStatus.OK Then
           g_InsPoint = pPointResult.Value

           txtInsPoint.Text = pPointResult.Value.ToString()
       End If
   Catch ex As System.Exception
       MsgBox(ex.Message, MsgBoxStyle.Critical, "Error")
   Finally
       ' Redisplay the dialog box and end user interaction
       docEditorInt.End()
   End Try
End Sub
```

21. Add the `LostFocus` method for the `cmbNoteTypes` control like you added the `Click` method for the `btnInsPoint` control in Steps 18 and 19. Then add the following code for the `LostFocus` event:

```
Private Sub cmbNoteTypes_LostFocus(ByVal sender As Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles cmbNoteTypes.LostFocus
    ' Get the string of the current item
    ' and assign it to the global variable
    Try
        g_NoteType = cmbNoteTypes.SelectedItem.ToString()
    Catch
        g_NoteType = ""
    End Try
End Sub
```

The `LostFocus` method is triggered when another control in the dialog box is clicked or receives focus.

22. Add the `Shown` method for the `frmInsNote` dialog box like you added the `Click` method for the `btnInsPoint` in Steps 18 and 19. Choose (frmInsNote Events) from the Class Name drop-down list. Then add the following code for the `Shown` event:

```
Private Sub frmInsNote_Shown(ByVal sender As Object,
                             ByVal e As System.EventArgs) _
                             Handles Me.Shown
    ' Setup the values for the ComboBox control
    If cmbNoteTypes.Items.Count = 0 Then
        cmbNoteTypes.Items.Add("General Disclaimer")
        cmbNoteTypes.Items.Add("Confidential")
        cmbNoteTypes.Items.Add("Draft")
    End If

    ' Process the values passed in
    If Not DialogValues = Nothing Then
        ' Step through each value
        For Each val As TypedValue In DialogValues

            ' Check the type of data passed
            Select Case CInt(val.TypeCode)
                ' Get the value for the insertion point
                Case 10
                    g_InsPoint = val.Value
                    txtInsPoint.Text = g_InsPoint.ToString()
                ' Get the value for the note type
                Case 2
                    g_NoteType = val.Value
                    cmbNoteTypes.Text = g_NoteType
            End Select
        Next
    End If
End Sub
```

The `Shown` method is triggered when the dialog box is displayed.

23. Replace the code inside of the `Private Sub OK_Button_Click` method, so it looks like the following code:

```
Private Sub OK_Button_Click(ByVal sender As System.Object,
                            ByVal e As System.EventArgs) _
                            Handles OK_Button.Click
    Me.DialogResult = System.Windows.Forms.DialogResult.OK
    Me.Close()

    ' Store current values in a list of dotted pairs
```

```
     DialogValues = New ResultBuffer(
         New TypedValue(LispDataType.ListBegin),
         New TypedValue(LispDataType.Int16, c_InsPoint),
         New TypedValue(LispDataType.Point3d, g_InsPoint),
         New TypedValue(LispDataType.DottedPair),
         New TypedValue(LispDataType.ListBegin),
         New TypedValue(LispDataType.Int16, c_NoteType),
         New TypedValue(LispDataType.Text, g_NoteType),
         New TypedValue(LispDataType.DottedPair))
End Sub
```

24. In the Solution Explorer, double-click *myCommands.vb*.
25. Near the top of the Code Window, below the last `Imports` statement, enter the following code statements:

```
 ' Add reference to the frmInsNote dialog box
 ' (Form Name value, not file name)
 Imports MyLISPFunctions.frmInsNote
```

26. Click before the `End Class` statement at the bottom of the *MyCommands.vb* file, and then press Enter twice to add a few blank lines. On one of the new blank lines, enter the following code to define a custom AutoLISP function to display the dialog box:

```
     ' Defines an AutoLISP function that displays the
     ' dialog box frmInsNote
     ' Usage: (displayinsnote)
     '        (displayinsnote (cons 2 "Draft"))
     <LispFunction("DisplayInsNote")> _
     Public Function DisplayInsNote(ByVal rb As ResultBuffer)
         Dim dialog As New frmInsNote
         dialog.DialogValues = rb
         dialog.ShowDialog()

         If dialog.DialogResult = _
                Windows.Forms.DialogResult.OK Then
             ' Return a list of dotted pairs based
             ' on the selected responses
             Return dialog.DialogValues
         Else
             ' Return nil if the dialog box is canceled
             Return New TypedValue(LispDataType.Nil)
         End If
     End Function
 End Class
```

27. Save and build the DLL and then load it into AutoCAD.
28. At the Command prompt, enter **(displayinsnote)**.

29. In the dialog box, click the Pick button and specify a point in the drawing. Then choose a value from the drop-down list and click OK.

    You should notice the specified values are returned as a list. You can also use the same structure to set the default values of the dialog box. The example is designed to allow a partial list of the options to be set as well, so you could set the default value of the ComboBox control and not the insertion point.

    Note: Not all of the error handling is in place for this example as it would make the example much more complicated and detract from what is being demonstrated.

30. The `DisplayInsNote` function is designed to accept a list like the one it returns. At the Command prompt, enter **(displayinsnote (cons 2 "Draft"))**.

    This time the dialog box is populated with the Draft item current in the ComboBox control.

The *MyNote.lsp* file that is included with the materials for this session demonstrates how to use the `DisplayInsNote` function in an AutoLISP routine and how to parse the return value.

```
;; Example of using the DisplayInsNote function
(defun c:MyNote ( / valList noteType noteIns noteText)
  (setq valList (displayinsnote))

  (if valList
    (progn
      (setq noteType (cdr (assoc 2 valList))
            noteIns (cdr (assoc 10 valList))
            noteText "")

      (if (= noteIns nil)(setq noteIns (list 0,0,0)))

      (setq noteText (**getNoteText** noteType))
      (command "._-text" noteIns 1 0 noteText)
    )
  )
)
(prompt "\n\nType MYNOTE to insert a standard note\n")

;; Gets the actual note text
(defun **getNoteText** (noteType / )
  (cond
    ((= (strcase noteType) "DRAFT")
        "DRAWING IS A DRAFT ONLY"
    )
    ((= (strcase noteType) "CONFIDENTIAL")
        "DRAWING IS CONFIDENTIAL"
    )
    ((= (strcase noteType) "GENERAL DISCLAIMER")
        (strcat "THIS TECHNICAL DOCUMENT HAS BEEN "
                "PREPARED TO MEET INDUSTRY STANDARDS AND PRACTICES.")
      )
    )
  )
(princ)
```

10 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:

- **AutoCAD Online Help System –** The AutoCAD .NET Developer's Guide in the AutoCAD online Help system contains a lot of information on using the AutoCAD Managed .NET API to create custom programs.

  To access the online help, go to: https://www.autodesk.com/autocad-net-developers-guide

- **ObjectARX SDK –** While it is named the ObjectARX SDK, it contains many samples for the AutoCAD Managed .NET API. For information on the ObjectARX SDK, see https://www.autodesk.com/developer-network/platform-technologies/autocad/objectarx
- **Through the Interface (Blog)** – Kean Walmsley (Senior Manager, SW Architect), offers a wide range of information on using the various programming APIs that are available for use with AutoCAD. Visit Kean's blog at https://through-the-interface.typepad.com/
- **Autodesk Developer's Network –** If you are serious about developing applications for AutoCAD, you should consider becoming a registered Autodesk Developer. For information on registering as an Autodesk Developer, see https://www.autodesk.com/adn
- **Autodesk Discussion Forums –** The Autodesk forums provide peer-to-peer networking and some interaction with Autodesk moderators. You can ask a question about anything in AutoCAD and get a response from a fellow user or Autodesk employee. To access the .NET forums, go to https://forums.autodesk.com, click AutoCAD, and then click .NET)
- **AUGI Forums** – The AUGI forums provide peer-to-peer networking where you can ask questions about virtually anything in AutoCAD or Autodesk software related and get a response from a fellow user. Visit AUGI at https://www.augi.com
- **Internet** – There are tutorials and information scattered across the Internet that you can read to learn more VB.NET as well as some specific to using the AutoCAD Managed .NET API. Use your favorite search engine, such as Google or Bing, to find this information.
- **Books** – There are many books out there on VB.NET, but there are currently no books in print any more that cover using the AutoCAD Managed .NET API. The only book that I was aware of, which I did not review it, was VB.NET Programming for AutoCAD Customization by Jerry Winters.

Appendix 1: AutoCAD .NET Wizard Not Showing in Visual Studio

The AutoCAD .NET Wizard isn't required for you to create a new project, but it can simplify the process.

Here are some things to try if the AutoCAD .NET Wizards fail to install or Visual Studio doesn't recognize them as being installed.

AutoCAD .NET Wizard Failed to Install the Files

The AutoCAD 2022 .NET Wizard installs the files by default under the following locations:

- **Visual Basic** – *%USERPROFILE%\Documents\Visual Studio 2019\Templates\ProjectTemplates\Visual Basic\Autodesk/AutoCAD 2022 VB plug-in.zip*
- **Visual C#** – *%USERPROFILE%\Documents\Visual Studio 2019\Templates\ProjectTemplates\Visual C#\Autodesk/AutoCAD 2022 CSharp plug-in.zip*

If the *Autodesk* folders and/or ZIP files are missing, try:

- Making sure Visual Studio 2019 or Visual Studio 2019 Community are installed first.
- Running the installer as an Administrator.
- Restart your workstation and try installing the wizards again.

If that still fails, craete the *Autodesk* folder under the previously mentioned locations and then download/rename the ZIP files from the following locations:

- **Visual Basic** – https://github.com/ADN-DevTech/AutoCAD-Net-Wizards/blob/ForAutoCAD2022/AutoCADNetWizardsInstaller/Output/AutoCAD%20VB%20plug-in.zip
- **Visual C#** – https://github.com/ADN-DevTech/AutoCAD-Net-Wizards/blob/ForAutoCAD2022/AutoCADNetWizardsInstaller/Output/AutoCAD%20CSharp%20plug-in.zip

AutoCAD .NET Wizard is Installed but Visual Studio Fails to Find the Templates

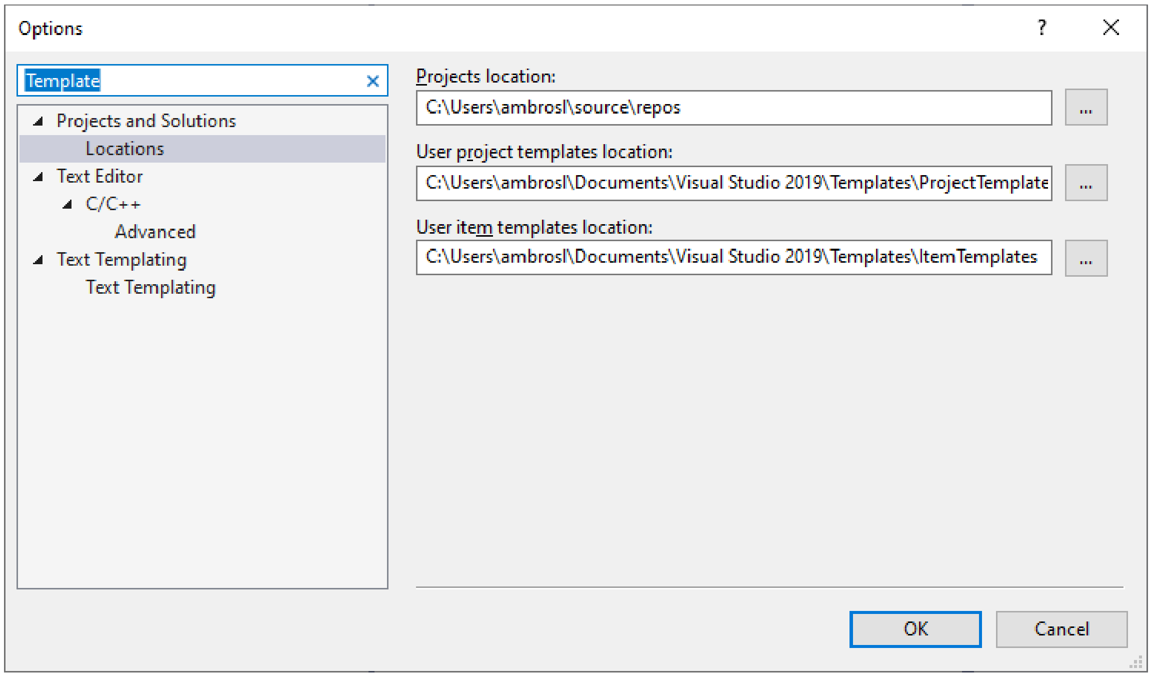
The AutoCAD 2022 .NET Wizard files are under the following locations, but Visual Studio still doesn't recognize them:

- **Visual Basic** – *%USERPROFILE%\Documents\Visual Studio 2019\Templates\ProjectTemplates\Visual Basic\Autodesk/AutoCAD 2022 VB plug-in.zip*
- **Visual C#** – *%USERPROFILE%\Documents\Visual Studio 2019\Templates\ProjectTemplates\Visual C#\Autodesk/AutoCAD 2022 CSharp plug-in.zip*

Visual Studio looks for Project and Item Templates in specific locations, and these locations can be user defined. To see where Visual Studio is looking for the templates in your installation, do the following:

1. Start Microsoft Visual Studio 2019. (Click Start menu > Visual Studio 2019)
2. In Microsoft Visual Studio, on the Visual Studio 2019 start page, click Continue without Code.

3. Click the Tools menu > Options.
4. In the Options dialog box, in the Search Options text box, enter **Template**.



Projects location:
C:\Users\ambrosl\source\repos

User project templates location:
C:\Users\ambrosl\Documents\Visual Studio 2019\Templates\ProjectTemplate

User item templates location:
C:\Users\ambrosl\Documents\Visual Studio 2019\Templates\ItemTemplates

5. Under Projects and Solutions, select Locations.
6. Verify that the locations specified are where the Autodesk folders and AutoCAD .NET Wizards are placed, and if not move the ZIP files or change the locations in which Visual Studio is looking for templates.

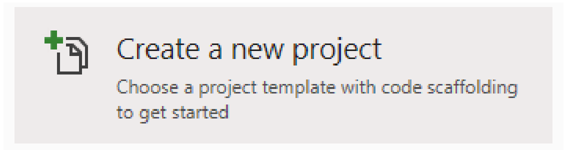
When All Else Fails… Create the Project from Scratch

If you can't still get the AutoCAD .NET Wizards to work in Visual Studio, proceed to Appendix 2: Create a New VB.NET Project without the .NET Wizard.

Appendix 2: Create a New VB.NET Project without the .NET Wizard

This section explains how to create a new VB.NET project from scratch without the assistance of the AutoCAD .NET Wizard as described earlier in the section Create a New VB.NET Project with the .NET Wizard.

1. Start Microsoft Visual Studio 2019. (Click Start menu > Visual Studio 2019)
2. In Microsoft Visual Studio, click Create a New Project (or click File menu > New Project).

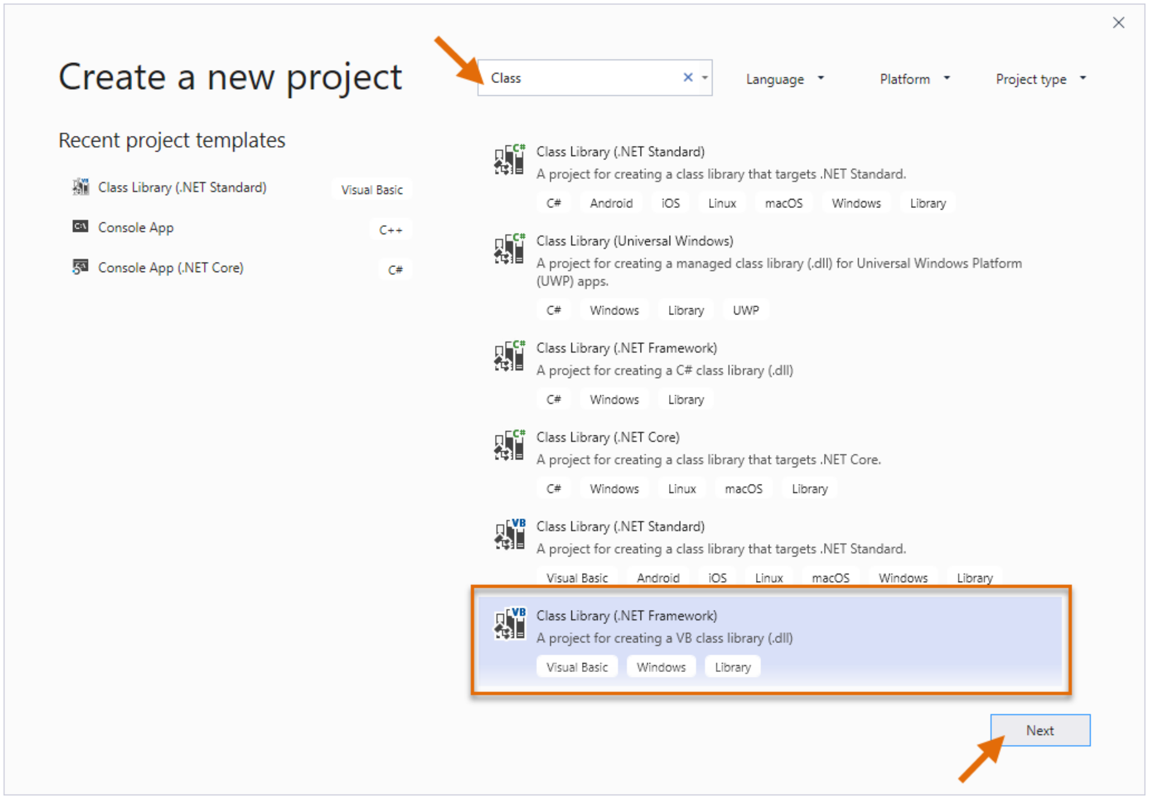


Create a new project

Choose a project template with code scaffolding to get started

3. In the Create a New Project dialog box, in the Search text box, type **Class**.
4. Choose Class Library (.NET Framework) for Visual Basic (VB) and click Next.

   Make sure it is for Visual Basic (VB) and not C#, and not the .NET Standard project type.



Create a new project

Class

Language | Platform | Project type

Recent project templates

Class Library (.NET Standard)	Visual Basic
Console App	C++
Console App (.NET Core)	C#

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

Class Library (Universal Windows)
A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.
C# Windows Library UWP

Class Library (.NET Framework)
A project for creating a C# class library (.dll)
C# Windows Library

Class Library (.NET Core)
A project for creating a class library that targets .NET Core.
C# Windows Linux macOS Library

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
Visual Basic Android iOS Linux macOS Windows Library

Class Library (.NET Framework)
A project for creating a VB class library (.dll)
Visual Basic Windows Library

5. In the Configure Your New Project dialog box, in the Project Name text box, type **MyLISPFunctions**.
6. Under Location, click the Ellipsis […] button.

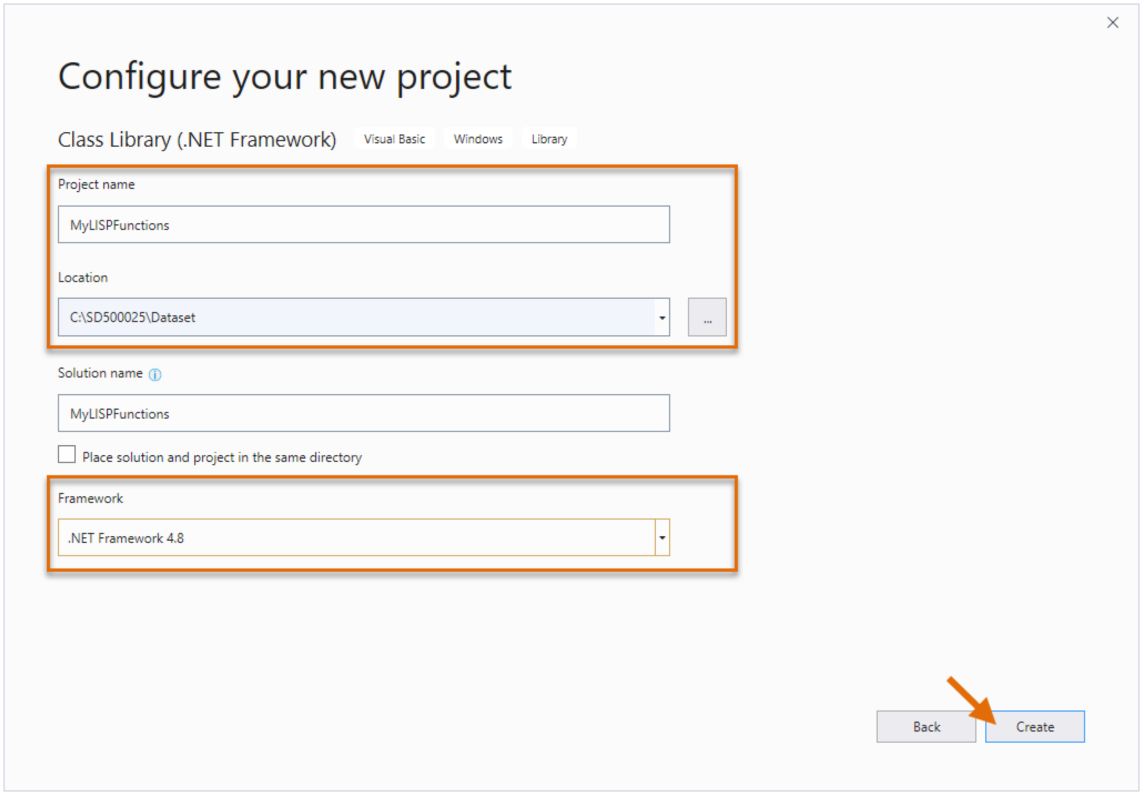
7. In the Project Location dialog box, browse to the location of the dataset for this session. Double click the Dataset folder and click Select Folder.

   The path now might look something like:

   C:\Dataset\SD500025

8. Click the Framework drop-down list and choose .NET Framework 4.8.

   If you don't have 4.8 installed, you can choose .NET Framework 4.7 as well.



Configure your new project

Class Library (.NET Framework) Visual Basic Windows Library

Project name: MyLISPFunctions

Location: C:\SD500025\Dataset

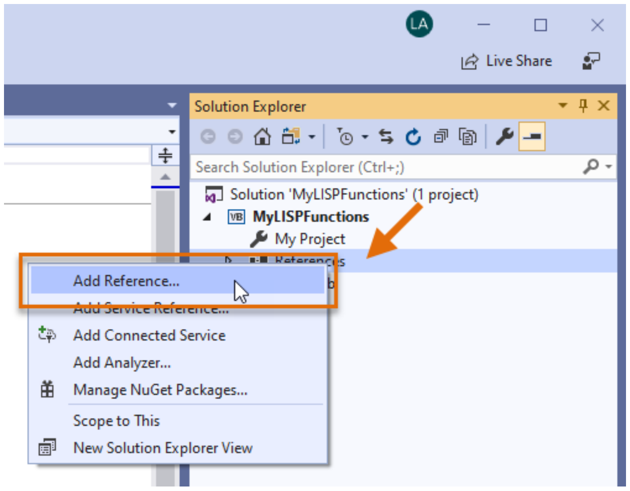
Solution name: MyLISPFunctions

☐ Place solution and project in the same directory

Framework: .NET Framework 4.8

9. Click Create.

10. In the Solution Explorer, right-click the References node and click Add Reference.

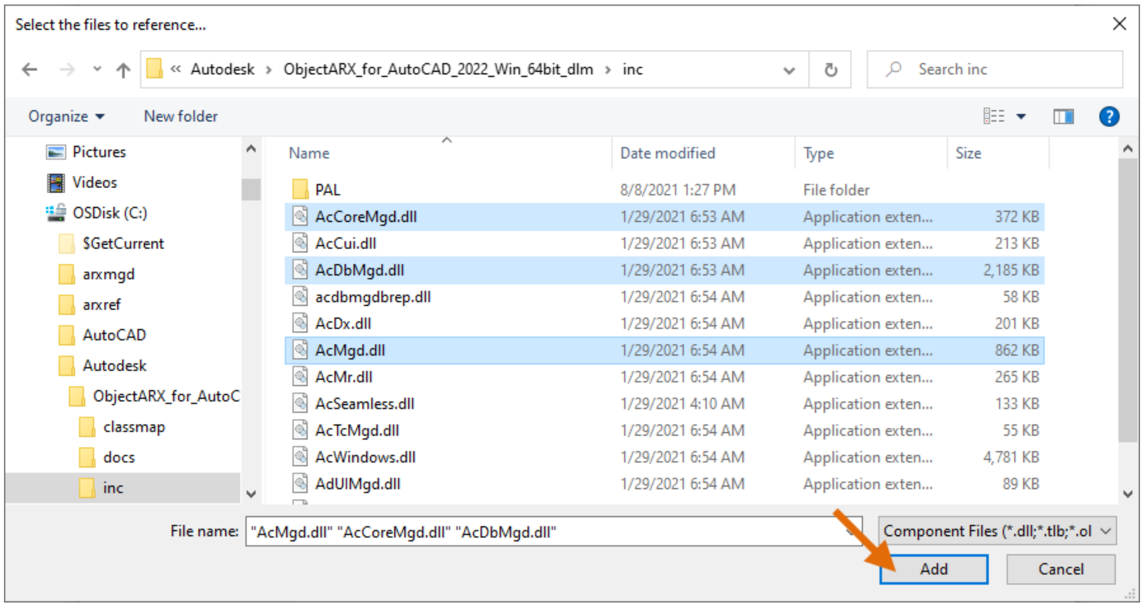


11. In the Add Reference dialog box, Browse tab, click Browse and browse to the folder in which you installed the ObjectARX SDK.

    The ObjectARX 2022 SDK is installed to the following location by default:

    C:\Autodesk\ObjectARX_for_AutoCAD_2022_Win_64bit_dlm

12. Continue browsing to the *Inc* folder. Press and hold Ctrl, and select the *AcCoreMgd.dll*, *AcDbMgd.dll* and *AcMdg.dll*. Click Add.



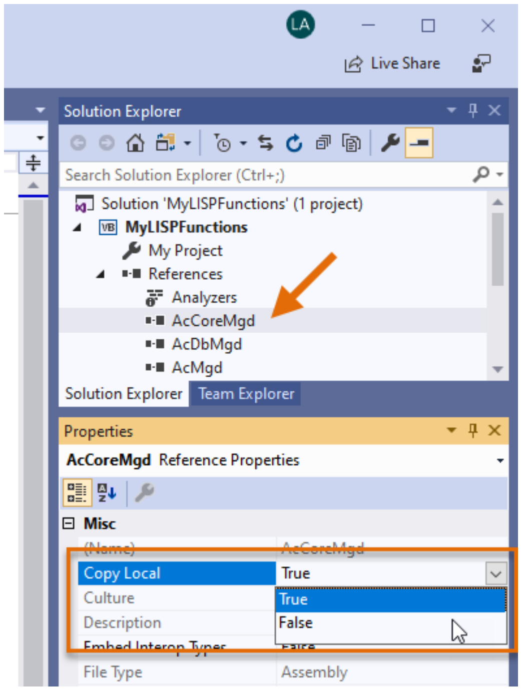
Name	Date modified	Type	Size
PAL	8/8/2021 1:27 PM	File folder	
AcCoreMgd.dll	1/29/2021 6:53 AM	Application exten...	372 KB
AcCui.dll	1/29/2021 6:53 AM	Application exten...	213 KB
AcDbMgd.dll	1/29/2021 6:53 AM	Application exten...	2,185 KB
acdbmgdbrep.dll	1/29/2021 6:54 AM	Application exten...	58 KB
AcDx.dll	1/29/2021 6:54 AM	Application exten...	201 KB
AcMgd.dll	1/29/2021 6:54 AM	Application exten...	862 KB
AcMr.dll	1/29/2021 6:54 AM	Application exten...	265 KB
AcSeamless.dll	1/29/2021 4:10 AM	Application exten...	133 KB
AcTcMgd.dll	1/29/2021 6:54 AM	Application exten...	55 KB
AcWindows.dll	1/29/2021 6:54 AM	Application exten...	4,781 KB
AdUIMgd.dll	1/29/2021 6:54 AM	Application exten...	89 KB

File name: "AcMgd.dll" "AcCoreMgd.dll" "AcDbMgd.dll"

Component Files (*.dll;*.tlb;*.ol

13. Click OK to accept the changes and close the Reference Manager.
14. Right-click over one of the newly added references and choose Properties.

15. In the Properties window, click the Copy Local field and then select False from the drop-down list.



Note: Setting Copy Local to False instructs Microsoft Visual Studio to exclude the referenced DLL in the build output for the project. If the referenced DLL is copied to the build output folder, it can cause unexpected results when you load your assembly file into AutoCAD.

16. Repeat Step 15 for *AcCoreMgd*, *AcDbMgd* and *AcMdg*.
17. In the Solution Explorer, right-click over the *Class1.vb* file and choose Rename.
18. Type **myCommands** and press Enter.
19. In the “You are renaming a file?” message box, click Yes to rename the class in the file as well.
20. In the Code Editor window for the *myCommands.vb* file, replace the all the text in the file with the following code statements:

```
Imports System
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.EditorInput

<Assembly: CommandClass(GetType(MyLISPFunctions.MyCommands))>
Namespace MyLISPFunctions

    Public Class MyCommands
    End Class
End Namespace
```

21. In the Solution Explorer, right-click over the project node MYLISPFunctions and choose Add > New Item.
22. In the Add New Item dialog box, Items list, select Resource File.
23. In the Name text box, select the default name and replace it with **myCommands** and click Add.
24. Click File menu > Save All.
25. Now that you have created the project, continue with the exercises in Section 5.

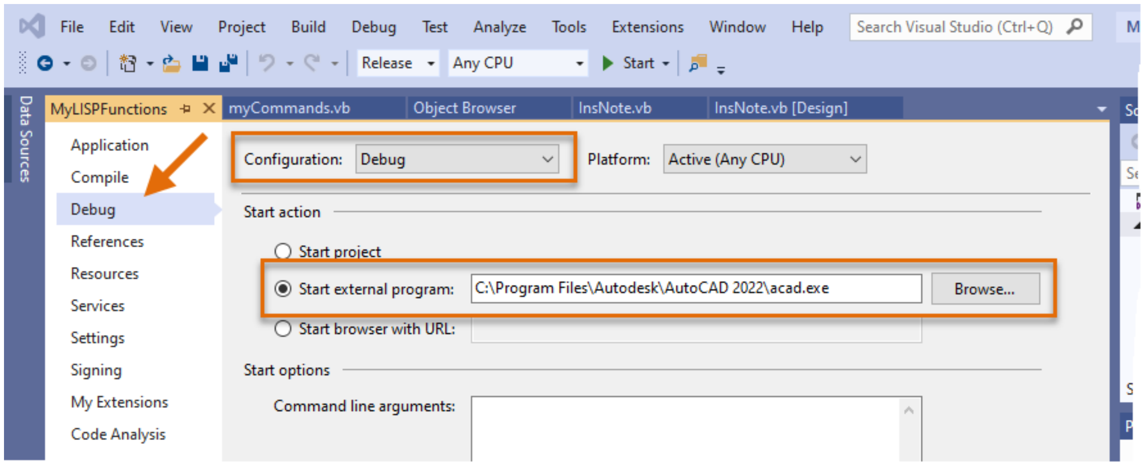
Appendix 3: Debugging and Building a VB .NET Project for Release

Instead of building a VB .NET project and then loading it into AutoCAD, you can launch AutoCAD from Visual Studio and allow you to debug your project while it is executing with in the AutoCAD program. This makes it much easier to identify problems in your code. Then once you have finished debugging your programs, you should build a Release version of your project instead of a Debug release.

Debugging a VB .NET Project

The following steps explain how to start debugging a VB .NET project and launch AutoCAD:

1. In Microsoft Visual Studio, open the project you want to debug and then click Project menu > MyLISPFunctions Properties.
2. In the Project Properties window, click Debug.



Configuration: Debug

Platform: Active (Any CPU)

Start action

Start project

Start external program: C:\Program Files\Autodesk\AutoCAD 2022\acad.exe

Start browser with URL:

Start options

Command line arguments:

3. In the Configuration drop-down list, choose Debug.
4. Under the Start Action section, click Start External Program option and then click the Browse button.

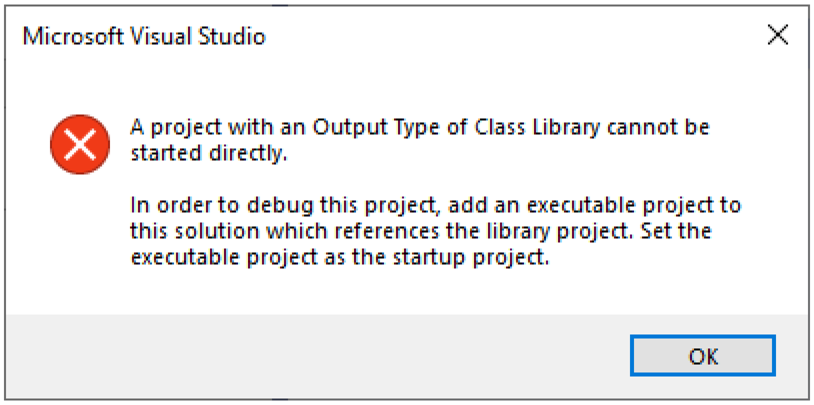
5. In the Select File navigation box, browse to and choose the *acad.exe* file in the AutoCAD Installation folder. Click Open.

   By default, the AutoCAD 2022 installation folder is:

   C:\Program Files\Autodesk\AutoCAD 2022

6. Once you have specified the AutoCAD executable location, you can start debugging your project by clicking Debug menu > Start Debugging.

   If the following error message is displayed, click Build menu > Configuration Manager. Then in the Configuration Manager, choose Debug from the Active Solution Configuration drop-down list and click Close.

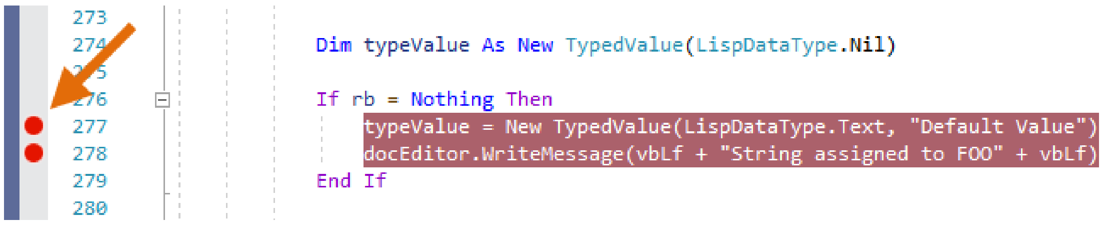


Microsoft Visual Studio

A project with an Output Type of Class Library cannot be started directly.

In order to debug this project, add an executable project to this solution which references the library project. Set the executable project as the startup project.

7. After AutoCAD starts, switch back over to Visual Studio and click in the margin to the left of the code statement you want execution to pause when running your code in AutoCAD.

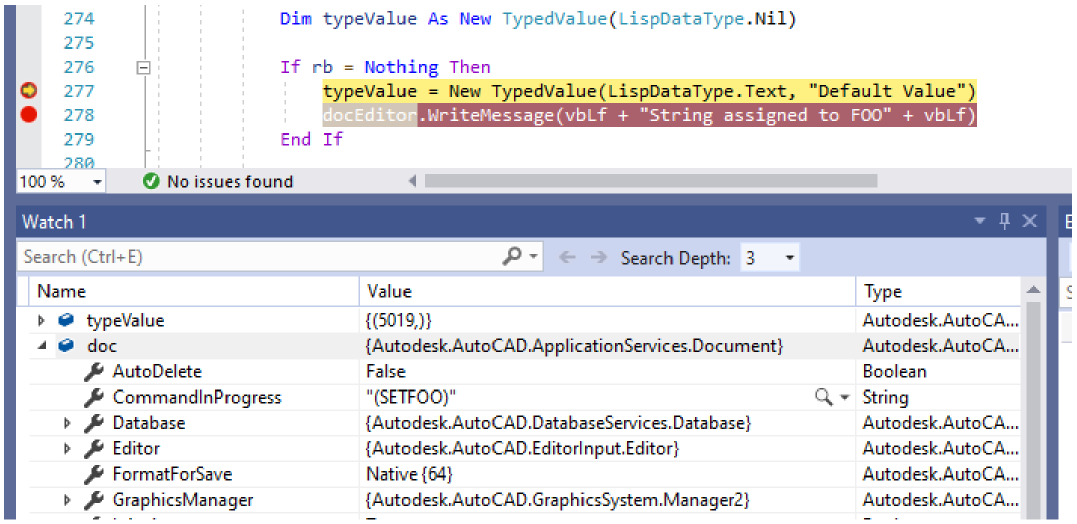


```
Dim typeValue As New TypedValue(LispDataType.Nil)

If rb = Nothing Then
    typeValue = New TypedValue(LispDataType.Text, "Default Value")
    docEditor.WriteMessage(vbLf + "String assigned to FOO" + vbLf)
End If
```

   This inserts what is known as a *breakpoint* which will interrupt execution.

8. Switch back to AutoCAD and load the debug version of the DLL for your project into AutoCAD with the NETLOAD command.

9. Execute your custom AutoLISP function or command. When the breakpoint is reached, execution is interrupted allowing you to check the values of variables in the current scope.



```
Dim typeValue As New TypedValue(LispDataType.Nil)

If rb = Nothing Then
    typeValue = New TypedValue(LispDataType.Text, "Default Value")
    docEditor.WriteMessage(vbLf + "String assigned to FOO" + vbLf)
End If
```

100 %   No issues found

Watch 1

Search (Ctrl+E)   Search Depth: 3

Name	Value	Type
typeValue	{(5019,)}	Autodesk.AutoCA...
doc	{Autodesk.AutoCAD.ApplicationServices.Document}	Autodesk.AutoCA...
AutoDelete	False	Boolean
CommandInProgress	"(SETFOO)"	String
Database	{Autodesk.AutoCAD.DatabaseServices.Database}	Autodesk.AutoCA...
Editor	{Autodesk.AutoCAD.EditorInput.Editor}	Autodesk.AutoCA...
FormatForSave	Native {64}	Autodesk.AutoCA...
GraphicsManager	{Autodesk.AutoCAD.GraphicsSystem.Manager2}	Autodesk.AutoCA...

10. Click the following from the Debug menu to step through your program:

- **Step Into** – Advances debugging into a function.
- **Step Over** – Moves debugging out of statements you don't want to debug.
- **Step Out** – Advances debugging outside of a function.

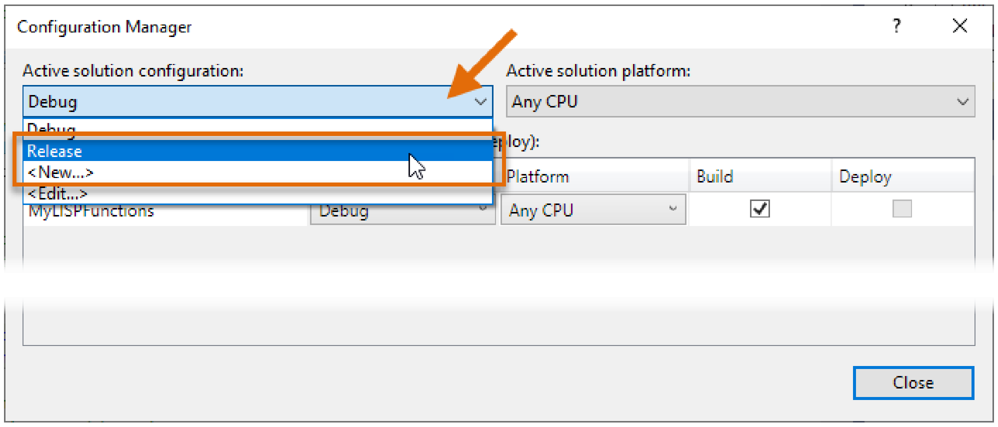
11. Click Debug > Continue to resme normal execution until the next breakpoint is reached.

12. Click Debug > Stop Debugging to end debugging and clos AutoCAD.

Building a Release Version of a VB .NET Project

When you are ready to use your custom functions and utilities created with VB .NET, you need to build your project for release. The following steps explain how to build a Release version of a VB .NET project:

1. In Microsoft Visual Studio, open the project you want to build and then click Build menu > Configuration Manager.



Configuration Manager

Active solution configuration: Debug

Debug
Release
<New...>
<Edit...>

Active solution platform: Any CPU

loy):

		Platform	Build	Deploy
MyLISPFunctions	Debug	Any CPU	☑	☐

Close

2. In the Configuration Manager, click the Active Solution Configuration drop-down list and choose Release.
3. Click Close to to save the changes made.
4. Click Build menu > Build MyLISPFunctions.

   By default, your DLL should be built to:

   C:\Dataset\SD500025\MyLISPFunctions\MyLISPFunctions\bin\Release\

Appendix 4: Using the acedGetSym and acedPutSym Methods

Note: This section demonstrates how to import functions that aren't part of the AutoCAD Managed .NET API. The examples here were required in with AutoCAD 2012 and earlier to get and set values in AutoLISP user-defined variables because the `GetListSymbol` and `SetLispSymbol` functions didn't exist at that time.

The way you work with AutoLISP variables is through the use of the methods `acedGetSym` and `acedPutSym`, which are part of the ObjectARX programming language. These two methods are brought into VB.NET by using a programming process called P/Invoke. The definitions of the ObjectARX functions look like:

```
' Get the ObjectARX functions acedGetSym and acedPutSym
' so we can get/set AutoLISP variables
<DllImport("acad.exe", CharSet:=CharSet.Unicode,
    CallingConvention:=CallingConvention.Cdecl,
    EntryPoint:="acedGetSym")>
Shared Function acedGetSym(ByVal varName As String,
                           <Out()> ByRef varValue As IntPtr) As Integer
End Function

<DllImport("acad.exe", CharSet:=CharSet.Unicode,
    CallingConvention:=CallingConvention.Cdecl,
    EntryPoint:="acedPutSym")>
Shared Function acedPutSym(ByVal varName As String,
                           ByVal varValue As IntPtr) As Integer
End Function


' If you want to use the acedGetSym and acedPutSym functions
' in AutoCAD 2017 and later, the function definitions would look like
' <DllImport("accore.dll", CharSet:=CharSet.Unicode,
'     CallingConvention:=CallingConvention.Cdecl,
'     EntryPoint:="?acedGetSym@@YAHPEB_WPEAPEAUresbuf@@@Z")>
' Shared Function acedGetSym(ByVal varName As String,
'                            <Out()> ByRef varValue As IntPtr) As Integer
' End Function

' <DllImport("accore.dll", CharSet:=CharSet.Unicode,
'     CallingConvention:=CallingConvention.Cdecl,
'     EntryPoint:="?acedPutSym@@YAHPEB_WPEAUresbuf@@@Z")>
' Shared Function acedPutSym(ByVal varName As String,
'                            ByVal varValue As IntPtr) As Integer
' End Function
```

In addition to the above method definitions, you will also need to include a reference to the `System.Runtime.InteropServices` namespace. Add the following to the top of your code module:

```
' Required for using P/Invoke
Imports System.Runtime.InteropServices
```

Once the method definitions are added, along with the namespace import you can then use the `acedGetSym` and `acedPutSym` in your application. The following two examples demonstrate how to get and set the value of a variable named Foo.

```
' Defines an AutoLISP function that gets the value of a variable
<LispFunction("FetchFoo_Legacy")>
Public Function FetchFoo_Legacy(ByVal rb As ResultBuffer)
    Dim intPtrRB As IntPtr = IntPtr.Zero
    Dim esVal As Integer = acedGetSym("foo", intPtrRB)
    Dim typeValue As New TypedValue(LispDataType.Nil)

    If (intPtrRB <> IntPtr.Zero) Then
        Dim newRb As ResultBuffer = ResultBuffer.Create(intPtrRB, True)

        For Each val As TypedValue In newRb
            If val.TypeCode >= 5000 Then
                Dim lspDataType As LispDataType = val.TypeCode
                typeValue = New TypedValue(LispDataType.Text,
                                           lspDataType.ToString())
            Else
                Dim dxfCodeType As DxfCode = val.TypeCode
                typeValue = New TypedValue(LispDataType.Text,
                                           dxfCodeType.ToString())
            End If
        Next
    End If

    Return typeValue
End Function

' Defines an AutoLISP function that sets the value of a variable
<LispFunction("SetFoo")> _
Public Function SetFoo(ByVal rb As ResultBuffer)
    Dim typeValue As New TypedValue(LispDataType.Text, "My Value")
    Dim newRb As New ResultBuffer
    newRb.Add(typeValue)

    Dim esVal As Integer = acedPutSym("foo", newRb.UnmanagedObject)
    Return typeValue
End Function
```