# Lab 8 – User Interface (Non CUI API)

In this lab, there will be three sections that will show how to use the .NET API to do the following:

1. Define a custom context menu.
2. Add a tab to the AutoCAD’s Options dialog.
3. Add a label to the user control that will run a command using Drag and Drop.

**Custom Context Menu**

To add a context menu without running a command we need to have our code run when our dll is loaded. To perform the load-time initialization, an AutoCAD .NET application can implement the IExtensionApplication .NET interface, and expose an assembly-level attribute which specifies this class as the ExtensionApplication. The class can then respond to one-time load and unload events. Paste these steps into the Class you have been using for the previous labs. Or open the Lab8 project that already has these steps.

// 1. Modify the asdkClass1 class to implement the IExtensionApplication interface.

// Add ": IExtensionApplication" to the class declaration.

// Note: There are two required interfaces that need to be Implemented.

// These are the Initialize() and Terminate(). Create these functions using

// public void keywords.

// Copy and paste the functions so steps 23 and 37 are inside the Initialize

// procedure and step 24 is inside the Terminate procedure.

// Also add this attribute by uncommented the next line of code.

// (needs to be oustide of the class)

// This line is not mandatory, but improves loading performances

// [assembly: ExtensionApplication(typeof(Lab8.adskClass))]

// 2. Declare a member variable as ContextMenuExtension. It will be instantiated

// in step 6

// 3. Create a private void procedure named AddContextMenu.

// Note: Put the Closing Curley brace after step 12.

// 4. Declare an editor variable. Instantiate it by making it equal to the

// Editor property of the Application.DocumentManager.MdiActiveDocument.Editor

// 5. Add a try catch block.

// Note: put the closing curley brace for the try below step 11.

// Put the catch below this. The curley brace for the catch is

// after step 12.

// 6. Make the ContextMenuExtension declared in step 2 equal

// to a new ContextMenuExtension

// 7. Make the Title property of the ContextMenuExtension

// instantiated in step 7 equal to "Circle Jig". (it is going

// to run the command completed in Lab7).

// 8. Declare a MenuItem variable named mi. Instantiate by

// making it eaual to a new MenuItem. For the string parameter

// use a string like "Run Circle Jig".

// 9. The way the Context menu works is that for each menu entry, we specify a

// specific member function to be called handling the menu click event.

// Use the MenuItem Click event (mi.Click += )to specify that we want the

// click event to be handled by a function named CallbackOnClick. You will

// create this function in step 20 - 22.

// 10. Use the Add method of the MenuItems collection of the

// ContextMenuExtension instantiated in step 6. Pass in the

// MenuItem created in step 8.

// 11. Use the AddDefaultContextMenuExtension of the Application.

// Pass in the ContextMenuExtension

// 12. Use the editor variable created in step 4 and write a message.

// to the command line Something like"

// "Error Adding Context Menu: " + ex.Message

// 13. Create a procedure named RemoveContextMenu.

// Note: Put the closing curley brace after step 19.

// 14. Declare an editor variable. Instantiate it by making it

// equal to the Editor property of the MdiActiveDocument

// 15. Add a try catch block.

// Note: put the try closing curley brace below step 18.

// put the catch after this. put the catch closing curley brace after

// step 19.

// 16. Use an "if" statement and test if the ContextMenuExtension

// declared in step 2 is not null (!=)

// Note: put the closing curley brace below step 18

// 17. In the if statement, use RemoveDefaultContextMenuExtension

// of the Application. Pass in the ContextMenuExtension declared

// in step 2

// 18. Make the ContextMenuExtension declared in step 2 equal to null

// 19. Use the editor and write a message. Something like

// "Error Removing Context Menu: " + ex.Message

// 20. Create a private void function named CallbackOnClick. This is the

// function that will be called when the Menu Item is clicked. Set in step 9.

// Note: Put the closing curley brace after step 22.

// 21. Use the using statement and create a variable as a DcoumentLock.

// Instantiate it by making it equal to the .MdiActiveDocument.LockDocument

// method. (Because this event is running outside of the Document context we

// need to lock the document). By design, AutoCAD’s data is stored in documents,

// where commands that access entities within them have rights to make

// modifications. When we run our code in response to a context-menu click,

// we are accessing the document from outside the command structure.

// In order to unlock the document we simply dispose DocumentLock object

// returned on the original lock request.

// Note: put the using closing curley brace after step 22.

// 22. Call the CircleJig() function.

// Added from step 25

//acedPostCommand("CANCELCMD");

// 23. Inside the Initialize procedure. (created in step 1) add

// a call to AddContextMenu(). This will add the Context menu when this .NET dll

// is loaded.

// Note: Proceed to step 24 below

// 37. Inside the Initialize procedure. Uncomment this line after doing

// steps through 24 - 27

// This will run this function when the dll is loaded in AutoCAD

// AddTabDialog();

// 24. Inside the Terminate procedure. (created in step 1) add

// a call to RemoveContextMenu().

// Before proceding to step 25 build and netload the dll. Right click

// on the ModelSpace background. You should see the new MenuItem. "Circle Jig"

// 25. Uncomment this to fix behavior where command line

// is not automatically set back to normal command after running

// the MenuItem.

// using System.Runtime.InteropServices; //(move outside of the class needed for P/Invoke)

// acedPostCommand("CANCELCMD") ' Move this after CircleJig() in step 22

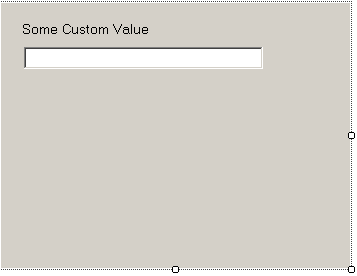
//[DllImport("acad.exe", CharSet = CharSet.Unicode, EntryPoint = "?acedPostCommand@@YAHPB\_W@Z")]

// public static extern bool acedPostCommand(string cmd);

**Adding a Page to the AutoCAD Options Dialog**

In this section of the you will add a new User Control which can be displayed as a page on the AutoCAD Options dialog. We can use this page to set default values for our application. In this example it will simply be a string member variable. Note: Use these instructions to create the user control and then copy in the steps. Or you can use the Lab8 project that already has the user control and the steps.

Add a User Control to the project. Name it something like “myCustomTab”. Add a Label and a TextBox so that it looks similar to the following:



Here are the steps for this section.

// Lab8 Steps 26 - 37. Add a tab to the Options dialog.

// To add a tab you need to subscribe to notifications when the options dialog

// is launched. This is done by passing the address of a member function to be called.

// You will also need to implement the callback function; the second argument

// passed into the callback is a "TabbedDialogEventArgs" object which we must

// use to call its AddTab member. AddTab takes a title string, and an instance

// of a TabbedDialogExtension object, which wraps our form. Within the constructor

// of TabbedDialogExtension, we pass a new instance of our usercontrol, and callback

// addresses. We can handle either OnOK, OnCancel or OnHelp.

// 26. First Declare a public static shared variable as a String. This variable will be set

// from our custom tab in the Options dialog. Name it something like "myVariable"

// 27. Add a publc static void procedure called AddTabDialog. We will call this procedure

// from the Initialize() function that was created in step 1

// Note: Put the closing curley brace after step 28.

// 28. Use the Application DisplayingOptionDialog event. Use += to specify

// the function that will be called when the Options command is run.

// you will create this function in step 29. (name it TabHandler)

// 29. Create a private static void function named TabHandler. This is the Sub that

// will be called when the Options dialog is displayed. (The name needs to be

// the name used in the Delegate parameter of step 28). The first parameter is an

// object name it something like sender.

// The second parameter is a TabbedDialogEventArgs.

// Use Autodesk.AutoCAD.ApplicationServices.TabbedDialogEventArgs e

// Note: Put the closing curley brace after step 33

// 30. Declare a variable as the user control that we are going to add to

// the Options dialog. (myCustomTab) Use the new keyword to instantiate it.

// 31. Delcare a variable as a TabbedDialogAction. Instantiate it using the new statement

// to create a new TabbedDialogAction for the parameter use the OnOk method of the

// User control variable from step 29. We can subscribe to, Ok,

// Cancel and Help. In this example, we chose to handle only OK.

// Note: The OnOk method will be added in Steps 35 - 36. (You can skip forward to these steps

// in the code window for the user form - myCustomTab - come back here of course)

// 32. Declare a variable as a TabbedDialogExtension. Instantiate it using the New statement

// to create a new TabbedDialogExtension for the control parameter pass in the User control variable

// from step 30. and the TabbedDialogAction pass in the TabbedDialogAction variable from step 31.

// 33. Use the AddTab method of the TabbedDialogEventArgs passed into the event.

// (named e). For the Name parameter use something like "Value for custom variable". For the

// TabbedDialogExtension parameter use the TabbedDialogExtension from step 32.

// Note: Proceed to step 34 in the TestTab function below.

// Run this command to get the value

[CommandMethod("testTab")]

public void TestTab()

{

Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;

// 34. Uncomment this line to print the value set from the Tab added in the previous

// steps to the command line.

// ed.WriteMessage(myVariable.ToString());

// Note: You may need to change myVariable to the name you used in step 26

// Proceed to steps 35 - 36 in the Code window for the UserControl. (myCustomTab)

// if you have not already done so. Also do step 37, close to step 23

// After steps 35-36 are complete Build, Load and run the AutoCAD OPTIONS to see

// the custom dialog. Set the value in the text box click Ok and then run

// this command you should see the value set in the Options dialog printed

// on the command line.

// After this proceed to steps 38-48 in the code

// window for UserControl1. (Adds functionality for Drag & Drop).

}

Copy these steps into the user control that will be shown in the Options dialog. (Needs to be inside the class).

// 35. Add a Public Sub named OnOk

// Note. Put the closing curley brace below step 36

// 36. Make the variable created in step 26 eqaul to

// TextBox1.Text. Need to qualify the variable with the class name.

// something like: adskClass.myVariable

// Note: There is one more step in this Custom Tab section of Lab8.

// Proceed to step 37 and uncomment the line that runs the

// AddTabDialog() procedure. (found after step 23).

// Before proceeding to the section where you will explore

// Drag & Drop from the PaletteSet (Steps Run AutoCAD and open the

// OPTIONS dialog. You should see the new tab.

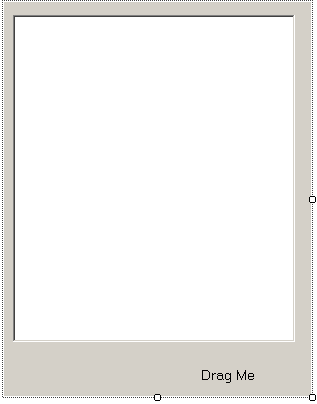
// Enter in some text in the TextBox and then

// click ok. Then run the testTab command. It should print

// the value you entered on the custom tab to the command line.

**Add Drag and Drop support to the PaletteSet created in Lab3**

In this section, we’ll add code which allows us to run a command when the user drags from the palette on to the AutoCAD editor. In order to support Drag and Drop, we first need an object to drag. Add an additional ‘Label’ below the TreeView, named ‘DragLabel’. From this label, we will be able to handle drag and drop into the AutoCAD editor. Here is a screenshot of the update to the user control. Note: you can update the project you have been working on or use the updated user control in the Lab8 project. (It has the following steps).



To detect when a drag event is taking place, we need to know when certain mouse operations take place.

Copy this step to the constructor for the user control

// 38. To detect when a drag event is taking place register the MouseMove event

// of the DragLabel. (DragLabel.MouseMove) Use += and the new keyword

// to create a new System.Windows.Forms.MouseEventHandler for the

// object parameter use the function we will create in steps 39-41.

// it is named

Copy these steps to the user control class

// 39. create a private void function named DragLabel\_MouseMove. Have it take

// two parameters and object and System.Windows.Forms.MouseEventArgs. Name

// the object sender and the MouseEventArgs e

// Note: Put the closing curley brace below step 41

// 40. It’s enough to see that we know when a mouse-move operation takes place.

// We can even go further to tell that the "left" mouse button is currently pressed

// Use an "if" statement. For the test see if the Left mouse button is being

// used: System.Windows.Forms.Control.MouseButtons == System.Windows.Forms.MouseButtons.Left

// Put the closing curley brace beow step 31.

// 41. (Ensure a reference to Windowsbase has been added).

// This is needed for System.Windows.DependencyObject()

// Call the DoDragDrop method of the Application. For the drag source

// paramter use the "this" keyword. For the data paramter use "this" as well.

// For Allowed effects use System.Windows.Forms.DragDropEffects.All

// For the Target paramter use the new statement and a class named "MyDropTarget"

// This class is created in steps 42-48. The DropTarget is how our MyDropTarget

// override is hooked up to the mechanism.

Copy steps 40-46 to the code window of the user control. A new class will be created.

// 42. Here we create a class that will detect when the object is "dropped"

// in the AutoCAD editor. Add a public class to the project called MyDropTarget which

// inherits from Autodesk.AutoCAD.Windows.DropTarget. (the name needs to be the

// same as used in step 41).

// Note: Put the closing curley brace below in step 48

// 43. Override the OnDrop procedure. (Use public override void) For the parameter

// use "System.Windows.Forms.DragEventArgs e"

// 44. Declare a variable as an Editor. Instantiate it by making it equal

// to Application.DocumentManager.MdiActiveDocument.Editor

// 45. Add a try catch block.

// Note: Put the try closing curley brace after step 47.

// put the catch after this and the closing curley brace after step 48

// 46. Use the using statement and declare a DocumentLock variable named

// docLock. Instantitate it using:

// Application.DocumentManager.MdiActiveDocument.LockDocument()

// Note: Put the using closing curley brace after step 47.

// Creating the variable will lock the document and unlock it

// when the vairable is disposed

// 47. Run the AddAnEnt procedure from lab 4. Need to

// qualify the name with the namespace and Class. (Lab8.adskClass.)

// 48. Add System.Exception ex to the Catch statement

// Use the Editor created in step 42 to display message to the

// user. If we get here something went wrong. Use something like

// this for the message parameter:

// "Error Handling OnDrop: " + ex.Message

// Note this is the End of Lab8. Run AutoCAD load the dll and run the

// Palette command. Drag the label to the drawing area.

// The AddAnEnd command should start when you drop it.

* Compile the code.
* Start AutoCAD.
* Netload the dll
* Right click on the drawing area and run the new context menu
* Run the OPTIONS command and notice the new tab. Add some text to the text box and click ok. Run the TESTTAB command. You should see the text from the options dialog printed on the command line.
* Run the Palette command. Drag the label onto the drawing area. The AddAnEnt command should start.

This is the End of Lab8