



OpenDCL Tutorial: DCL Has Evolved!

Barry Ralphs

Introduction: I know what you're thinking: "Why learn OpenDCL when there's VBA and .NET?" If you haven't already heard, VBA isn't going to be around much longer and do you really want to port your hundreds of Lisp routines to .NET? This tutorial will introduce you to the basics of OpenDCL and its abilities. We'll discuss how to easily design stunning GUIs (Graphical User Interfaces) for your Lisp routines, and we'll also cover advanced topics, such as securely embedding your UI into a VLX file for public distribution. You should have general AutoCAD® and AutoLISP® knowledge.

About the Author:

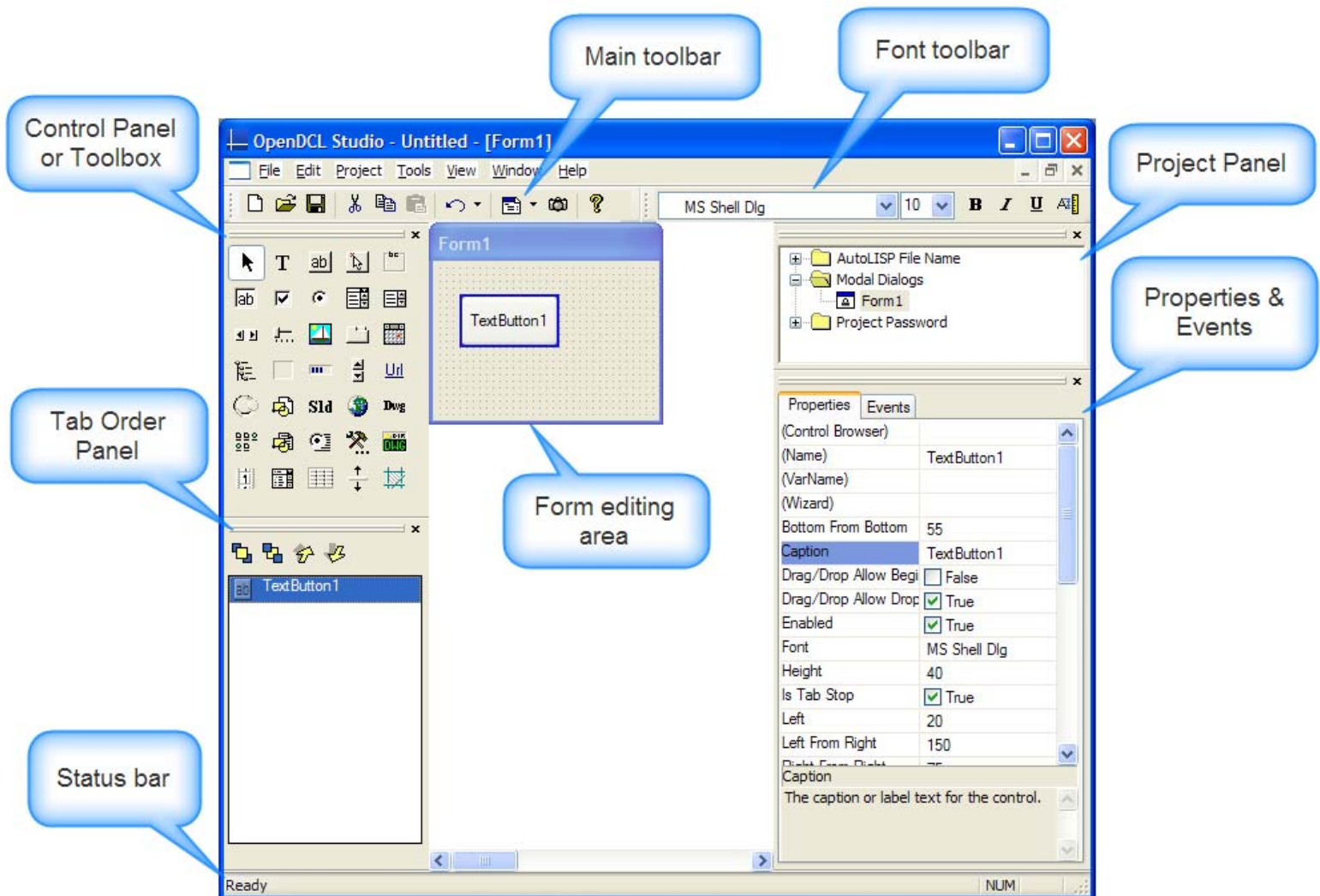
Barry Ralphs is currently the CAD manager for Tipping Mar & Associates, a structural engineering firm in Berkeley, CA. Prior to his current position, he spent nearly six years as the CAD manager for ARUP Los Angeles. Barry has been customizing AutoCAD® with OpenDCL and its predecessor ObjectDCL, since its introduction in 2001. He is the webmaster for OpenDCL.com, has written a number of the sample projects that install with OpenDCL Studio. He is also one of the lead beta testers for the project. You can usually find him in the OpenDCL support forums helping out other newcomers to the project under the username "BazzaCAD".
barry.ralphs@gmail.com

Getting started with OpenDCL

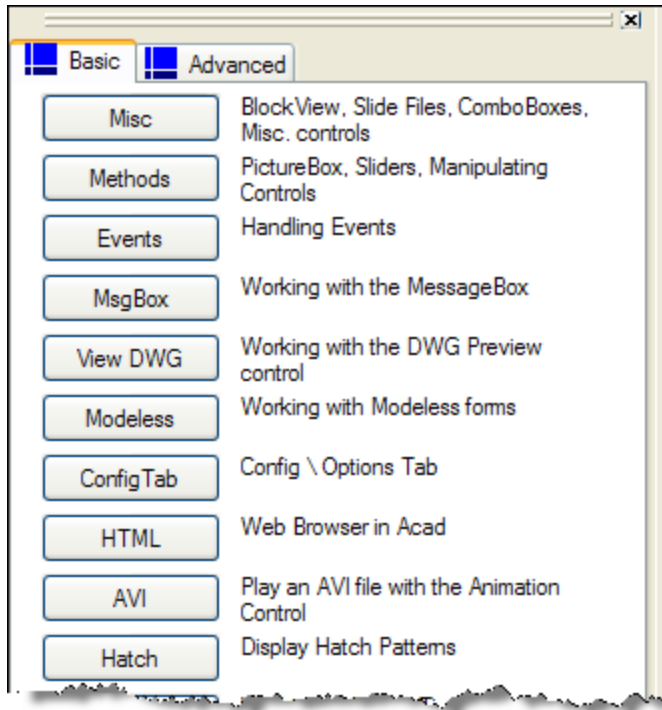
The first thing you'll need to do is visit the OpenDCL.com website and click the "Download" link on the sidebar of the page. You'll then be taken to the download page on SourceForge.net for the OpenDCL project. Under the list of Packages to download, select "OpenDCL Studio". At this point you won't need to download the "Runtime" as it will be installed with the "Studio". Now under the list of Filenames, select your preferred language code to download, English (ENU), German (DEU), Chinese (CHS), Spanish (ESM), and French (FRA). After the download is complete, just run the MSI file and it will be installed in less than 1 minute.



You should now have an "OpenDCL Studio" icon on your desktop; double click it now to launch it. See the image below for more information about the Studio workspace.



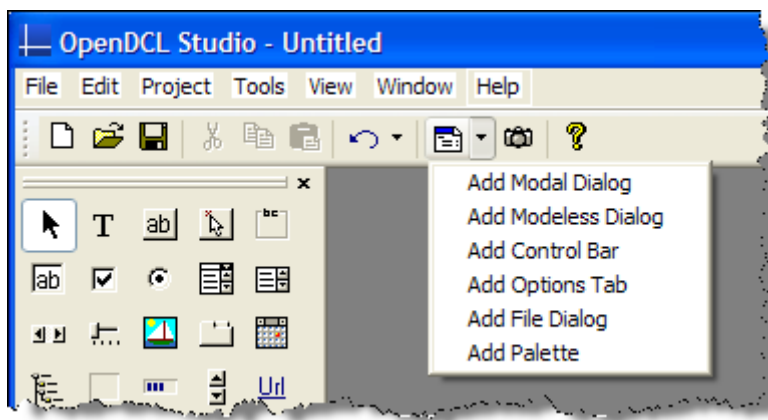
Now is a good time to see what OpenDCL [ODCL] can do. The Studio installs a number of samples that demonstrate its capabilities. Open Windows Explorer and browse to “C:\Program Files\OpenDCL Studio\ENU\Samples”. This is the typical install location for the English version of ODCL on a Windows 32 bit OS. Drag the “_MasterDemo.lsp” file into your AutoCAD drawing window.



This will display a Docked Control Bar form with a number of buttons. Each button will launch a different example. Click through each of the samples, to see what ODCL can do. After exploring the sample projects, you're ready to start your first project with OpenDCL.

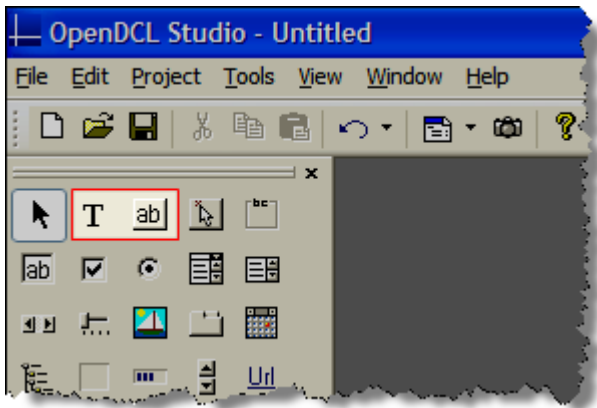
Basic Level 1: Creating your first “Hello World” project.

In OpenDCL Studio, pick “Add Modal Dialog” from the main toolbar or the “Project” pull-down.



A new form will appear in the editing area. The form's grid is provided as an aid to positioning controls. You can set the spacing of the grid by selecting the Tools menu > Grid Spacing. You can also choose not to display it at all by setting the spacing to 0.

For this example resize the form by dragging the bottom right border, or by changing the “Height” & “Width” in the Properties Panel to 80 & 250.



Now place a “Label” and a “Text Button” on the form by clicking their buttons on the Control Panel Toolbox, then dragging a rectangle out on the form’s surface.

Change the “Caption” property of the button to “OK”. The form should now look similar to this:



Save the project file as “HelloWorld.odcl” to an AutoCAD support directory or another directory of your choice that is in AutoCAD's search path.

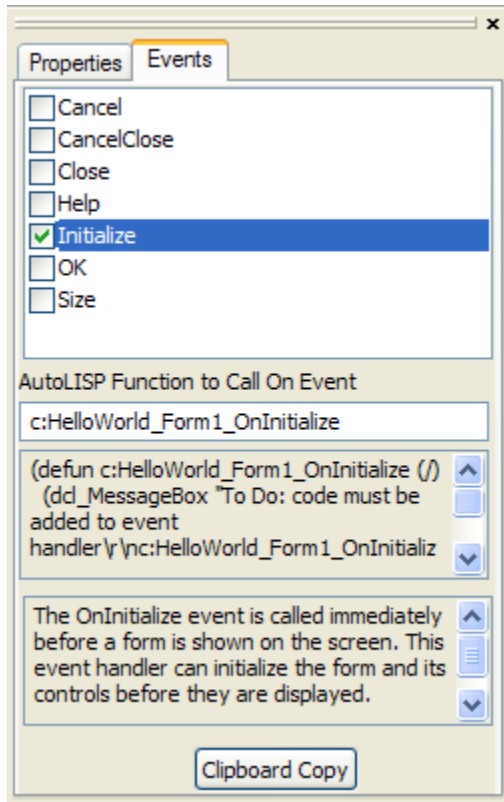
If AutoCAD isn’t already running, go ahead and launch it now and open the Visual LISP editor (type VLIDE at the command line). Create a new LISP file and save it as “HelloWorld.lsp” in the same location as the project file (in the AutoCAD search path). Type out or copy and paste the code below into the new file.

```
; Ensure the appropriate OpenDCL ARX file is loaded
(command "OPENDCL")

(defun c:Hello ()
  ; call the method to load the HelloWorld.odcl file.
  (dcl_Project_Load "HelloWorld" T)
  ; call the method to show the Hello World dialog box example
  (dcl_Form_Show HelloWorld_Form1)
  (princ)
)
```

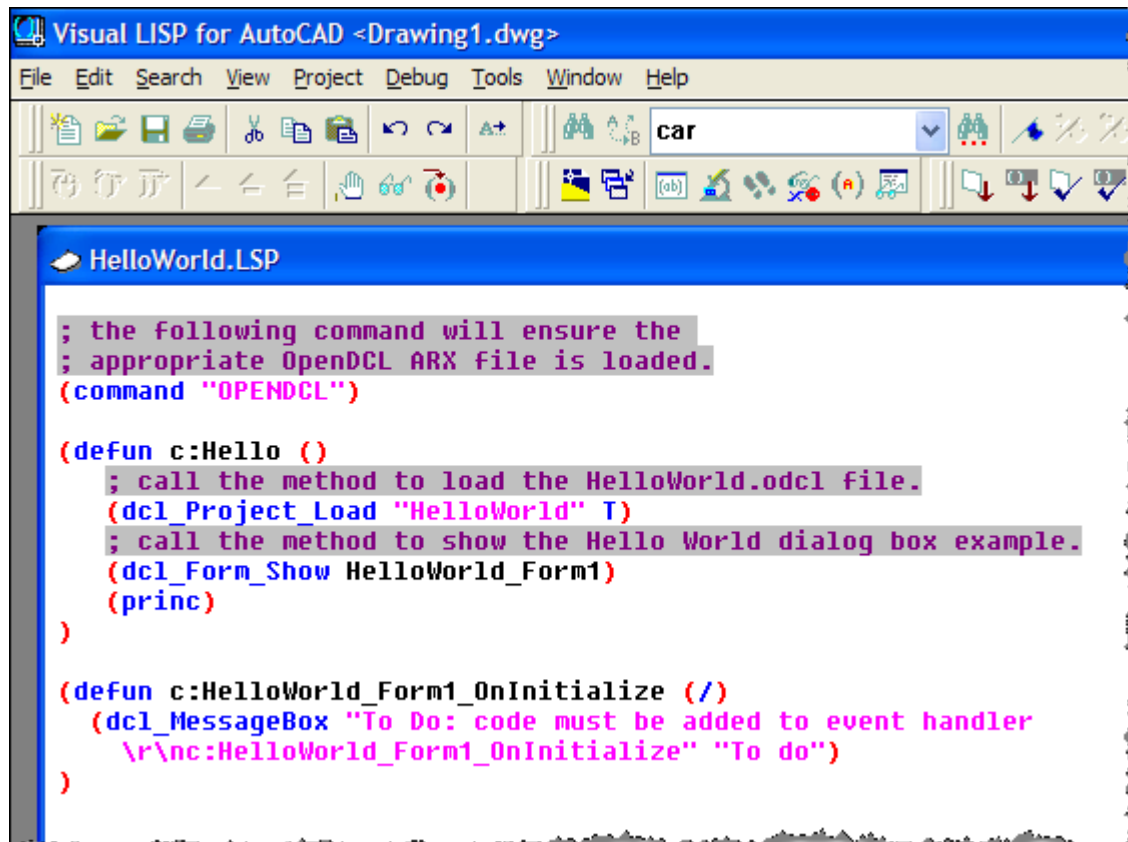
Now save the LSP file and load it into AutoCAD by hitting **CTRL+ALT+E**. You can now run your command by typing **HELLO** at the command line. The new dialog should pop open and look the same as it did in the ODCL Studio. Note that nothing happens when you try to click on the OK button. This is because the OK button hasn’t been assigned an Event yet. Also, note that while this dialog is open, you can’t interact with AutoCAD. This is called a “Modal Dialog”, as that’s the type of dialog we chose when we created it. To close the dialog, click the red X in the upper right corner.


Now let's put the "Hello" into our "Hello World" project. Now that we have our form designed, we can add the code to actually make it do something. To accomplish this, we're going to use Events. Go back to the Studio and select the form without selecting any of the controls.

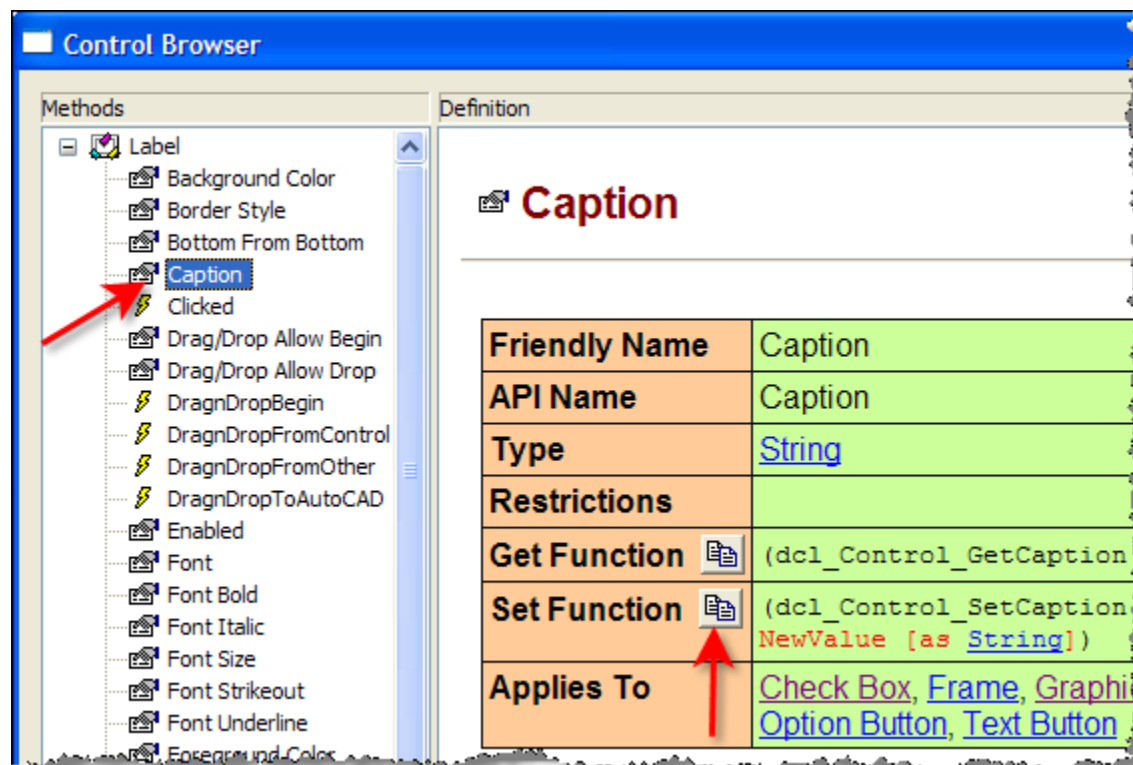


Next to the Properties tab is the Events tab. In it you can select the types of events you'd like to associate with a control or form. Each event has pre-written template code and a brief description on it at the bottom of the window. Select the "Events" tab, then put a check next to the "Initialize" event. This will display the pre-written **defun** code for the **OnInitialize** function. Now hit the "Clipboard Copy" button and the code will be written to your clipboard. The OnInitialize function is designed to run whenever the dialog is about to be shown. It can be used to perform any number of tasks and in the next step we'll modify it to include something useful (the default is a simple message box).

Go back to the Visual LISP editor [VLIDE] and paste the code into your file, it should look like this:



Now back in the Studio, double-click on **Label1**. A new window called the “Control Browser” will open with a list of functions that can be used with the Label control. Near the top in the left hand list, click on the “**Caption**” property. The right hand window shows details about use of the “Caption” property are displayed. Click on the “Copy to Clipboard”  button next to “**Set Function**”.



Save the ODCL file and switch back over to VLIDE. Update the `c:HelloWorld_Form1_OnInitialize` function with the new “SetCaption” code. e.g. delete the `dcl_MessageBox...` line and replace it with the code on your clipboard. You’ll also need to change the argument for the “SetCaption” code. Replace “**NewValue [as String]**” with “**Hello World**” (don’t forget the double quotes, as this argument expects a string value). The completed function should appear as below:

```
(defun c:HelloWorld_Form1_OnInitialize (/)
  (dcl_Control_SetCaption HelloWorld_Form1_Label1 "Hello World!")
)
```

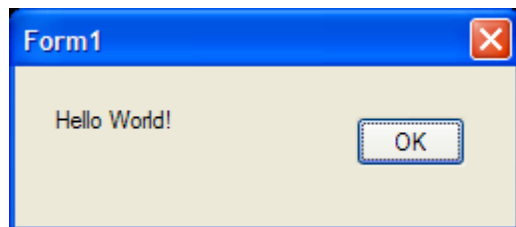
Save your LSP file and load it into AutoCAD again, run the **HELLO** command and note the Label now says “Hello World!” instead of “Label1”. This is because the OnInitialize event fires before the form is shown and changes the labels “Caption” property. Instead of hard coding “Hello World” into our ODCL project at design-time, we’ve given our code the ability to change the label on-the-fly or at run-time. Note the OK button still doesn’t do anything, so let’s fix that now.

This time around instead of copying the Event code to our clipboard, we're going to save ourselves a few clicks. Back in the Studio under the Tools pull-down menu, select "Write Events to Lisp File". Now select the OK button, and switch to the Events tab, if it's not already selected. Check the "Clicked" event, and then hit the "Add to File" button. A window will open asking you to "Specify Project AutoLISP File". Choose the "HelloWorld.lsp" file, and then save the ODCL project. When you switch back to VLIDE, you'll get a Message Box saying "Revert buffer to disk contents of file" YES/NO, Choose Yes and you'll notice the `c:HelloWorld_Form1_TextButton1_OnClicked` Event code appear at the bottom of your LSP file. **WARNING:** if you didn't previously save your work in the VLIDE, it will be overwritten when you reload or revert to the disk contents, so remember to save often.

In the Studio, double-click on a blank area of the form to open the "Control Browser". Click on the "Close" Method (the one with the green box, not the Close Event with the lightning bolt). Click the "Copy to Clipboard" button next to **Syntax**. Now in VLIDE paste it into the OnClicked Event so it looks as follows:

```
(defun c:HelloWorld_Form1_TextButton1_OnClicked (/)
  (dcl_Form_Close HelloWorld_Form1)
)
```

Save your LSP again & reload it into AutoCAD and run the **HELLO** command one more time. Now when you click the OK button the form will be closed.



Let's see if we can make the "Hello World" project a little more interactive. Add a "Text Box" control under the Label, so it looks like this:



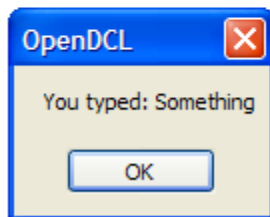
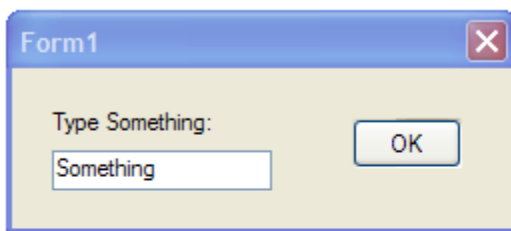
Remove "TextBox1" from the "Text" property of the newly added control. Double-click on the Text Box and select the "Text" Property on the left column. Now copy the **Get Function** code to the clipboard. Save the project at switch back to VLIDE.

Paste the GetText code into the OnClicked Event code block and update the rest of the code block as follows:

```
(defun c:HelloWorld_Form1_TextButton1_OnClicked (/)
  (setq sText (dcl_Control_GetText HelloWorld_Form1_TextBox1))
  (dcl_messagebox (strcat "You typed: " sText))
  (dcl_Form_Close HelloWorld_Form1)
)
```

Also, change the "Hello World!" caption to "Type Something:".

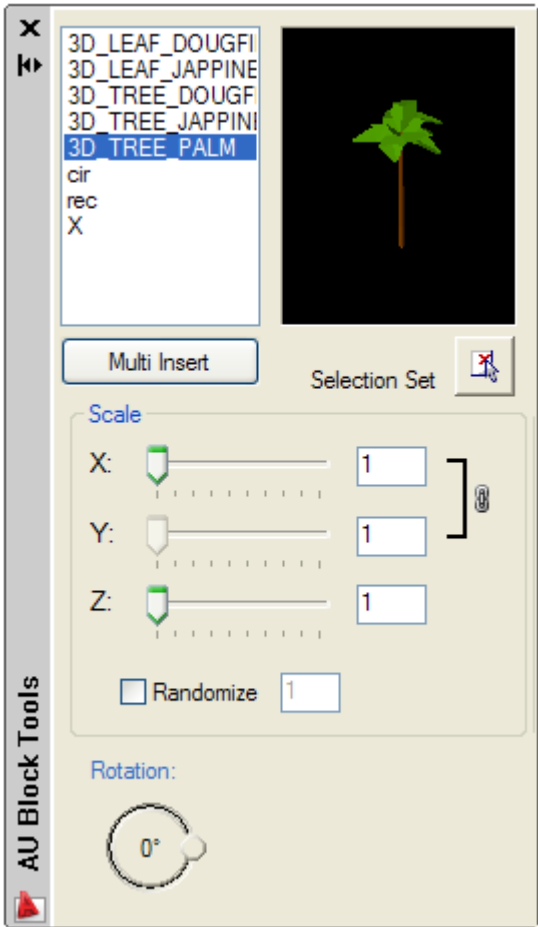
```
(defun c:HelloWorld_Form1_OnInitialize (/)
  (dcl_Control_SetCaption HelloWorld_Form1_Label1 "Type Something:")
)
```



Save, reload, and run **HELLO** to see the new and improved Hello World. Type something into the Text Box and hit OK. A MessageBox will pop up showing you what you typed. This demonstrates how to retrieve the property of a control and use it somewhere else.

This concludes the Basic Level introduction to OpenDCL. If this all made sense, you are ready to move on to the Intermediate Level content. If not, maybe you should start over again and give it another try. If something didn't work properly you probably didn't save the ODCL file or the LSP file before you tried running the command. Remember, you can always visit the Forums at OpenDCL.com to ask for help. With close to 200 register members, you'll probably get your answer sooner than you think.


Intermediate Level 2: Creating a real world Palette project.

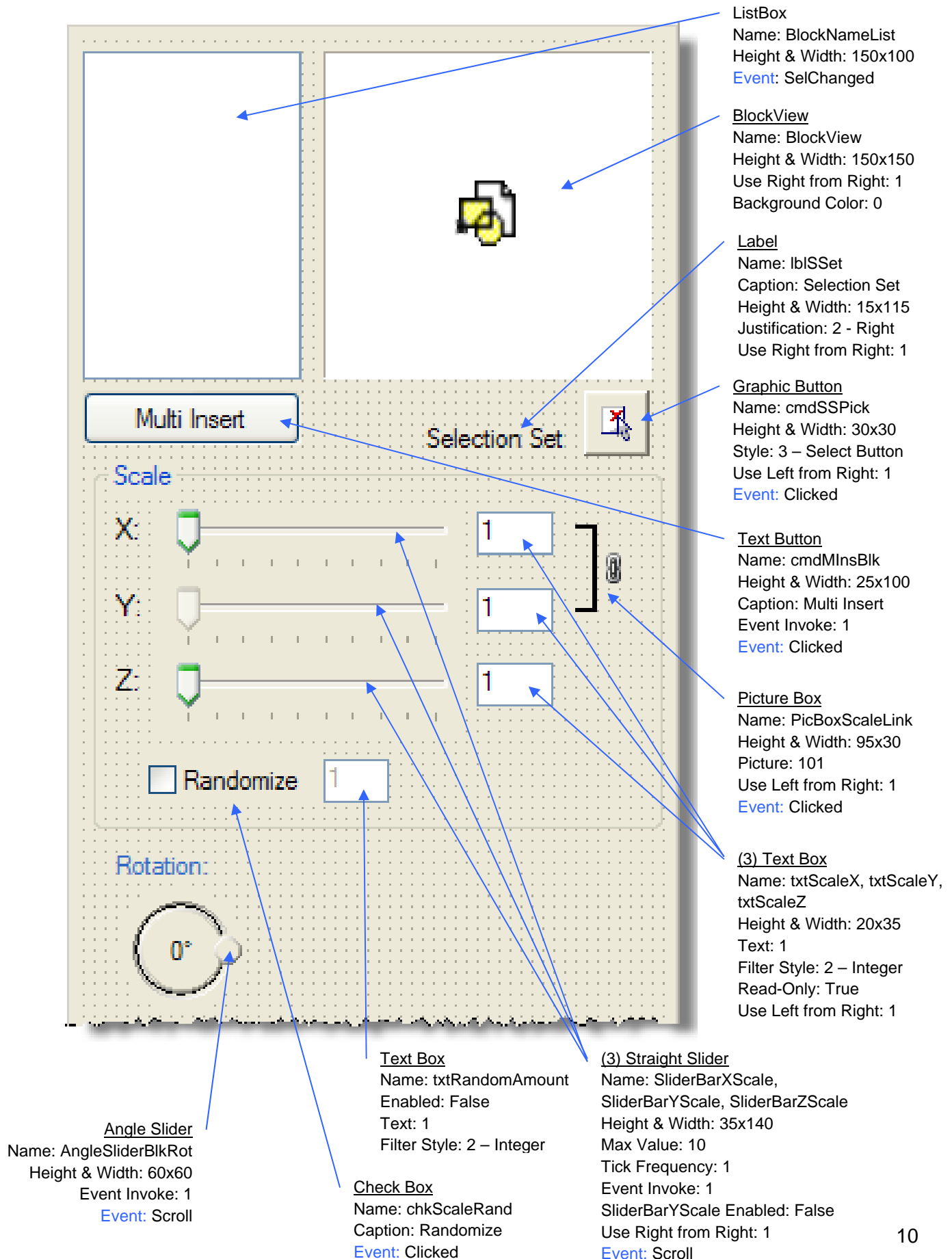


Let’s try to create a more advanced project that we can use in the real world. This project will allow us to preview blocks that are defined in our drawing, insert them, rotate, and scale them with a random factor. The image on the left shows the finished project.

Start a new project in ODCL Studio and save it as “AUBlockTool.odcl”. For this project we’ll use a Palette form, which is added from the toolbar or pull-down menu. Change the following properties for the Palette form in the Properties panel:

| | |
|-----------------|----------------|
| Name: | PalBlkTool |
| Title Bar Text: | AU Block Tools |
| Width: | 275 |
| Keep Focus: | True |

Now that you have the palette form created, see the next page for all the controls to add. Try to position the controls as shown on the image. Any properties that are noted should be changed in the Properties panel, most importantly the Name. You’ll notice some of the controls have Events associated with them; hold off on adding them until the next step. When you get to the **PictureBoxScaleLink** Picture Box, you’ll need to have some images pre-loaded into your project. These images can be found in the zip file that you downloaded this tutorial, under the sub-folder “Level 2\Start”. If you need to download the tutorial with all its supporting files again you can find it [HERE](#). Unzip the files into your working support path. Now hit the “Picture Folder”  on the main ODCL toolbar. Click the “add” button three times to load the images in their given order (100= lock-1.bmp, 101= lock-2.bmp, 102= lock-3.bmp).



You'll also need to add a few other controls that aren't noted above. They are the "Scale" frame, "X:", "Y:", "Z:" labels & "Rotation:" label.

You should now have the form complete with all the controls on it. Remember to save your ODCL project before moving on. Now let's test our form to ensure everything is looking good. Startup AutoCAD and open an existing drawing that has some blocks already defined in it, or create some test blocks in it. Open VLIDE and open "AUBlockTool_Start.lsp". Again, it can be found in the ZIP file or downloaded from [HERE](#). This file has a number of pre-written utility functions that this exercise will rely upon. Copy the following code to the top of the LSP file:

```
(defun C:AUBT ( / )
  (command "OPENDCL")
  (dcl_Project_Load "AUBlockTool" T)
  (dcl_Form_Show AUBlockTool_PalBlkTool)
  (princ)
)
```

Save the LSP file as "AUBlockTool.lsp" and load it into AutoCAD. At the command line type "AUBT" to run your newly created command. The palette should open and look very similar to above; however none of the controls will work since we haven't added any events to them yet. Let's do that now. Go back in the Studio and click on the form without selecting any of the controls. Now switch the Properties tab over to the Events tab and check the "Initialize" event. Hit the "Clipboard Copy" button at the bottom of the window, then save the ODCL project. Flip back over to VLIDE and paste the "OnInitialize" function into your code. Replace the default (dcl_MessageBox) code with the following:

```
(defun c:AUBlockTool_PalBlkTool_OnInitialize ( / )
  (dcl_ListBox_Clear AUBlockTool_PalBlkTool_BlockNameList)
  (dcl_LISTBOX_ADDLIST AUBlockTool_PalBlkTool_BlockNameList (GetBlockNames))
)
```

This will clear the ListBox then call the (GetBlockNames) function and populate the list with the names of all the blocks in the drawing.

It's time to test the LSP again. But first, let's save the LSP, then reload and run AUBT. The palette will open again, but this time it should show a list of all the blocks defined in the drawing (if the list is empty you didn't open a DWG with blocks predefined in it).

Let's add some more events to the code, click on the ListBox in the top left corner of the palette, then switch to the Events tab and Check the "SelChanged" event. Hit the "Clipboard Copy" button, save, and flip back over to VLIDE and paste the "OnSelChanged" function into your code. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_BlockNameList_OnSelChanged (nSelection
sSelText /)
  (dcl_BlockView_DisplayBlock AUBlockTool_PalBlkTool_BlockView sSelText)
)
```

When the selection of the ListBox changes, the BlockView will display the selected block.

Select the **cmdSSPick** Graphic Button, check its “Clicked” event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_cmdSSPick_OnClicked (/)
  (setq *blocks* (ssget '((0 . "INSERT"))))
  (UpdateSStext)
)
```

*When this button is clicked, the selected blocks will be added to the ***blocks*** global variable and the label next to it will be update by the (UpdateSStext) utility function.*

Select the **cmdMInsBlk** Text Button, check its “Clicked” event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_cmdMInsBlk_OnClicked ( / CSel BlkName X Y Z
rot pt)
  (setq CSel (dcl_ListBox_GetCurSel AUBlockTool_PalBlkTool_BlockNameList))
  (setq BlkName (dcl_ListBox_GetText AUBlockTool_PalBlkTool_BlockNameList
CSel))
  (setq X (dcl_Control_GetValue AUBlockTool_PalBlkTool_SliderBarXScale))
  (setq Y (dcl_Control_GetValue AUBlockTool_PalBlkTool_SliderBarYScale))
  (setq Z (dcl_Control_GetValue AUBlockTool_PalBlkTool_SliderBarZScale))
  (setq rot (dcl_Control_GetValue AUBlockTool_PalBlkTool_AngleSliderBlkRot))

  (setq *blocks* (ssadd))
  (while (setq pt (getpoint "Select base point\\Right-click to exit: "))
    (command ".insert" BlkName "X" X "Y" Y "Z" Z "R" rot pt)
    (ssadd (entlast) *blocks*)
    (UpdateSStext)
  )
  (princ)
)
```

This will get the current selection out of the ListBox, then get the text of the current selection and set it to the BlkName variable. It will get the values of the X,Y, & Z sliders and the rotation from the Angle Slider. It will start a selection set, while it inserts the block multiple times, until the user exits. As the blocks are being inserted they will use the saved values from the X,Y, & Z scales and the rotation, along with adding them to the selection set and updating the label.

Select the **SliderBarXScale** Straight Slider, check its “Scroll” event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_SliderBarXScale_OnScroll (nValue /)
  (dcl_Control_SetText AUBlockTool_PalBlkTool_txtScaleX (itoa nValue))
  (DoScale 41 nValue)
  (UpdateLinks (itoa nValue))
  (princ)
)
```

As the slider bar is scrolled, this code will update the value in the Text Box next to it, scale the block in the X direction with the (DoScale) function, and scale the block in the Y and Z directions if they’re linked with the (UpdateLinks) function.

Select the **SliderBarYScale** Straight Slider, check its “Scroll” event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_SliderBarYScale_OnScroll (nValue /)
  (dcl_Control_SetText AUBlockTool_PalBlkTool_txtScaleY (itoa nValue))
  (DoScale 42 nValue)
  (princ)
)
```

This will do the same as the X version above, except it scales in the Y direction and it doesn't check if the scales are linked, since this was done already.

Do the same thing for **SliderBarZScale**, but change the “Y” to “Z” in two places & change the **42** to **43**.

Now is a good time for another test. Remember to save both the ODCL project and LSP file, then reload it into AutoCAD, and run AUBT. Now when you click on the ListBox the BlockView should update to preview the selected block. When you hit the “Multi Insert” button, you should be prompted to insert the block until you exit. Also, when you scroll any of the Slider bars, the block should scale in that direction.

Now select the **PicBoxScaleLink** Picture Box, check its “Clicked” event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_PicBoxScaleLink_OnClicked ( / Pic)
  (setq Pic (dcl_Control_GetPicture AUBlockTool_PalBlkTool_PicBoxScaleLink))
  (cond
    ((= Pic 100);_ not linked
      (dcl_Control_SetPicture AUBlockTool_PalBlkTool_PicBoxScaleLink 101)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_SliderBarYScale nil)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtScaleY nil)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_SliderBarZScale T)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtScaleZ T)
    )
    ((= Pic 101);_ XY linked
      (dcl_Control_SetPicture AUBlockTool_PalBlkTool_PicBoxScaleLink 102)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_SliderBarYScale nil)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtScaleY nil)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_SliderBarZScale nil)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtScaleZ nil)
    )
    ((= Pic 102);_ XYZ linked
      (dcl_Control_SetPicture AUBlockTool_PalBlkTool_PicBoxScaleLink 100)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_SliderBarYScale T)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtScaleY T)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_SliderBarZScale T)
      (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtScaleZ T)
    )
  );_ cond
```

```
(princ)
)
```

First this code gets the current picture in the Picture Box, and then it goes into a 3 way toggle. If Pic is 100, the scales aren't linked, so change the picture to 101, making X & Y linked. Also, disable the Y SliderBar & TextBox and Enable the Z SliderBar & TextBox. This is because when X & Y are linked, the user should only be able to scroll X. Y will be scrolled automatically with the (UpdateLinks) function from above. If Pic is 101 or 102, the code is very similar; it's just bumping the picture to the next one inline and enabling or disabling the other controls.

Now select the **chkScaleRand** Check Box, check its "Clicked" event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_chkScaleRand_OnClicked (nValue /)
  (if (= nValue 0)
    (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtRandomAmount nil)
    (dcl_Control_SetEnabled AUBlockTool_PalBlkTool_txtRandomAmount T)
  )
)
```

This will enable or disable the Text Box next to it, so the user can enter a different random amount. Note, a Check Box's value can be: 0 = Unchecked, 1 = Checked, or 2 = Indeterminate

Now select the **AngleSliderBlkRot** Angle Slider, check its "Scroll" event, copy the code, save and paste it back into VLIDE. Update the function as follows:

```
(defun c:AUBlockTool_PalBlkTool_AngleSliderBlkRot_OnScroll (nValue / cnt ed)
  (setq cnt 0)
  (if *blocks*
    (progn
      (while (< cnt (sslength *blocks*))
        (setq ed (entget (ssname *blocks* cnt)))
        (entmod (subst (cons 50 (dior nValue)) (assoc 50 ed) ed))
        (setq cnt (1+ cnt))
      );_ while
    );_ progn
    (princ "\nNothing in the selection set.")
  );_ if
  (princ)
)
```

This will rotate all the blocks in the ***blocks*** global variable.

Your project should now be complete. Save the ODCL and LSP, and run AUBT again. You should be able to click on the Picture Box and have it toggle between the 3 pictures. Clicking on the Randomize Check Box, should enable the Text Box next to it and when you scroll a Slider Bar the blocks should scale a little differently, based on the random amount. The Rotation Angle Slider should now work also.

We are now basically done with this example project. We just have one last thing to clean up. Go back to the top of your LSP file and remove the reload flag "T" from the (dcl_Project_Load) method. So it should now look like this:

```
(dcl_Project_Load "AUBlockTool")
```

So why are we doing this? When you're creating and debugging your ODCL project is a good idea to keep the reload flag on, because you're making a lot of changes and you'll want to see them applied right away. When the reload flag is set to T, it forces ODCL to reread the ODCL project file from disk every time it's called. So when you make a change to the ODCL file and save it, then run your code again, it will reread the ODCL file and display whatever changes you've made to the project. On the other hand, all this reading from disk could slow things down on larger projects. So when you're done testing your project and getting ready to deploy it, you should remove the reload flag. This will speed things up, since the project is now being read from memory (RAM). This also has another benefit; the controls will remember their last values. For example, if the user inserts some blocks, scaled them in X=5 and Y=7 and a rotation of 30, then closed the form and reopened it, the controls will still be at X=5 and Y=7 and a rotation of 30. Unlike when we had the reload flag set to T, where the controls would always start up at X=1 and Y=1 and a rotation of 0.

A few side notes about this example, because of time constraints and for readability, a few events have been left out of this example. Since this example is using a Palette form, the user could open or switch documents while it's active. You could add the "DocActivated" event to the Palette, so when the user switches documents, it will repopulate the list of blocks on the form. However this Event won't fire when a new documents is created. So a better solution would be to add some code to your "Acaddoc.lsp" or something similar, to manage this for you. See the "DocActivated" Event in the ODCL help file, for more information in this area. Another consideration, is when the user closes all the documents, but keeps AutoCAD open. Luckily, there's an event for this also. The "EnteringNoDocState" Event will fire just before the last drawing is closed. You should take this opportunity to close your modeless forms, since they won't work in this state.

This concludes our example project. The AU Block Tool should be fully functional on your workstation. Now you'll need to decide how you want to push out (or deploy) your project to others. This will be cover in the next section.

Advanced Level 3: Deploying your project.

When you've finished creating and testing your ODCL project, you'll want to push it out to others so they can use it. If you're a CAD Manager or someone how's developing a tool for in-house use, you're almost done. However, if you're planning to publish your software out to the public, you'll want to take a few extra steps to secure your source code.

Option 1 (in-house use):

The easiest way to give your office access to your project is to just copy your ODCL and LSP file into a network location that everyone can read and is in an AutoCAD support path. Even better would be a folder that you have read/write access to, but the end users only have read access to. The end users will also need the "runtime" installed on each of their workstations. The "runtime" is an ARX module that is loaded into AutoCAD. It will run in AutoCAD 2004 to AutoCAD 2009 x64 bit. It is what manages everything under the hood, by reading the .ODCL file and displaying the forms in AutoCAD. Again you have some options here. You can go back to the OpenDCL.com download page & grab the "runtime" installer (the MSI file) and install it on every PC in your office. However, if you work for a large company or have multiple offices, you'll probably want to add the installer to your Windows logon script or whatever other means you use to push out software updates. Or you can copy the "Runtime" to a network location and load it inside AutoCAD programmatically. This may be a little easier to manage since you only have to deal with one central location, but on the other hand you also have to manage loading the ARX programmatically for multiple versions of AutoCAD. When you installed the ODCL Studio it also installed the "runtime". It's typically located under: *"C:\Program Files\Common Files\OpenDCL"*. Copy the contents of this folder, along with the sub-folder, to your central network location that all the workstations have access to. The ENU sub-folder contains the runtime resources for the English language pack. If you installed a different version of OpenDCL, you may have a different sub-folder. Now have a look at the "ManualLoading.lsp" Sample file provided with ODCL Studio. It's typically installed under: *"C:\Program Files\OpenDCL Studio\ENU\Samples"*. You can copy this file into your "Acad.lsp" file, so it will load the "runtime" every time AutoCAD starts up. That's pretty much it. Your office should be up and running with OpenDCL and you can start pushing out stunning UI's to your colleagues.

Option 2 (deploying for the masses):

To secure your source from prying eyes, you'll want to encrypt it into a VLX file. Of course this isn't 100% secure, as almost everything is hackable, but it will keep out the majority of the snoopers. Fortunately OpenDCL also gives us a means to embed our ODCL project file into our LSP source code so it will also be protected. Follow the steps below to securely embed your ODCL project into a VLX file.

Open your ODCL project in the Studio. Go to File-> Save As and change the file extension to .LSP. A good naming convention to follow is FileName.odcl.lsp, this way you know the file was generated from an ODCL file and not a typical LSP file. The Studio will encode your project to Base64 and wrapping in LSP. You can now open the intermediate .odcl.lsp file in any text editor and copy it into your LSP source file. Add a (setq) around the copied encoded string

and replace the `(dcl_Project_Load)` with `(dcl_project_import)`. It should look something like this for our AU Block Tool example:

```
(defun C:AUBT ( / )
  (command "OPENDCL")
  (setq project

'("YWt6A+2rAADje5J3BuKT5SUSaplrubH8/59gNmIW3chwX9BPCPOTpj5WcWZWXLi/uMNMlXeH8J
W7"

"aGmpnnnybszkZBSMfiVXPG8Ub3Rb37bD//xef9drLKtpYFSbe3tfemZyK9XU1JzgDQiT50fF4I8G
"

"+KO7QP2okueHmEmhAqGB4pHEDXnBYvqWGZhkhmDqPmak3nAknsWGAUWkX36GhUp+/ewzO32BISL9
"

"Nyg8X6o+Q5o9b5rco3ZzznzBw/uZdR9nANW/TbsGQGmTJDIr8WgjlRnVeHeClj9cqLwMMM9ikVEc
"

"MH8AETpEb5WuXJx0gmAHIE9MdI5Rz2SOEl3YUL4Kl07t3uqVZ4b/ex76Hmw6ccJK3WLhYphpSyGs
"

"hnWyYTRtMxIS/8rfLTpNxIZBQJAd4PMxHnQmcJ0tH7FJralcreZizI2AFiCZisFpiGH6olmSSRLR
"

"S/kUUb0sAWe9Ucl58sGkrPzvG8wguBG48ccYRxUCWy2m/byAIqZjSIWzRAZlnNUXRbpVgJPOZyQf
"

"XQDKH1EAVjtNChy7ZEFpMURckgMI5VShpvpJkT8ud7yJ3yA2dlL46yxyzbNiwPUxo/cxYsSn5lPg
"

"is/OUXaFXR+BKKafH0MPp2VTvKBHIlnfAe3ApLqLYXeFefyhXZLTvAC3GGkE0d3vtVQsAesGWHIV
"
  "OQ/1zOvTEhtVJae0KMjM6+NeSEZg17Vwji7IzLlJhYHgr3l9")
  )
  (dcl_project_import project nil nil)
  (dcl_Form_Show AUBlockTool_PalBlkTool)
  (princ)
)
```

Note, don't copy the encoded string above, as it's been truncated for readability and won't work.

Now that you've copied the encoded string into your LSP file, you can delete the intermediate .odcl.lsp file if you wish. Now let's build the VLX file. In the VLIDE, save your LSP then go to File->Make Application->New Application Wizard... On the first screen select "Simple", then hit Next. Browse to a folder to save you application. You can save it anywhere you like. Then give you application a name. We'll use "AUBT" for this example; therefore "AUBT.VLX" will be created. Hit Next. On the third screen, hit "Add" and select your LSP source file; "AUBlockTool.lsp" in this case. Hit next again, and then on the last screen ensure "Build Application" is checked and hit the Finish button. Your application has now been compiled into a VLX file and can be distributed to the public.

See the Distribution Sample project for more information on this topic, located in: *"C:\Program Files\OpenDCL Studio\ENU\Samples\DistSampleReadMe.txt"*

We now have a few options for distributing our application to the public.

Option 1 (quick and dirty):

Zip up your VLX file along with the OpenDCL "runtime" installer (.MSI file) and some instructions like:

1. Run the MSI file to install the OpenDCL Runtime.
2. Copy AUBT.VLX into one of your AutoCAD support paths.
3. Append your "acad.lsp" file to load the application with `(load "aubt.vlx")`

Option 2 (use an installer):

Write your own installer. Sorry, don't ask me how, I've never done it. However you can go back to the OpenDCL.com download page and download the "runtime.msm" file. This is known as a merge module, and it can be merged into your own custom installer. This way the OpenDCL runtime will be installed at the same time as your application.

Bonus Example:

You can also add ActiveX controls to your OpenDCL projects. Here's a form with the Adobe PDF Reader and an Apple QuickTime movie embedded in it.



Appendix:

- ODCL – Abbreviation for OpenDCL or file extension for an OpenDCL project.
- LSP – Abbreviation for AutoLISP or file extension for an AutoLISP file.
- Form – Synonym for dialog box.
- Control – An item on a form (button, list box, combo boxes).
- Event – Happens when the user interacts with a control (Clicked, Scrolled).
- Method – A function to manipulate forms & controls.
- Property – An attribute or characteristic of a control (Visible, Color).
- VLIDE – Visual Lisp Integrated Design Environment.

This tutorial was written in November 2008 using OpenDCL 5 Beta 3, build number: 5.0.1.3 and AutoCAD2009 sp2. Newer versions may contain other features or work differently then described in this tutorial.