ObjectDCL Ver2 Beginners Manual

Welcome!

Welcome and thank you for choosing ObjectDCL. This manual is designed to teach you the basics of ObjectDCL and to give you a general understanding of how to program in an event driven programming environment. This document is organized so that its section headings are navigable by the sidebar using the MS Word menu option in View/Online Layout.

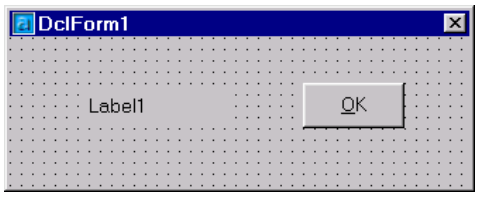
Launching the ObjectDCL Editor

An ObjectDCL shortcut was created during the installation routine. Use this shortcut to launch the ObjectDCL Editor program. You should see a Toolbar below the Windows menu and four dockable windows arranged two on the left and two on the right side of the screen. The View menu has check marks visible that control the visibility of these windows. Double clicking on the bar at the top of each window toggles between a minimal window and docked window. These windows can also be dragged into the central workspace and resized. Clicking on them again will return them to their previous docked position. These windows can be located wherever you desire.

Creating Your First Dialog Box, Step by Step, "Hello World"

To create your first example dialog box we are going to layout a few simple instructions for you to follow. You are first going to create a project in which you will design you first dialog

An ObjectDCL File



In the ObjectDCL editor, choose create a new project. Under the 'Project' pull down menu, click the 'Add Modal Form' selection. A new dialog box will appear. For our example resize the dialog box using the corner grips and draw one Label control, and one TextButton control by selected them from the ToolBox on the right side of the Editor. You could change the "Caption" property of the Label to say "Hello World" but leave it as is. We are going to demonstrate how to change a control's property when the dialog box is starting up, but has not been shown yet. Change the caption property of the Button to say "OK", so the dialog should look something like this:

An AutoLisp File

Create a blank AutoLisp file, for our example we will call it HelloWorld.lsp. At this point all we need are two functions to be added. Cut and paste the code below into your new file. If the symbol T is included with the load project call, the project reloads over the one currently loaded into AutoCAD.

This is useful to see changes made to the dialog form saved to the .odc file. Once the design of the form is complete, remove the T prevent the project form overwriting the currently loaded forms. Odcl_LoadProject returns true when loading and nil if forms are already loaded.

Important Note:

Programmers developing on a network drive should be aware that problems with loading the most current save have been reported. See Project section.

```
;; this method is called to ensure that the ObjectDCL.arx file is loaded into AutoCAD.
;; Please ensure that the ObjectDCL.arx file is in the AutoCAD search path

(defun ObjectDCL_LoadArx ()
  (if (not (member "objectdcl.arx" (arx)))
    (arxload "objectdcl.arx" "ObjectDCL.arx not found.")
  )
)

(defun c:Hello ()
  ;; ensure the ObjectDCL.arx file is loaded
  (ObjectDCL_LoadArx)

  ;; call the method to load the Hello.odc file.
  ;;the T forces project to reload each time to make it easier to see changes
  (Odcl_LoadProject "Hello.odc" T)

  ;; call the method to show the Hellow world dialog box example
  (Odcl_Form_Show "Hello" "DclForm1")

  (princ)
)
```

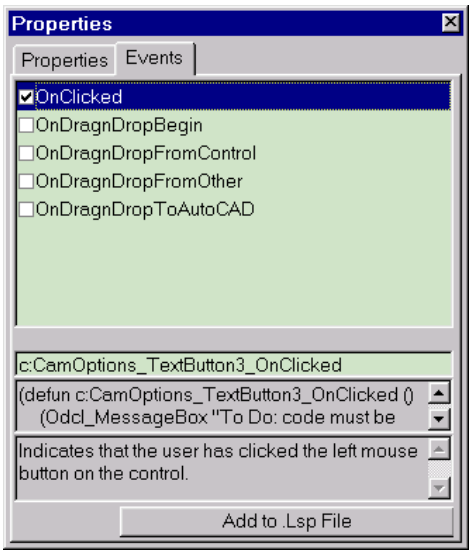
Setting Up the OnInitialize Event to Set the Label to Display "Hello World"

To set a control at startup of the dialog time, follow these simple steps. Select the dialog box, right click on it and select 'Events' or double click on the surface of the dialog box, either will display this property wizard dialog box:

Select the OnInitialize event from the list on the left. Click on the 'Add to .Lsp File' on the right. You will be prompted to select which AutoLisp file to write to, select the Hello.lsp file we just created. The function defun that is show in the middle control on the right will be written out to your Hello.lsp file.

Open up the lisp file and modify the new defun function to look as shown below:

```
(defun c:DclForm1_ DclForm1_OnInitialize ()
    ; call the method to set the textual display of the
Label to state "Hello World"
    (Odcl_Control_SetProperty "Hello" "DclForm1"
"Label1" "Caption" "Hello World")
)
```



Setting Up an Event to Close the Dialog Box

To allow the user to close the dialog box when the OK button is pressed, follow these simple steps: Click on

the "Events" tab in the Properties docked box on the right.

Next select the "Clicked" item from the ListBox on the left and press the ‘Add to .Lsp file’ in the lower right corner. The function defun that is show in the middle control on the right will be written out to your Hello.lsp file.

Next we need to modify the event so it will close the dialog box when the "OK" button is clicked. Go into the hello.lsp file with an editor and modify the defun to include the code below.

```
(defun c:DclForm1_TextButton1_Clicked ()
    ; call the method to close the Hello World dialog box example.
    (Odcl_Form_Close "Hello" "DclForm1")
)
```

Your program will never need to call this method directly, ObjectDCL will call it for you when the user press the OK button. The C: in the defun may seem like it should be declared without the C: because its a function to be called by another program and not by a user, but ObjectARX requires the C: in many cases to be able to actually activate the function. By requiring this, we have taken this confusing step out for you, so you don’t have to keep track of what events from what dialog box types require what type of declaration of defun’s.

In Standard DCL you would normally use the ActionTile calls to set variables and close commands to close the dialog box, with ObjectDCL we have setup defun’s that it calls instead. This allows more flexible and powerful steps, procedures and checking to take place when a user takes an action. Using this event defun system, you can easily setup any action to check the user’s input against certain conditions and rules. You can also call ObjectDCL supplied to modify the look or contents of ObjectDCL controls.

The Final Step

Save the Hello.odc file to the AutoCAD support directory or other support searchable directory of your choice. Load the Hello.lsp file and type Hello at the command line. The dialog box will now appear. Press the OK button and the dialog box will then close.

Modifying your Dialog

Now that you see the result of your work you may wish to change some of the properties of your dialog and re-display it inside AutoCAD. The code that initialized your dialog loaded it into memory.

```
; call the method to load the Hello.odc file.
  (Odcl_LoadProject "Hello")
```

If you change your dialog and save it in the ObjectDCL editor, the dialog will not update when launched by AutoCAD because it already reside in memory. To force AutoCAD to reload your dialog add the symbol T to the load method as follows:

```
; call the method to load the Hello.odc file.
  (Odcl_LoadProject "Hello" T)
```

Go ahead and play with the values in the properties and watch how the dialog changes its appearance in AutoCAD.

Remove the T once your code is finished and ready to publish, that way your dialog will stay and memory and display faster.

Keeping variables local

Please refer to event handling under the ObjectDCL Programming Tips in the ObjectDCL Project section.

What do ObjectDCL dialogs look like?

Described below is how to see ObjectDCL supplied example dialogs in AutoCAD for the first time. ObjectDCL does supply other simpler examples on the web for download. You can find a list of these examples based on questions asked by other AutoLisp programmers at: www.objectdcl.com/knowledgebase/knowledgebase.html

To run the samples described below, add both the directories C:\Program Files\3rd Day Software\ObjectDCL and C:\Program Files\3rd Day Software\ObjectDCL\Examples to AutoCAD's search path.

ObjectDCL Demo Projects:

The example directory contains some sample dialogs ready to run in AutoCAD. Load lisp file "demo.lsp" into AutoCAD to create the following commands that can be executed on the AutoCAD command line:

Demo

Launches a dialog showing tabs, block viewer, sliders, combo boxes, html link, progress bar, and various buttons.

Demo2

This example will activate a modal dialog box, which is designed to demonstrate how to programmatically manage the PictureBox, Slider controls, and placement of controls. The PictureBox tab demonstrates how to call most of the functions that would be used to create your own custom controls.

DemoDockable

When the dockable tree is launched it docks to the left side of the screen. If the use drags it out on to the screen it does not re-dock.

DemoModless

Shows a dialog that can be resized.

Events

This example will activate a simple dialog box that demonstrates how to program a simple ObjectDCL program that will handle user driven events.

Html

Activates a modal dialog box with the HtmlControl. This control is a self contained Web Browser that uses the Internet Explorer 4.0 or better.

Viewdwg
Activates a modal dialog box with the DwgPreview control. This control will display the dwg bitmap thumbnail of any dwg file with the preview saved in it.

Folder
Opens a window from which a user can select a file folder.

Msgbox
Displays a dialog box that demonstrates the more flexible message box that includes icons and multiple button options to the user.

Tree
A nice example of a tree control.

Executing these commands in AutoCAD will show what kind of dialogs can be developed with ObjectDCL.

How Object DCL Works.
ObjectDCL is comprised of two separate programs, an Editor that provides a graphical development environment (GDE) where you create your dialog forms and the Arx file that works inside AutoCAD. The objectdcl.exe launches the Editor in a separate process outside AutoCAD and that is where one creates projects and its dialog forms. We use the term Editor when referring to ObjectDCL Editor.

This Editor can save your Project containing your dialog forms to a file with an “.odc” extension meaning “object dcl control” or an .ods extension meaning “object dcl secure” that can be used for distributing your work to others. A better usage would have been “s” for source and “c” for compiled like with AutoCAD menus. Unfortunately the “ods” came later and its too late to change now. Just think opposite to menus.

A project can contain multiple forms (dialog boxes) that can be displayed in AutoCAD by first loading it using the (Odcl_LoadProject "FileName") then the (Odcl_Form_Show "FileName" "DialogName") calls.

"Event" is a term used to describe when the user or system takes an action in the dialog that may require a response by the associated lisp program. For example using the above Hello World project, the "OnClicked" event was added so that the dialog box could be closed when the user presses the "OK" button. It is important to note that events can also be activated by calls to the control from AutoLisp as well.

When using ObjectDCL's "Add to .lsp file" button to add events handling defuns to the lisp file, the lisp statements that are written are really prototypes created for your convenience.

Distributing the dialog boxes requires that the .arx and .ods files (Not .odc files) distributed and included with your lisp and other files. Your license provides unlimited distribution of the runtime files. You may distribute the .odc files if you wish, but it is recommended to distribute the .ods files because they are guaranteed that not to be readable or modifiable by other users.

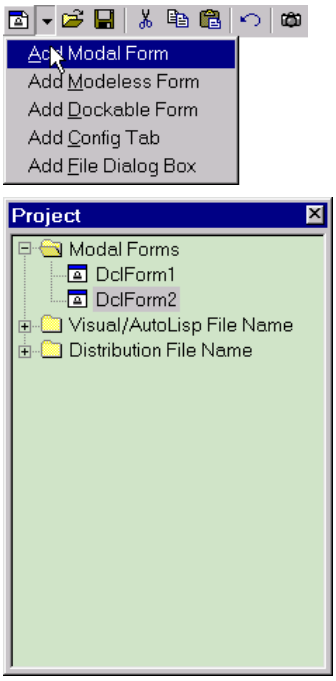
Create a Project

Start a new project using the File pull-down menu and save it to a directory that shall appear in AutoCAD's path. Project files always have an .odc extension. Under Project menu in the menu bar you will find from one to five types of forms that can be inserted in your project. See Dialog Forms section for a description of form types and version that makes them available.

Modal

A modal dialog box grabs complete focus of AutoCAD and will not allow the user to take any actions out side of it's boundaries, until it is closed. Modal dialogs disable AutoCAD command functions and require that the dialog box be closed before continuing. This means that when the (Odcl_Form_Show …) method is called the next line of code will not be activated until the dialog is closed (AutoCAD 2000 or 2002 only). Other dialog boxes can be called from an open modal dialog and displayed as its child(ren). Modal dialogs can stay open while AutoCAD commands are called from an AutoLisp defun using (Command …) only if the "EventInvoke" property of the control that called the event defun is set to "1 - Allow (Command…)". This dialog is similar to the kind you create with native DCL except that it is possible to allow the user to resize the dialog box on screen.

Create Two Forms



On the tool bar under file you should see a white icon at the left of the tool bar. Press this twice. This will insert two forms into the project workspace. In the forms title bar you should see an AutoCAD icon followed by DclForm1 and DclForm2. Each time you add a form its name is displayed in a tree in the project window at upper right of the workspace. To remove an unwanted form from the workspace, right click on the form name in the project window and select “Remove…” Note that the arrow next to the white icon has other types of forms that can be inserted. The number of types do very depeding on the version of software purchased. See “Dialog Forms available in ObjectDCL” section.

The default form name given by the system allows one to insert and locate several forms without naming them first. You are expected to change this name to something meaningful, in fact it is highly recommended as that "DclForm1", "DclForm2", etc. would be confusing coming back to the sometime later by you or another programmer. Please refer to handling events under “Programming Tips” in the ObjectDCL Project Section. The information provided there will explain the significance of starting with the desired name and what is required to change it after events have already been programmed.

The Properties window at lower right has not yet displayed anything. With the mouse pointer, pick in the gray body of the DclForm1 and notice that the Properties window now displays information about the form selected.

Pick the title bar of DclForm2 and see that it is highlighted in the project window. Also notice that the Properties window still displays properties for form1. To show properties for any form one must select in the body of the form rather than just the title bar.

Place close attention to this behavior when adjusting control properties between two forms. While displaying properties of a control in one form, pick a control in another form an notice that the form and control appear to have focus but the properties remain unchanged. To work around this one must always pick in the body of the form first before selecting a control on a new form.

Toggle back and forth between the bodies of two modal forms and notice that the Properties list changes for only two properties. The name of the form and the title bar text to be displayed. The "(Name)" is how the form is identified in the code and the "TitleBarText" is what will be displayed when the form is displayed. These names do not usually stay the same although they can be the same. The "TitleBarText" can be changed at will both in the editor and by AutoLisp, but the "(Name)" must not be changed without changing all references too it. The "TitleBarText" is just what the user will see on the form and nothing more.

To edit properties in the property window use the mouse to highlight properties in the left column. Notice that there is a description that appears at the bottom of the widow under each property name. Also notice that some of the properties have controls that appear in the right panel that provides selection options.

ToolBox

The ToolBox at left contains buttons that can be selected to insert control or select control in the Dialog box form. Pausing over a button launches a tool tip describing what the button is for. Tool tips are not displayed unless the ObjectDCL application has focus even though these buttons highlight as the mouse passes over.

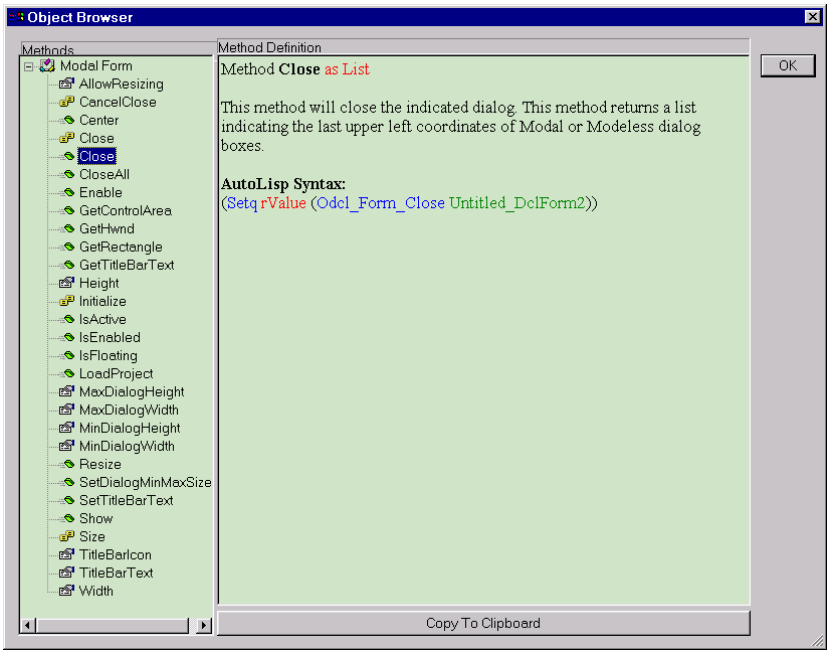
The number of valid controls shown in the ToolBox is dependent on which version of ObjectDCL you have. The black arrow in the upper left corner is the selection tool. This tool is automatically activated each time a control is inserted into a form. When this tool is active a selection arrow is displayed. Use this arrow to give focus to any of the controls in the form by selecting in the body of the control. When a control has focus, hot grips appear that can be gripped to resize the control using the arrow.

Inserting a control

Select the control below the “T” in the ToolBox. Selecting the Label control causes the selection arrow to turn into cross hairs when over the body of the form. Notice that the crosshairs themselves do not snap to the grid. When the pick button is depressed an outline starts nearest a grid point and can be dragged into a box shape. The crosshairs appear offset from the outline being shown during dragging. Resizing by dragging the corners of a control causes the properties to update as soon as the mouse button is released. Some of the properties in the list will also change the location and size of the control.

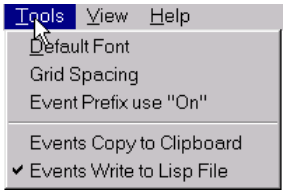
Object Browser

After inserting a control, right mouse click on it to show a menu that contains the Object Browser selection at the bottom. The option show a window that contains a list of methods, properties, and events that are valid for use with that control. Notice that the colored icon is specific to each category. White for properties, yellow for events, and green for methods. Some selections display a copy to clipboard option that reduces the amount of work needed to create the supporting lisp code. Methods that are used to set or get properties have two buttons that allow the desired method to be copied. These methods for working with properties are associated with the with icon. Please do not be confused by the window labels. The the left list should be labeled methods, properties and events. The right window would be better labeled as simply "Description". The AutoLisp Syntax is the method that is doing the work. The first statement shown, "Method **Close** as List" means that this is a close method and that it returns data in the form of a list.



Grid Size

Please refer to the Tools menu section.



Properties

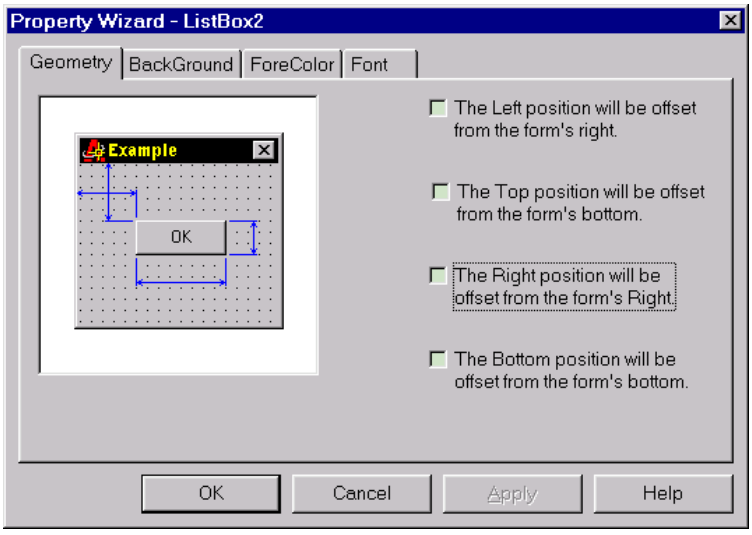
ObjectDCL provides many properties for each control. Some properties are always present for each control, such as Visible and Enable. We could go into a complete list of all properties for each control and their definitions, but we will only describe the most important properties here for you. For a complete list, see the online help.

(Name)	The Name Property is a tag that is used by ObjectDCL to determine which control or dialog to access. Initially given as the name of the control. Can be changed to suit.
(Object Browser)	A comprehensive list of methods that apply to this specific control. RMB on any control will bring up a menu where this choice is available.
(VarName)	This name is a composite. "ODCL"+_"Filename"+_ "(Name)"+_"ControlName" The VarName becomes a global symbol when the dialog is called. At run time it contains a long integer value that ODCL uses as a memory address. If you change the name of the dialog after it is programmed and add controls, the name will look different then what was previously programmed. This can cause your new control

	to appear as if it does not work. Be very careful about changing the dialog name or filename of your project.
(Wizard)	Activates the Properties Wizard dialog box for each control by clicking on the dotted button shown when this property is selected. RMB on any control will bring up a menu where this choice is available.
Caption	Sets the text displayed in the Label control.
Enabled	When True the control is usable by the user, when set to false the control is grayed out and unusable.
EventInvoke	If set to "0 - Direct Invoke" the event will be called directly with the most efficient ObjectARX method to call AutoLisp functions. If set to "1 - Allow (Command…)" then the AutoLisp program will be able to call the (Command …) function, but the command line will take focus from the control. The setting of 1 is the default for the Modeless and Dockable forms, while a setting of 0 is the default for the Modal form. The property is only available in AutoCAD 2000 and 2002.
Font	This setting will set the font used by the control.
Height	Sets the Height of the control.
Left	Sets the distance from the left edge of the dialog box to the left edge of the control.
Picture	Selecting the picture name from the picture collection will set the picture to be displayed in the control.
Text	*This property controls the text to be displayed in the control. Most useful for setting and retrieving the values from the TextBox control.*
TitleBarText	*Sets the text to be displayed in the title bar of the form.*
Top	*Sets the distance from the bottom of the title bar to the top edge of the control.*
Value	*This property contains the value of the SliderBar, ScrollBar, SpinButton and AngleSlider controls.*
Visible	*Determines if the control is visible to the user at run time.*
Width	*Sets the width of the control.*

Property Wizard

The Property Wizard for each control can be accessed either by right mouse click, double-clicking on the control or picking the "(Wizard)" dotted button from the property list in the lower right side of ObjectDCL. The Property Wizard displays the appropriate tabs for the control being inspected. Possible dialog tabs with associated controls are:

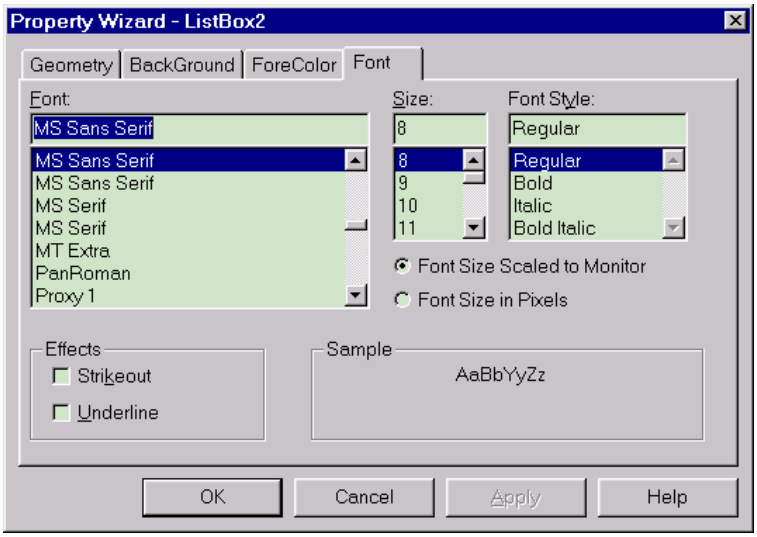


Geometry

As the dialog box is resized the controls will know how to move or resize themselves within the confines of the dialog box or tab pane. This can only be seen at run time. For check boxes are provided to define from where on the form a control maintains its position. The graphic shows an extension line to assist in visualizing the affect. What this means is that, as the dialog box is resized the controls will know how to move or resize themselves within the confines of the dialog box or tab pane. This has been provided to simplify resizing AutoLisp programming to a minimum. **Note:** It's also highly recommended that all controls placed in a Config Tab pane have resizing rules applied, because the Preferences dialog box adjusts its size according to the screen resolution of the computer.

Font

Used to assign a system font to a control. ObjectDCL uses two editions to the fonts that are not normally available, "Font Size Scaled to Monitor" and "Font Size in Pixels".



BackColor

Assigns a background color from the standard AutoCAD 256 color pallet, or a Windows System color value.

ForeColor

Assigns a foreground color from the standard AutoCAD 256 color pallet, or a Windows System color value.

Image List

Used to define what images are available for use at run time by the Tree and ListView controls. These controls require all their available images to be loaded into a separate image list collection in the ObjectDCL editor. Even though the Picture Folder provides ready available pictures for the PictureBox and GraphicButton, the Tree and ListView controls require all their available images to be loaded in the property wizard. It's important to mention that copying and pasting will not carry images across to the pasted control and that all images loaded into one Image List must be the same size.

Button Styles

Used to assign a specific type of visual button style to a control.

Combo Styles

Some styles define how the combo box behaves and others define what will automatically populate the control, such as an AutoCAD color list, line weights or arrowheads.

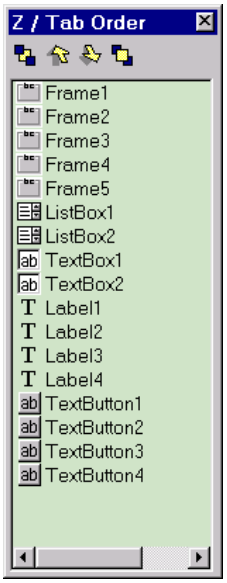
Filters

The filters option allows text-input filters to be assigned to a text box control.

Tabs

The Tabs option allows on to define the text label and quantity of tabs displayed in a TabControl.

Z / Tab Order



The Z / Tab Order list box shows each control inserted in the current form. The "Tab key" moves the focus to controls in your dialog in the order they appear descending the list. Selecting a control in the list shows the grips on the control and its properties in the property list.

Using the Tab Order list for selecting controls avoids accidentally moving them with the mouse while selecting them on the dialog form. Note that the frames have been place at the top of the list at right. This allows the controls that follow to appear inside the frame by sitting on top (in the foreground). The buttons at the top of the list allow moving a highlighted control(s) around in the list. More than on can be selected using standard Windows means.

Project Lisp file

Each project has an associated lisp file. The beginning of this file should include a subroutine that makes sure that the objectdcl.arx is loaded. The defun below should be called each time a particular form is launched by the lisp code calling that form.

```
(defun ObjectDCL_LoadArx ()
  (if (not (member "objectdcl.arx" (arx)))
    (arxload "objectdcl.arx" "ObjectDCL.arx not found.")
  )
)
```

Note: Each defun below is preceded by a c: that ObjectArx requires in order to activate the event defuns by some dialog box styles. For consistency the c: is used all the time by all dialog box styles.

For example:

```
(defun c:Hello ()

  ; ensure the ObjectDCL.arx file is loaded
  (ObjectDCL_LoadArx)

  ; call the method to load the Hello.odc file.
  (Odcl_LoadProject "Hello")

  ; call the method to show the Hellow world dialog box example
  (Odcl_Form_Show "Hello" "DclForm1")

  (princ)
```

)

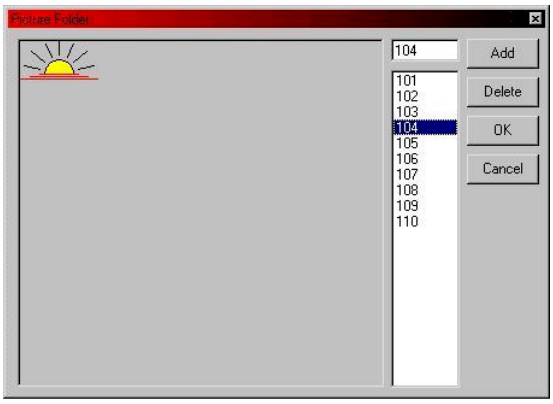
```
(defun c:DclForm1_ DclForm1_OnInitialize ()
    ; call the method to set the textual display of the Label to state “Hello World”
    (Odcl_Control_SetProperty "Hello" "DclForm1" “Label1” “Caption” “Hello World”)
)
```

This statement replaces the action_tile call back that is commonly used by generic dcl:

```
(defun c:DclForm1_TextButton1_Clicked ()
    ; call the method to close the Hello World dialog box example.
    (Odcl_Form_Close "Hello" "DclForm1")
)
```

The defun created by the ObjectDCL can return more information than is possible with an action_tile.

Picture Folder



To add pictures to the GraphicButton and the PictureBox, pictures may be loaded into the Picture Folder first. The picture folder is where these two controls retrieve the pictures that they display. ObjectDCL allows you to add picture files such as icons, bitmaps and JPEG’s into the project file to be loaded into the GraphicButton or PictureBox controls. The picture folder can be accessed through the ‘Project’ – ‘View/Edit Picture Folder’ menu, the toolbar button with the camera icon, or you can directly add a picture to the folder and assign it to a control using the Picture property. If you wish to display a picture loaded from a file at run-time, use the Odcl_PictureBox_LoadPicture call of the PictureBox to display more sophisticated picture files, such as the Bmp, Jpeg, Ico, and Wmf’s. This call will paint the picture in the PictureBox.

Tools menu

ObjectDCL comes with a Tools menu. This menu gives you control over default grid and fonts used to when creating a new control. Checking Event Prefix use “On” will and On to event names, for example, “Clicked” vs. “ OnClicked”.

Grid Setting

The menu option Tools/Grid Spacing allows the user to change the default grid displayed while editing forms to the spacing you may require. The slider allows a setting of 0 and 5 through 20. Any setting of 0 will make the grid disappear.

Default Font

The menu option Tools/Default Font allows the user to set the default font that all textual controls use and can be set to any font available on your system. If you specify a font that is not normally

on a standard computer you will be responsible to distribute this font with your program. Please note that this setting does not apply to active activeX controls. ActiveX controls have existing default fonts already setup.

Security Protection

Under File menu, picking the selection “Create Distribution File” will launch a file dialog that displays .odc files to package. In order to use this option you must first have saved your project which creates an .odc file. However, it is not necessary to have a project open when creating a distribution file. Once the distribution file has been created, it is rebuilt each time a change is made to one of the .odc files by the ObjectDCL Editor.

You can select multiple .odc files to include in your target .ods distribution file. Once you have made your selection a second save-as box will open allowing you to save your .ods file. Also notice that there is an entry in the project tree called “Distribution File Name” under which can be seen the current paths to the project .ods file. Dbl-clicking the path will bring up a save-as dialog that allow changing the target filename.

Dialog Terminology

Controls

Controls are objects that are re-usable and provide the parts of the visual interface of a program. (What you see.) Examples include Label, TextBox and TextButton.

Event

An action that was initiated by the user or by the operating system of the program. Examples include a keystroke, a mouse click, etc.

Forms

A dialog box that floats above the AutoCAD main window or is docked to one side of the main window.

Picture Folder

A collection of pictures and images that are used by most controls that can display user defined images.

ImageList

An internal list or collection of images stored separately from the picture folder. The ImageList is attached to the control that makes use of the image list. At this time the Tree and ListView controls use the ImageList to specify which items will display which image(s).

Methods

A method is a lisp program function that is provided to manipulate form or control and place or retrieve information. Methods work at runtime in all events. Runtime starts after the dialog has been loaded into AutoCAD, as opposed to designtime, that’s when you are working in ObjectDCL. The OnInitialize event that occurs after the form show method is called is also considered runtime. Methods are used in the onInitialize event to prepare the dialog form to be displayed. After the form is displayed, many methods are available to manipulate the form.

Object

An object is a basic, essential component of a program, which contains properties to define its characteristics and methods to define its tasks and recognizes events to which it can respond. Examples of objects in ObjectDCL include controls and forms.

Properties

A property is a distinguishing characteristic of an object like its size, position, text font, or color. Methods are used to used to contribute data to an object and retrieve information from the object by retreiving or setting its properties..

Tile

This is the DCL name used to designate a control on a DCL dialog box.

Projects

ObjectDCL project files are organized differently internally than the old DCL files. The reason the format is so different is because it is a binary file format. This allows the .odc file format to store more information in a secure, compact, quick to load, format. This also allows for images and pictures to actually be stored internally in the project file.

ObjectDCL can display more than one dialog box style at a time. This allows ObjectDCL to load more than one project file at a time. Because of this, each call to an ObjectDCL form or control must include the project file string (must be the same exact string passed into the Odcl_LoadProject call), and the Form name string. Because of the versatility this allows, you will be required to keep track of these two strings.

Programmers developing on a **network drive** should be aware that problems with loading the most current save for the Editor have been reported. This was may be do to time differences between the network computer and local computer. It is thought that older files in cache were loaded rather than the most recent save. Also, including the .odc extension with the project name forces a C++ file search rather than the AutoCAD findlile search. Using the odc extension may cause a more reliable update. The existence of the .odc extension will not cause a problem when there is only the .ods file available. When both are present the Editor uses the .odc file. For best results, keep the .odc file on your loacal machine. If there remain problems getting the project to update unload and reload the ObjectDCL.arx file.

(Odcl_LoadProject ProjectFileName [Optional]ForceReload)

This method loads the requested project into memory for later use.

Arguments:

A. A string is required to indicate which project file to load.

B. An option 1 or T will force the project to be reloaded if previously loaded. This argument is useful for testing and debugging. It's recommended that this optional argument be removed before distribution.

Programming ObjectDCL Forms

The ObjectDCL Editor provides a way to change the properties that manipulate a form or controls appearance. Once the forms are displayed in AutoCAD they can only be modified by the method calls provided by ObjectDCL, like Odcl_Form_Center. Modifications are not allowed while the form is showing on the screen using Odcl_Control_SetProperty. Doing so will provoke an error

message box. Use Odcl_Control_SetProperty to setup the the dialog box before calling form show. Use the OnInitialize event for this purpose.

Dialog Forms available in ObjectDCL

Of the five forms listed below only the first can be created with Standard DCL.

Modal Forms(all versions)

While this box has focus the user may not interact with AutoCAD. This is Standard DCL behavior.

Modeless(Std & Pro)

Allows the user to interact with the AutoCAD while the dialog is being displayed. Standard DCL does not allow this. A modeless dialog box that allows the user to take actions outside of its boundaries. Basically it acts the same as a floating toolbar. It allows the user to draw and interact with AutoCAD but floats there on the screen awaiting user input.

Dockable(Pro)

A style of modeless dialog that will dock at the edge of the screen. This dialog box will allow itself to be docked to side, top or bottom of the AutoCAD main frame. With this style resizing is obviously a necessity and a rigid fixed size dialog box is not allowed.

Configuration Tab (Pro)

This style in AutoCAD 2000 and 2002 allows you to create your own tab pane that will be displayed in the AutoCAD Options / Preferences dialog box. To have the tab display automatically when the dialog box is called, the project must be loaded into memory first.

File Dialog (Pro)

It is preferable to use **(getfiledia BoxTitle DefaultFileName DefaultExtension Flags)** for most open and save duties. See: “Selecting Single file with Regular VisualLisp “ under “Other ObjectDCL Commands”. Also if there is a need to select multiple file at once see: **(Odcl_MultiFileDialog [Optional] sFileFilter [Optional] sTitle [Optional] sInitialDir)** in the same section.

Customizes MS standard file dialog to which you can add additional controls. The file dialog appears in its minimum size when first displayed in the ObjectDCL Editor. If you add controls to the standard dialog they will swap sides with the predefined file dialog pane.

Note that this dialog box is used and customized by many widows programs. Windows remembers the last size of the file dialog displayed by any windows program. For this reason if you resize the box for your purpose it will appear that size when launched by another program.

Usage

When the user selects a listed file, the name of the file is returned as a string with no path.. When the form first shows there is no file selected. Pressing the open button has no affect until either a file is selected or something is typed in the edit box. The path must be found using a (setq path (odcl_…GetPathName CustDialog) method that is best placed in the OnClose event before form

show. Since the form show returns either the selection or nil, test the results of form show and (strcat the path and filename to create a proper

CreationPrompt

This property when true causes the following behaviour:
If the file name is typed in the box that does not exist, a message asks if you wish to create a new file. A “Yes” results in the name that was typed to be returned. A ”No” returns focus and highlights what was typed.

When the CreationPrompt property is false a message asks for you to verify the the file name is correct. Focus does not return to the edit box.

Properties

AsReadOnly
CreationPrompt
EventInvoke
ExtCanBeDiff
FileMustExist
Filter
Height
Left
MutipleSelction
OverWritePrompt
PathMustExist
ShowCancel
ShowReadOnlyCheckBox
ShowTypeComboBox
ShowTypeLabel
Style
Top
Width

Methods

See: **Other ObjectDCL Commands**
For information on methods used with the filedialog.

Controls available in ObjectDCL

See “Control Event Programming” for more information about how to interact with the controls listed.

AngleSlider

The AngleSlider control has no DCL equivalent. This control is a round slider control that displays in its center the currently selected angle in degrees.

BlockList

The BlockList control has no DCL equivalent. This control is based on the ListView control and will list all the blocks in the current drawing and their preview images (if any) in two display formats. Available in AutoCAD 2000 and 2002 only.

BlockView

The BlockView control has no DCL equivalent. This control will display any 2D or 3D block stored in the current drawing or load any DWG file for full display as well. As an added feature this control allows the user to Orbit, Pan and Zoom the blocks that it currently displays. Available in AutoCAD 2000 and 2002 only.

CheckBox

The CheckBox control is equivalent to the Toggle tile in DCL. This control provides a small box that displays a check box when set.

ComboBox

The ComboBox control is equivalent to the Popup_List tile in DCL. This control provides a text box with a drop down button that displays a ListBox when clicked. The main difference is that there are several styles available, where only one was previously available. These styles include AutoCAD colors, AutoCAD arrowheads, AutoCAD line weights, Combo (where the text box is editable), Simple (where the list box appears below the editable text box), drop down (where the text box is read only), AutoCAD plot style names and AutoCAD plot style tables.

DwgPreview

The DwgPreview control has no DCL equivalent. This control will display the DWG thumbnail preview of a selected DWG file.

Frame

The Frame control is equivalent to the Boxed-Row and Boxed-Column tiles provided in DCL. This control provides an etched border, used to visually organize controls and information. Unlike DCL, a parent/child relationship does not exist in ObjectDC between the frame and its contents.

GraphicButton

The GraphicButton control has no DCL equivalent. It is a button that allows the user to display any user defined or pre-provided graphic (Bitmaps, Icons, and JPEG files). There are two border styles available. The first is the standard style identical to the TextButton Control's border. The second border style is the new 'Cool' button style - identical to the Internet Explorer buttons - where no border is displayed until the user moves the mouse over the control, at which time a thin border appears.

HtmlControl

The HtmlControl has no DCL equivalent. This control acts as a self-contained web browser that can be placed in a dialog box. Microsoft Internet Explorer 4.0 or better must be installed on the end user’s computer for this control to work.

Label

The Label control is similar to the Text tile in DCL. Some plain DCL tiles have a built in label that is a property of the tile. Some ObjectDCL controls have an equivalent “Caption” property and others do not. The purpose of a Label control is to display text anywhere on the dialog form. This control is static at runtime in the sense that the viewer can not change it. Click and double click are the only events possible for a Label control. A label’s justification property respects only the 0 – Left case.

ListBox

The ListBox control is equivalent to the List_Box tile in DCL. This control displays a list of information the user may pick from.

ListView

The ListView control has no DCL equivalent. The control is provided by Microsoft's Internet Explorer and is implemented for your use. This control will list multiple information and images in four display formats.

MonthCalendar

The MonthCalendar control has no DCL equivalent. This control displays months and days to the user in a simple calendar format, allowing them to select the exact month or day.

OptionButton

The OptionButton control is equivalent to the Radio_Button tile in DCL. This control provides a small round box that displays a bullet, when set.

PictureBox

The PictureBox control has no DCL equivalent. It displays any picture previously loaded into the project's picture library (at design time) or load a picture file at run time. This control uses new graphics technology to load and display Bitmaps, Icons, JPEG's and WMF files. For example it can be used to display company logos, screen captures, etc. This control also comes complete with a large compliment of drawing functions that could be used to create your own custom controls.

ProgressBar

The ProgressBar control has no DCL equivalent. This control is used to display a visual progress to the user indicating completion of a time consuming computational function.

Rectangle

The Rectangle control has no DCL equivalent. This is a simple control meant only to act as different kind of rectangle to visually organize controls and information.

ScrollBar

The ScrollBar control is equivalent to the Slider tile in DCL. This control allows the user to slide a button along the bar to pick a value, or may be used to control the display of information in another control(s).

SliderBar

The SliderBar control has no DCL Equivalent. It is a control that is similar in function to the ScrollBar control, but visually different. This control is more suited to allowing the user to visually pick a numeric value.

SlideView

The SlideView control is equivalent to the Image and Image_Button tiles in DCL. This control displays an image from a slide file or slide library file created in AutoCAD.

SpinButton

The SpinButton control has no DCL equivalent. This control is used in conjunction with a TextBox control with an integer filter setting. As the user clicks on the arrows, the corresponding TextBox controls value changes.

TabControl

The TabControl has no DCL equivalent. This control displays multiple tabs to the user for selecting. As each tab is clicked a different set of controls is displayed to the user. At design time, to achieve this, separate forms are supplied for each tab. These forms are used as tab panes to display the required controls under each tab when clicked.

TextBox

The TextBox control is equivalent to the Edit_Box tile in DCL. This control is a simple text editor and comes with several user input filter styles - some not previously available. These filter styles include string, AutoCAD angle units, integers, AutoCAD numeric units, AutoCAD symbols, uppercase, lowercase, password, and Multiline string. This box will word wrap.

If any of these predefined filter styles do not match what your program needs, a method call Odcl_TextBox_SetFilter has been provided to allow your program to define at run time which characters should be allowed into the TextBox. See the definition of mentioned method below to learn exactly how to put it to use.

TextButton

The TextButton control is the equivalent of the Button tile in DCL. This control has one border style as in DCL, and only displays text. Also see Frames for warning.

TreeControl

The TreeControl has no DCL equivalent. This control is used to efficiently organize and display large amounts of information to the user without clutter or taking up large amounts of screen real estate. This control is the same control used in the Windows file explorer that displays the file directories.

Calls to tree controls can return either a descriptive string or long integer ID. The ID represents the position of a node on the tree. These nodes are independent of the name of the item located there. Think of them as addressees for that node position.

UrlLink

The UrlLink control has no DCL equivalent. This control is similar to a Label control except that it displays its text as a visual URL link that will launch either an email or a web page.

Using Controls

What follows is a discussion of programming for various controls used by the author.

ComboBox

The ComboBox is very versitile as can be seen by inspecting its property list. I is really three controls in one. An edit box, a button and a list box. Launch the ComboBox Wizard and click on

the ComboBox Styles tab. Notice that there are 12 styles available. Notice that styles 0 ans 1 are editable by the user. Style 2, the familiar drop down selection box is read only.

The text methods apply to the editable boxes only. Define the internal list.

Event Programming

(varname)

This name is a composite. “ODCL”+_“Filename”+_“(Name)”+_”ControlName”
Where Filename is the name of the .odc file that contains your form and (name) is the name of the form shown at the top of the property list.

The VarName becomes a global symbol when the dialog is called. At run time it contains a long integer value that ODCL uses as a memory address. If you change the name of the dialog after it is programmed and add controls, the name will look different then what was previously programmed. This can cause your new control to appear as if it does not work. Be very careful about changing the dialog name or filename of your project.

All forms can still be called using the string method used in version 1 that was kept for legacy support.

Ver1 method

(Odcl_Form_Show "ProjectName" "FormName")
Please note that the "ProjectName” has to be the exact same string used in the Odcl_LoadProject call. It need to include any path or extension that you may have used. The form name is case sensitive and must appear exactly as it does in the form list.

Ver2 method

(Odcl_Form_Show ProjectName_FormName)

Using the global symbol (ver2) method yields very fast results especially significant when calling picture box with images. Version 1 method always works because it looks up the form in a table to find its address. That can be useful for testing your code. That allows one to show that the form exists even thought he varname is incorrect.

Label

The Label control is similar to the Text tile in DCL. Some plain DCL tiles have a built in label that is a property of the tile. Some ObjectDCL controls have an equivalent “Caption” property and others do not. The purpose of a Label control is to display text anywhere on the dialog form. This control is static at runtime in the sense that the viewer can not change it. Click and double click are the only events possible for a Label control.

The Label can be used for labeling other controls like the Listbox. The “Caption” property shows text on the form at design time. At run time (Odcl_Control_SetProperty controlname “Caption” “string”) can change the text displayed. One can also retrieve the current text using Odcl_Control_GetProperty.

ListBox

The ListBox control is equivalent to the List_Box tile in DCL. This control displays a list of information the user may pick from. List index is zero based (first item is 0).

Common Lisp Driven Events

Odcl_ListBox_AddList Fill with list of strings
Odcl_ListBox_Clear Clear listed items
Odcl_ListBox_AddString Add a string at the bottom of list
Odcl_ListBox_DeleteString Delete a string at index position
Odcl_ListBox_SetCurSel Highlight string at index position

Common Form Driven Events

When ObjectDCL creates event functions for you it adds the appropriate form name and control name as a prefix to the function name. This is represented below by the dots after C:. These events must be checked in the Editor to be active in the ARX file.

C:…On_Initialize ()
Place ObjectDCL methods inside this lisp function to set up the initial state of the dialog form.

C:…OnSelChanged (nSelection sSelText)
The parameter nSelection is the index of the item having focus. While sSelText returns the string displayed at the index.

C:…OnKillFocus ()
User is leaving a control. This event will fire when enter pressed if enter is changing focus to another control.

C:…OnReturnPressed ()
The user pressed enter. Include inside this defun what action needs to be taken to update the dialog form. This event always comes completes before kill focus event fires. Pressing return in a list box will close the dialog as an accept. The method return as tab will be added at a later date.

C:…OnDblClicked ()
Like above.

Method Programming

A method is lisp program function that is provided to manipulate form or control or retrieve information while the form is displayed. Note that methods intended to work with forms have "Form" in the name and methods that work on controls have "Control" in the name. (Odcl_Form_SetTitleBarText FormName "New Text") will show the desired text on the title bar when the form is being displayed.

Control Properties, working with-

Below you will find a list of often used properties and common to selected controls. The lisp function that works with properties is written as follows:

```
(Odcl_Control_GetProperty
        VarName
        "Property"
```

)

(Odcl_Control_SetProperty
VarName
"Property"
Setting
)

All control properties can be queried or changed using the above functions. SetProperty can be used to change the controls properties before the form is displayed if included in the OnInitialize event defun.

"Caption"

The "static" text displayed on a control. In DCL this was called the label. It is not changed by the user but can be changed programmatically after the form is shown. By this definition caption will not be found on controls that accept textual input from the user.

"Enabled"

True = 1 False = 0 [Boolean]
The tile is functional when enabled is true and disable when enabled is false. When disabled the control is grayed out.

"Visible"

When false hides the control on the form.
True = 1 False = 0 [Boolean]

(Odcl_ControlZorder ControlName Direction)

This method positions the control in the TaB Order.
0 being last
1 being first

Other ObjectDCL Commands

ObjectDCL provides some utility commands that offer new capabilities.

(Odcl_ActivateEmail sEmailAddress)

This method will launch a new email message from the user's default email program and address it to the supplied email address.

(Odcl_BrowseFolder sTitle [Optional] sInitialFolder)

This method will prompt the user with a dialog box that allows the selection of directory folders only.
Arguments:

STitle	A string that describes this dialog box's purpose.
SInitialFolder	An optional string that selects a directory at startup time of the dialog.

Returns a NIL if the user canceled the dialog box or a string indicating the directory chosen.

(Odcl_GetColorValue Red_Or_ColorIndex [Optional] Green [Optional] Blue)

This method will return the color value specified from the arguments. The RedOrColorIndex can be set as 1 to 255 for an AutoCAD color or -1 to -23 for a system color. If you need to create a different color, set the red, green and blue values from 0 to 256. The value from this method is only useful for setting colors of ActiveX controls.

Arguments:

Red_Or_ColorIndex	An integer that indicates the AutoCAD color index, system color or red value.
Green	An optional integer that indicates the green value.
Blue	An optional integer that indicates the blue value.

(Odcl_GetHardDriveSerialNumber sDrive)

Argument:

sDrive

A string that indicates the drive letter, example "C:\\", that you require the serial number from.

Example:

(Odcl_GetHardDriveSerialNumber "C:\\")

Returns:

The serial number of the selected drive.

(Odcl_GetHardDriveSize sDrive)

Arguments:

SDrive

A string that indicates the drive letter, example "C:\\".

Returns: The size of the selected drive.

Example: (Odcl_GetHardDriveSize "C:\\")

(Odcl_GetPictureSize ProjectFileName PictureId)

Arguments:

ProjectFilename

The same string used in the Odcl_LoadProject method is required to indicate which of the possible multiple projects that could be loaded to use.

PictureId

An integer indicating the picture id of which picture is to be queried for its size.

Returns: (nWidth nHeight) the size of a picture, indicated by its Id.

(Odcl_GetScreenSize)

Returns the current resolution of the display.
Ex: 1024 x 768

(Odcl_GetVersion)

This method returns the version of the currently loaded ObjectDCL2.arx file.
(Please note that this is not available to ObjectDCL.arx file for version 1.0 through 1.08.)

(Odcl_HideErrorMsgBox)

This will turn off all error message boxes ObjectDCL generates to inform the AutoLisp programmer of programming mistakes. This method is intended to be called by an AutoLisp program once it's released to market.

(Odcl_LoadProject ProjectFileName [Optional]ForceReload)

This method loads the requested project into memory for later use.
Arguments:
ProjectFileName

A string is required to indicate which project file to load.

ForceReload

An option 1 or T will force the project to be reloaded if previously loaded. This argument is useful for testing and debugging. It's recommended that this optional argument is removed before distribution.

(Odcl_MessageBox "Message ""Title" [Optional]ButtonFlag [Optional]IconFlag [Optional]HelpFlag)

Use this method when calling an alert box from an ObjectDCL event defun rather than the native AutoLISP Alert Box. It uses Microsoft's system MessageBox. The Alert Box allows the user to still access an ObjectDCL form while it is displayed.

This method displays a standard windows message box. The message box contains a message and title, plus any combination of predefined icons and push buttons. In its simple form:

(Odcl_MessageBox "Message ""Title")

Arguments:

Message
A string that will be used display the message to the end user. Control character \n preceding text will cause it to display on the next line.One can use (strcat to combine several strings for readability.

(setq Message
(strcat
"First line"
"\nSecond line.
))

Title
A string indicating the title to be displayed on the dialog bar.

ButtonFlag
An optional integer argument that controls which button will be displayed in the message box.

ButtonFlag Description

1 Display three push buttons: Abort, Retry, and Ignore, with Abort as default.

2 Display one push button: OK. This is the default if no value is entered.
3 Display two push buttons: OK and Cancel, with OK as default.
4 Display two push buttons: Retry and Cancel, with Retry as default.
5 Display two push buttons: Yes and No, with Yes as default.
6 Display three push buttons: Yes, No, and Cancel, with Yes as default.
11 Display three push buttons: Abort, Retry, and Ignore, with Retry as default.
12 Display two push buttons: OK and Cancel, with Cancel as default.
13 Display two push buttons: Retry and Cancel, with Cancel as default.
14 Display two push buttons: Yes and No, with No as default.
15 Display three push buttons: Yes, No, and Cancel, with No as default.
21 Display three push buttons: Abort, Retry, and Ignore, with Ignore as default.
26 Display three push buttons: Yes, No, and Cancel, with Cancel as default.

IconFlag
An optional integer argument that controls which icons if any will be displayed in the message box.

IconFlag Description

0 No Icon will be displayed. This is the default.
1 An exclamation point icon is displayed in the message box.
2 An icon consisting of a lowercase letter ‘i’ in a circle is displayed
3 A question mark icon is displayed in the message box.
4 A stop sign icon is displayed in the message box.

HelpFlag
An optional integer argument that indicates if a Help button should be added to message box.

HelpFlag Description

0 No Help button will be displayed. This is the default.
1 A Help button will be displayed along with the other button(s).

On closing dialog

Integer Returned:

1 The OK button was clicked.
2 The Cancel button was clicked.
3 The Abort button was clicked.
4 The Retry button was clicked.
5 The Ignore button was clicked.
6 The Yes button was clicked.
7 The No button was clicked.

(OdcI_MultiFileDialog [Optional] sFileFilter [Optional] sTitle [Optional] sInitialDir)

or

(Odcl_MultiFileDialog [Optional](sFileFilter…) [Optional] sTitle [Optional] sInitialDir)

This method will prompt the user to select one or more files and will return a list of the files selected including there full paths. It uses Standard Windows mutifile selection methods. When calling this method you have the option of not specifying any file filter, this will default to "AutoCAD Drawing (*.dwg)|*.dwg". For the occasion that a different file filter or filters is required you can pass a simple string or a list of string indicating the filters to be used.

Please note when specifying a filter the description of the filter must always be separated by a | symbol (this is most likely the character above the \ character on your keyboard). Some other examples are listed below:

Arguments:

SFileFilter
An a single string or list of strings that will be used as the filter string for extensions.
sTitle
An optional string that will be displayed in the title bar of the file dialog.
SinitialDir
An optional string that will be the default initial directory of the file dialog. Please note that all directory strings must be defined in the format: “c:\\dir”
Examples:

(Odcl_MultiFileDialog

(list "Picture Files|*.bmp;*.jpg;*.ico;*.dib”

“Bitmaps (*.bmp,*.dib)|*.bmp;*.dib”

“JPEG Files (*.jpg)|*.jpg”

“Icon Files (*.ico)|*.ico"
)

“Select the picture file(s)”

“c:\\pictures”

)

Returns
This method will return a list of the selected items. If the user canceled the dialog box a NILL will be return.

Selecting Single file with Regular VisualLisp

(getfiledia BoxTitle DefaultFileName DefaultExtension Flags)
BoxTitle – title
DefaultFileName – default file name or path
DefaultExtension – extension filter limits display list to file with that extension

Flags – behaviour control

New File save in: commonly uses Flag 1
Open File look in: comonly uses Flag 0

Flags:
Bit coded field accepts values 1-15 representing the choices below added together.
Flag = 1 (bit 0) Prompts for name of new file Save in:
Flag = 4 (bit 2) Arbitrary file name allowed Look in:
Flag = 8 (bit 3) Set without Flag 1 performs library search but does not return path.
Flag = 16 (bit 4) Uses default file name for path only. Leaves name box blank
Flag = 32 No overwrite warning when creating new file (Flag 1).
See Visual Lisp Help for a more complete and confusing description of flags.

(Odcl_NavigateToUrl sUrlAddress)

This method will launch a web browser and automatically link up to the supplied Url address.

Arguments:
SUrlAddress A string indicating the Url address to be linked to.

(Odcl_NavigateToUrl sUrlAddress)

This method will launch a web browser and automatically link up to the supplied Url address.
Arguments:
sUrlAddress A string indicating the Url address to be linked to.

(Odcl_SetCmdBarFocus)

This method forces the command bar to take focus from the current active Modeless or Dockable dialog box.

ObjectDCL Programming Tips

Many users have asked question about how to program a certain procedure or control. Answers to these questions have been posted to www.objectdcl.com/knowledgebase/knowledgebase.html for everyone to download. Other programming tips are provided below.

Changing Project and Form names

So, after studying your lisp code for hours you realize that the names used by odcl are way to long and you want to go back and redo the names in your project. Yes, this is possible. Can you avoid doing this and finish your project? If you do choose to change it goes like this.

Save the current project file to a new file as “1.odc” and perform the following operations to each form in the project. Click on the body of the form. Change the name on the form to something like “A”, and a dialog will request that this name be used for all varnames and their children. Answer yes. Then go through the list of controls and re-check each event so the name will update. Now go to the lisp code and save that as a new name to experiment with. Carefully replace the project name and form name with the new ones you just made. Load the new project and see if the new code will run the dialog. There will probably be some pat that you missed updating. If all fails go back to the original files and try shorter names on your next project.

Disallowing Close Dialog

To prevent the user from closing a Modal or Modeless dialog a CancelClose event has been provided. Available in AutoCAD R2000 and 2002 only.

```
;The following code below will allow you to tell ObjectDCL if closing should be allowed.
(defun c:DclForm1_DclForm1_CancelClose (bUserPressedEsc / )
  (setq cancelClose nil)
  (if (= AllowClose T)
    (setq cancelClose nil)
    ; else
    (setq cancelClose T)
  )
  ; place the variable to inform ObjectDCL is it should cancel the close.
  cancelClose
)
```

Drag and Drop

Over half of ObjectDCL's controls support drag and drop. The TextBox and ListBox will allow the user to drop something into them, but will not allow the user to drag from them to another control or AutoCAD.
See Online Help for more information.

Error Handling

A message box will be displayed if your AutoLISP code makes an incorrect call and indicate which method was called incorrectly. If a method does not return a value if successful, it will return a –1 to allow for error testing.

For example: the ObjectDCL ComboBox_SelectString call, will return a –1 to indicate that your program has selected a string not available (item #10 was called but only 9 were available).

Calling (Odcl_HideErrorMsgBox) at the beginning of your code will turn off all error message boxes ObjectDCL generates to inform you of programming mistakes. Add this method to your AutoLisp program to make it ready for distribution.

Event Handling

Control initiated events are displayed in a list in the property window under the events tab next to a check box. The events listed for a particular control are those that can call code in your lisp file. Highlighting an event displays this code in a preview window. A button below the window allows you to either copy this code to the clipboard or adds this statement to your lisp file. A user setting in the menu determines this option. The lisp statement created will require further editing to customize it for your use. The statement contains an alert call that should be replaced with your code.

Note that the ObjectDCL Editor constructs the event lisp function name by including the dialog form name as a prefix. This is the name used for the entire dialog form not the name of a control that can be seen as the first item in the property list.

The check box is provided to tell the ObjectDCL Editor whether you want to have the event active. After transferring the code to your lisp file, either by copy or write to file, the check box is automatically checked for your convenience. If you uncheck the box at a later time the event will

no longer make a call to your lisp statement. There is nothing harmful about having an event checked that does not find corresponding lisp code.

The event defun name is remember by the ObjectDCL Editor at the time the event check box is activated. The Editor makes a call to your lisp function that is based on that name. If you change the dialog form name and uncheck and recheck the event box, the event call will use the updated name and not the name that was originally created. The code in your lisp file must match the name that the editor has remembered for that particular event.

For this reason one should be careful when selecting a name for a dialog form. Changing the name at a later date can require catching up many lisp events.

Microsoft's MFC technology (which ObjectDCL is based on) does not preserve the order of fired events. This means that events taking different times to complete may return in an unexpected order. If you code needs the results from one event in order to complete another, the required event should set a flag on completion to notify the dependent routine that it is time to proceed.

The return from ObjectDCL events maybe handled either by the main function as localized variables or outside the main function creating global variables.
The following example shows nesting events for a modal form in the main function:

```
(defun c:Events (/ sVariable)
  (defun c:EventHandling_TextBox1_EditChanged (sText / )
    (setq sVariable sText)
  )
  (defun c:EventHandling_OK_Clicked ()
    (Odcl_Form_Close "Demo" "EventHandling")
  )
  (setq sVariable "")
  (ObjectDCL_LoadArx)
  (Odcl_LoadProject "Demo")
  (Odcl_Form_Show "Demo" "EventHandling")
  ;; AutoLisp will not call this next line in AutoCAD R2000 or 2002 until the user has
  ;;closed the modal dialog box.
  (alert sVariable)
)
```

Note that the AutoLISP function must wait for the completion of the line before alert because the form is modal.

Forcing the Command Bar to take focus

Sometimes for various reasons you may need the AutoCAD command bar to steel focus from any active Modeless or Dockable dialog boxes. To do this a command called Odcl_SetCmdBarFocus has been provided. This command simple informs the command bar to take focus.

Graphic Buttons

Graphic buttons look like the ones seen in the AutoCAD toolbar. It is possible to overdraw the borders of a picture button if the image is the same size or larger than the button.

Hiding and Redisplaying Dialog Boxes in AutoCAD

ObjectDCL remembers the property settings of all controls after they have been closed. Upon reopening, the dialog appears as before. Controls such as the SlideView, ComboBox, ListBox, ListView and Tree controls, are setup by methods rather than properties and require repopulating by your code.

Icons Used in the Titlebar

By default the title bar of each odcl form displays the little blue "a" icon seen in AutoCAD2002. To use your own, look in the property list window for TitleBarIcon. Browse for the icon of choice and see it display in your form. Odcl assigns a number to each icon as it is added to the picture folder attached to the project. To display the same icon in multiple form just select the icon number for that form's title bar property.

Making forms MDI Aware

Each time a different drawing receives focus your code may need to update any displayed ObjectDCL dialog boxes. The event OnDocumentActivated has been added to Dockable and Modeless forms so that you may receive notification from the dialog box that it needs to be updated. Note that the event is not fired when a new drawing is created or a drawing file is opened. With these two cases your lisp code that is auto-loaded should handle the initialization of the dialog box.

Naming Coventions

Using long names for your odcl project file name and form name results in very large defun and symbol names because odcl uses the as prefixes. A short names wll make your code easier to read.

ObjectDCL's PictureBox

The PictureBox can be used to create your own control that you define the way the control looks and feels. For further information on how to draw your own control using the PictureBox, see the Online Help

Objects created while ObjectDCL active

User Comment:
"Yesterday I was testing ObjectDCL with some interactive entities creation and editing without closing (!!!) the ObjectDCL dialog box. My conclusions are that not only can VisualLisp functions be used to do this! Also entities created by entmake and command can be used with ObjectDCL forms (...(redraw) required after entmake to show results on screen, but this is no big problem). Using standard DCL this is impossible."

Option Button Groups

Having more than one frame containing a group of OptionButtons requires that the first option button in each group have its IsTabStop property set to false. Microsoft's MFC uses this property to determine the start of a new group. See Online Help for example.

Transparent GraphicButton

The GraphicButton is transparent by default. This means that almost no special calls are required to activate the transparency of the control. When placing a GraphicButton over a PictureBox or

SliveView control the GraphicButton will auto-detect this and draw it's background by cutting and pasting the below control's image.
See Online help for more information.

VLX Separate Name Space Problem

Question:
I can't get method to be activated from a VLX compile using the separate name space argument, but I can with a lsp or fas file?

Answer:
What is happening is that ObjectDCL creates a global variable to indicate the control being referenced and destroys it once the dialog is closed. This greatly increases the access speed. Since ObjectDCL creates its global variable in the document, you will need to use the VL-DOC-REF function to import the value stored in the global variable. Doing this will make the program work the same as an un-compiled version.

A simple example is shown below.

```
(vl-load-com)
(if (not (member "objectdcl.arx" (arx)))
(arxload "objectdcl.arx" "ObjectDCL.arx not found.")
);end if
; import the ObjectARX functions into this separate name space
(vl-arx-import "objectdcl.arx")
(defun c:test10 ()
(Odcl_LoadProject "Test" T)
(Odcl_Form_Show Test_Test)
);end defun
(defun c:test_TextButton1_OnClicked ()
(Odcl_Control_SetProperty (vl-doc-ref 'test_test_Ctrl1) "Text" "New Text goes here")
;the above line works in a vlx the same way an un-compiled lisp.
);end defun
```

Working in Visual Lisp

There is a menu option under tools where you can check whether you prefer copy to clipboard option or write to lisp file option. The copy to clipboard option is very straight forward and the safest.

If you choose the write to lisp file method, you must create a lisp file that the ObjectDCL Editor will write to. Create a YourNameTemp.lsp file that you can use to collect the statements that you can then cut and paste in to your dialogs true lisp file.

The reason for using a temporary file is that while a visual lisp project file is open in the visual lisp editor, changes to the disk file are detected and a dialog can appear asking if you want to revert to the disk contents of the file. Answering "Yes" to this will cause you to loose any changes you have made to your dialogs lisp file.

FAQ'S

Is it possible to use (Odcl_MultiFileDialog for single selection?

This command is more convenient than creating a Custom file Dialog Box form available in ObjectDCL Pro. Is it available to all versions of ODCL. I does however allow the user to pick more than one filename at a time.

Why do VarNames return NIL after switching drawings?

The VarNames return a NIL once I switch to a new drawing in SDI (single document) mode? Also the OnDocActivated Event does not work in SDI mode for Modeless and Dockable forms.

Answer:

What is happening is that when AutoCAD is in SDI mode the events that indicate the document (Current Drawing) has switched do not activate. This is because in SDI mode at the time the document switch event is received, it too early to activate any Lisp defuns or set the AutoLisp VarName variables. The ObjectdCL.arx checks for the MDI/SDI mode that you have set in the options dialog. If single document mode is set the arx does not attempt to update these variables. Therefore the user must take care of this in the lisp program.

(Odcl_UpdateVarNames)

Is called with no arguments. Place this method at the beginning of your code to update the VarName AutoLisp variables in the current drawing so that they can be used to call a Dialog Box. In multiple document mode, this step is not necessary because the AutoLISP variables are preserved. Including the above function in your code to handle SDI cause no harm if the user has MDI option set.

Another alternative available it to call your dialog boxes using the version 1 format in this format (Odcl_Form_Show "ProjectName" "FormName") instead of the ver 2 format that uses pointer (Odcl_Form_Show ProjectName_FormName) and this method will not need to be called. The controls display much faster especially when updating picture box control slide view control.

Why doesn't my dialog show the values I set with Control_SetProperty?

All your Odcl_Control_SetProperty methods must be within the code you have provided for the OnInitialize event in order to correctly prepare a dialog while it's created but before it is shown. Setting up a dialog's initial value(s) must take place during the OnInitialize event call. The event check box should be checked in the ObjectDCL Editor in the event list for the dialog form. That way ObjectDCL will call the (defun C:form_name_OnInitalize () ...) code you provided or copied to handle that event and run the methods that prepare the dialog.

Why can't I see the changes I made to my dialog form when displayed in AutoCAD?

Make sure there is the symbol "T" in project load statement. Try unloading and reloading the ObjectDCL file using AutoCAD menu selection Tools/Load Application.

Programmers developing on a **network drive** should be aware that there may be problems with loading the most current save for the Editor. This may be do to time differences between the network computer and local computer. It is thought that older files in cache were loaded rather than the most recent save. Also, including the .odc extension with the project name forces a C++ file search rather than the AutoCAD findlile search. Using the odc extension may cause a more reliable update. The existence of the .odc extension will not cause a problem when there is only the .ods file available. When both are present the Editor uses the .odc file. For best results, keep the .odc file on your loacal machine. If there remain problems getting the project to update unload and reload the ObjectDCL.arx file.

Troubleshooting tips

Checking visual lisp symbols created by the ObjectDCL Editor

It is possible for your lisp symbols to become unsynchronized with the ones created by ObjectDCL. Perhaps a change was made to a VarName in the Editor but not updated in your lisp code. Rather than fire up the editor to check what is the problem with a control VarName, is it is much easier to use the Apropos Window in Visual Lisp to inspect the symbols created by the Editor. For example: To see all the symbols defined for a particular dialog form, type in "ProjectName_FormName" and a list will be displayed.

DblClick the form name and see a long integer from VarName that points to the form's memory location. If the control or form has a lisp symbol name that does not match the one defined in the Editor it will show nil.

DblClick one of the controls and see the integer 0. This occurs because controls do not have their VarName's set until the form shows. To see the contents of a control symbol while the form is being shown, place a breakpoint in one of the function calls that can be made by an action in the dialog (pressing button). Evaluation will then stop in the Visual Lisp IDE for you to inspect a forms control symbols in the Apropos Window.

Known Bugs and Limitations

Events become unchecked Fixed ver 2.01

Sometimes when working in the Editor events for a particular form become unchecked. This will cause them not to fire when they should. If this should happen look though all the events for that form and recheck them.

To reproduce this error, show the events for a text button, the OnClicked event should be checked. Uncheck the event so that the copy to clipboard button will work. Press the copy to clipboard button. This should recheck the button. Change focus to the form body. Change focus back to the text button. See that the OnClicked event is no loger checked. What seems to be happening is that the copy to clipboard shows the check mark but save the state change.