Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

The AutoLisp Tutorial - DCL

Dialog Control Language

Ngôn ngữ điều khiển hộp thoại

Contents

Getting Started (Mở đầu)

I'm not going to explain everything there is to know about Dialog Control Language. There are plenty of books on the subject. I will attempt to give you a good understanding of how DCL interacts with AutoLisp. You should be able to write your first DCL driven Autolisp program in no time. Let's get to it.

Tôi không giải thích mọi điều hiểu biết về ngôn ngữ điều khiển hộp thoại. Có rất nhiều tài liệu về mộn học này. Tôi sẽ cố gắng cung cấp cho bạn một kiến thức tốt về cách mà DCL tương tác vớI Autolisp. Bạn sẽ nhanh chóng có thể viết chương trình Autolisp điều khiển DCL đầu tiên của bạn . Nào ta hãy bắt đầu nhé.

Here is the basic order for things to happen inside the AutoLisp file: (Đây là trật tự cơ bản cho mọi thứ xảy ra trong tập tin Autolisp:)

```
(defun C:MyProgram()          ; Define the main program (Định nghĩa chương trình chính)
```

```
(defun myFunction1()        ; Define the functions you need like saveVars (Định nghĩa chương
                              trình mà bạn cần chẳng hạn như saveVar để lưu biến)

  ;do stuff he               ;Thực hiện một số nhiệm vụ nào đó

 )                          ; Kết thúc myFunction1

 (defun myFunction2()       ; (Định nghĩa chương trình mà bạn cần)

   ;do stuff here           ;Thực hiện một số nhiệm vụ nào đó

 )                          ; Kết thúc myFunction2

 (setq list1(list "a" "b" "c" "d"))  ; Build the list if any are required (Tạo một danh
                                       sách nếu được yêu cầu)

 (setq list2(list "1" "2" "3" "4"))  ; (Tạo một danh sách nếu được yêu cầu)

 (setq list3(list "x" "y" "z"))      ; (Tạo một danh sách nếu được yêu cầu)

 load_dialog                         ;Load the DCL file (Tải tập tin DCL)

 new_dialog                          ;Load the DCL definition (Tải định nghĩa DCL)

 start_list                    ;Put the list in the dialog box (Đặt danh sách vào hộp thoại)
 add_list
 end_list

 start_list                    ;As many list as you need (Thêm danh sách nhiều như bạn muốn)
 add_list
 end_list

set_tile                       ; Set the tiles (if necessary) (Đặt các phần tử hộp thoại nếu cần)

 mode_tile                     ; Enable or disable tiles (as necessary) (Kích hoạt hoặc tắt các
                                 phần tử hộp thoại nếu cần thiết)

 ;Set up as many action tiles as you need. (Đặt các phần tử hành động nhiều như bạn muốn)

 (action_tile "accept" "(setq d 2)(myFunction1)(done_dialog)")
 (action_tile "cancel" "(setq d 1)(myFunction2)(done_dialog)")

 ;Two action tiles need to have the done_dialog call. One of these needs to save the settings before
 ; the done_dialog call.

 ; Hai phần tử hành động cần phải có để gọi hộp thoại hành động. Một trong chúng là để lưu việc đặt lại
```

```
    ;trước khi gọi hộp thoại hành động

    start_dialog                ; Start the dialog box  (Bắt đầu hộp thoại)

    unload_dialog               ; Unload the dialog box (Thoát khỏi hộp thoại)

   ; Do stuff if the user presses the cancel key  (Thực hiện một số nhiệm vụ nào đó nếu người sử dụng
                                                   nhấn nút Cancel)

    (if (= d 1)

      ;do stuff here if cancel was pressed (Thực hiện một số nhiệm vụ nào đó nếu
                                            người sử dụng nhấn nút Cancel)

     )

    ; Do stuff if the user presses the accept key (Thực hiện một số nhiệm vụ nào đó nếu người sử dụng
                                                   nhấn nút Accept)

    (if (= d 2)

      ;do stuff here if okay was pressed (Thực hiện một số nhiệm vụ nào đó nếu người
                                          sử dụng nhấn nút OK)

     )

)                                                  ; close the program (Đóng chương trình)
```

We will discuss this in detail later on. (Chúng ta sẽ thảo luận chi tiết vấn đề này ở phần sau)

I've found the hardest thing to get a grip on is laying out the dialog box. Please take a look at the rows and columns section if you need help with laying out a dialog box. For now let's look at the basics. *I'm going to throw a lot at you, but don't panic. I'll cover them a little closer later on.* We will cover these items first: button, row, column, boxed_row, and boxed_column.

Tôi phát hiện điều khó nhất để nắm vững là bố trí hộp thoại. Hãy xem phần các hàng và các cột khi bạn cần giúp đỡ việc bố trí một hộp thoại. Bây giờ ta sẽ xem xét phần cơ bản. *Tôi sắp quăng cho bạn một lô xích xông các vấn đề, nhưng đừng có run nhé. Tôi sẽ giải quyết chúngtừng chút một ở phần sau.* Chúng ta hãy bắt đầu với các mục Nút, hàng, cột, hàng hộp và cột hộp

Okay and Cancel Buttons - The DCL code for a these buttons look like this: (Mã DCL cho các nút này là:)

```
: button {                  \\ Define the button  (Định nghĩa nút)

  key = "accept";           \\ Define the action key (This is my name assigned to this button)
                               (Định nghĩa khóa hành động. Đây là tên của tôi gán cho nút này)
```

```
    label = " Okay ";          \\ This is the text displayed on the button. (Đây là chữ hiển thị trên nút)

    is_default = true;         \\ Allows the button to activate when the user presses the enter key. (Cho phép kích hoạt khi người sử dụng nhấn Enter)

  }                            \\ Close the button definition (Đóng định nghĩa nút)


  : button {                   \\ Define another button (Định nghĩa một nút khác)

    key = "cancel";            \\ Define the action key. (I chose this name for the button.) (Định nghĩa khóa hành động. Tôi chọn tên này cho nút mới)

    label = " Cancel ";        \\ Text displayed on the button. (Chữ hiển thị trên nút)

    is_default = false;        \\ Does not allow this key to activate when the enter key is pressed. ( Không cho phép khóa này kích hoạt khi nhấn Enter)

    is_cancel = true;          \\ Activate this key when the close button [ X ] is selected. (Kích hoạt khóa này khi nút Close [X] được chọn)

  }                            \\ Close the button definition  (Đóng định nghĩa nút này)
```

Rows and Columns designate areas to lay your controls on. [Controls are list boxes, edit boxes, buttons, etc.]. (là các vùng được thiết kế để đặt các nút điều khiển của bạn.[Nút điều khiển là các hộp danh sách, hộp chỉnh sửa, các nút, vv…vv])

Here is the DCL code for a column: (Đây là mã DCL cho cột)

```
: column {

}
```

Here is the DCL code for a row: (Đây là mã DCL cho hàng)

```
: row {

}
```

Simple right? Thật đơn giản, phải không?

Here is the code for a column with two buttons: Đây là mã cho một cột với hai nút:

```
: column {

  : button {
    key = "accept";
    label = " Okay ";
    is_default = true;
  }

  : button {
    key = "cancel";
    label = " Cancel ";
    is_default = false;
    is_cancel = true;
  }

}
```

Notice the buttons are stacked on top of each other.

(Chú ý là các nút chồng lên nhau)

Here is the code for a row with two buttons: Đây là mã cho một hàng với hai nút:

```
: row {

  : button {
    key = "accept";
    label = " Okay ";
    is_default = true;
  }

  : button {
    key = "cancel";
    label = " Cancel ";
    is_default = false;
    is_cancel = true;
  }

}
```

Notice the buttons are side by side.

Chú ý là các nút nằm cạnh nhau

Let's turn the above into Boxed Rows and Boxed Columns. Here is the code for a boxed_column with two buttons: Quay trở lại với các hàng hộp và cột hộp đã nói ở trên. Đây là mã cho một cột hộp với hai nút:

```
: boxed_column {

  : button {
    key = "accept";
    label = " Okay ";
    is_default = true;
  }

  : button {
    key = "cancel";
    label = " Cancel ";
    is_default = false;
    is_cancel = true;
  }

}
```

Notice the box around the buttons.

Chú ý tới hộp xung quanh các nút

Here is the code for a boxed_row with two buttons: Đây là mã cho một hàng hộp với hai nút:

```
: boxed_row {

  : button {
    key = "accept";
    label = " Okay ";
    is_default = true;
  }

  : button {
    key = "cancel";
    label = " Cancel ";
    is_default = false;
    is_cancel = true;
  }

}
```

Notice the box around the buttons.

Chú ý tới hộp xung quanh các nút

Now we know the difference between a row and a boxed_row. We also know that controls inside a column get stacked on top of each other [vertical] and controls inside a row are next to each other [horizontal].

Giờ ta đã biết sự khác nhau giữa hàng và hàng hộp. Ta cũng biết rằng các nút điều khiển trong một cột thì xếp chồng lên nhau [thẳng đứng], còn trong một hàng thì xếp nằm cạnh nhau.[nằm ngang]

Important: You should never ever attempt to build a dialog box without a cancel button. Trust me. *Or you can do what I did and find out for yourself.*

Chú ý quan trọng: Đừng bao giờ cố gắng tạo một hộp thoại không có nút Cancel. Hãy tin tôi đi. *Hoặc là bạn có thể cứ làm điều tôi đã làm và tự tìm ra kết luận của bạn.*

Overview (Tổng quan)

Let's take a look at the entire code for the dialog box above, including the LSP file: (Ta hãy xem xét mã hoàn chỉnh cho hộp thoại trên bao gồm cả tập tin Autolisp)

DCL FILE NAMED: DCL_TEST.DCL AUTOLISP PROGRAM NAMED: DCL_TEST.lsp

TÊN TẬP TIN DCL: DCL_TEST.DCL TÊN CHƯƠNG TRÌNH AUTOLISP: DCL_TEST.lsp

```
DCL_TEST : dialog {

  : boxed_row {

    : button {
      key = "accept";
      label = " Okay ";
      is_default = true;
    }

    : button {
      key = "cancel";
      label = " Cancel ";
      is_default = false;
      is_cancel = true;
    }

  }

}
```

```
(defun C:DCL_TEST()
  (setq dcl_id (load_dialog "DCL_TEST.dcl"))
  (if (not (new_dialog "DCL_TEST" dcl_id) )
(exit))
  (action_tile "cancel" "(setq
ddiag1)(done_dialog)")
  (action_tile "accept" "(setq
ddiag2)(done_dialog)")
  (start_dialog)
  (unload_dialog dcl_id)
  (if (= ddiag 1)
    (princ "\n \n ...DCL_TEST Cancelled. \n
")
  )
  (if (= ddiag 2)
    (alert "You pressed the OKAY button!")
  )
)
```

AutoLISP

Let's take a look at the AutoLisp part of this simple program. (Ta hãy xem xét phần Autolisp của chương trình đơn giản này)

First we defined our program: (Trước hết ta xác định chương trình của mình)

```
(defun C:DCL_TEST()
```

The second thing was to load the DCL file from disk. Insert a path if necessary, but I always liked to use the autocad search path to find the file. Notice the dcl_id variable used. We will need this later on. This identifies the file opened.

Thứ hai là tải tập tin DCL từ ổ đĩa. Đưa vào đường dẫn nếu cần, mà tôi luôn thích sử dụng đường tìm kiếm của AutoCad để tìm tập tin đó.Chú ý rằng ta sử dụng biến dcl_id mà ta sẽ cần sau này. Điều này cũng như tập tin được mở.

```
;;;--- Put up the dialog box (Đặt hộp thoại)
  (setq dcl_id (load_dialog "DCL_TEST.dcl"))
```

Since a DCL file can contain multiple dialog box definitions, [I'll show you an example later on.] we need to specify which dialog box we want to load. This is taken care of in the next statement. Notice the name matches the definition inside the DCL file.

Do một tập tin DCL có thể chứa nhiều định nghĩa hộp thoại [tôi sẽ chỉ cho bạn một ví dụ ở phần sau], chúng ta cần mô tả hộp thoại nào mà ta muốn tải. Điều này được bảo đảm trong thông báo tiếp theo. Chú ý tới cái tên trùng với định nghĩa trong tập tin DCL

```
  ;;;--- See if it is already loaded (Kiểm tra xem nó đã được tải chưa)
  (if (not (new_dialog "DCL_TEST" dcl_id) ) (exit))
```

Next, we define our action keys. These tell the program what to do whenever a user presses a button, selects an item in a list, etc. Notice the two action keys. "cancel" and "accept" . These match the keys in the DCL file. I made these names up. You can name them whatever you want as long as it is not a name used by AutoCAD and it only appears once in each DCL definition. You can't have two controls with the same key, the program would get confused. So would I!!! Whenever the user presses the cancel key, the program sets variable ddiag to 1 and closes the dialog box. Upon pressing the Okay button, [key = accept], the program sets variable ddiag to 2 and closes the dialog box. That's it. Nothing else to add except, you could have just about anything in here, including instructions to execute a function [I'll demonstrate that later.]

Tiếp theo, ta xác định các khóa hành động của chúng ta. Điều này sẽ bảo chương trình điều phải làm bất cứ khi nào người sử dụng nhấn một nút, chọn một khoản mục trong một danh sách, vvv….. Chú ý tới hai khóa hành động "Cancel" và "Accept". Chúng trùng với các khóa trong tập tin DCL. Tôi đã tạo các tên này. Bạn có thể đặt bất cứ tên gì bạn thích, dài thoải mái mà không phải các tên đã được AutoCad sử dụng và nó chỉ xuất hiện một lần trong mỗi định nghĩa của DCL. Bạn không thể có hai khả năng điều khiển cho cùng một khóa, chương trình sẽ bị rối. Tôi đã bị như vậy!!!. Bất cứ khi nào người sử dụng nhấn khóa Cancel, chương trình sẽ đặt biến ddiag về 1 và đóng hộp thoại. Áp dụng việc nhấn nút Okay [khoá là Accept], chương trình sẽ đặt biến ddiag thành 2 và đóng hộp thoại. Vậy đó. Chẳng có gì khác được thêm vào trừ phi bạn phải có cái gì đó ở đây, bao gồm các chỉ dẫn để thực hiện một hàm [Tôi sẽ biểu diễn điều đó sau]

```
  ;;;--- If an action event occurs, do this function (Nếu một sự hành động xảy ra, thực hiện hàm sau)
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(done_dialog)")
```

Finally, the big step. Everything is defined, let's display the dialog box.(Cuối cùng, một bước vĩ đại. Mọi thứ đã kết thúc, ta hãy hiển thị hộp thoại)

```
  ;;;--- Display the dialog box (Hiển thị hộp thoại)
  (start_dialog)
```

The program halts here until the dialog box is issued a "done_dialog" call. In the mean time the user is interacting with the action keys. When the user presses a button the program kills the dialog box with the unload_dialog call. Notice the dcl_id passed as a parameter. You could have more than one dialog file open at one time. So, the unload_dialog function needs to know which one to unload.

Chương trình sẽ treo ở đây cho tới khi hộp thoại có một thao tác gọi "Hộpthoại_hànhđộng".Lúc đó người

sử dụng phải có tương tác với các khoá hành động. Khi người sử dụng nhấn một nút, chương trình sẽ hủy hộp thoại với việc gọi lệnh hộp thoại thoát tải (unload_dialog). Chú ý rằng biến dcl_id thoát ra như một tham số. Bạn có thể có hơn một tập tin hộp thoại mở trong một lần. Vì thế hàm lệnh unload_dialog cần phải biết thoát khỏi hộp thoại nào.

```
 ;;;--- Display the dialog box (Hiển thị hộp thoại)
(unload_dialog dcl_id)
```

```
  Finally the dialog box is gone.  What next?   Find out which button the user pressed?
Exactly!  Remember setting the ddiag variable to either 1 or 2.  We will use that now to
determine which button was pressed.  There is a better way to do this, I'll show you
later.
```

Hộp thoại đã kết thúc xong. Điều gì kế tiếp? Hãy tìm xem nút nào đã được người sử dụng nhấn? Chính xác?Hãy nhớ tới việc đặt biến ddiag về giá trị 1 hoặc 2. Ta sẽ sử dụng chúng để xác định xem nút nào được nhấn. *Có một cách tốt hơn để làm điều này, tôi sẽ chỉ cho bạn sau.*

```
;;;--- If the cancel button was pressed - display message (Nếu nút Cancel bị nhấn, hiển thị)
(if (= ddiag 1)
  (princ "\n \n ...DCL_LSP Cancelled. \n ")
)

;;;--- If the "Create" button was pressed (Nếu nút Create bị nhấn)
(if (= ddiag 2)
  (alert "You pressed the OKAY button!")
)

And the last step, close the autolisp program.(và bước cuối cùng, đóng chương trình Autolisp)
)
```

The basic DCL model (Một mẫu tập tin DCL cơ bản)

I will now give you the model I use for 98% of all dialog based programs I write. I start with this basic program and build from there. It is basically the same thing we've gone over before. I put up a dialog box. I allow the user to press buttons and manipulate data. The only difference is, when the user presses the OKAY button I save the settings in the dialog file before it closes. I do this inside the action key call with a function called saveVars. Without further ado, here's the model:

Bây giờ tôi sẽ cung cấp cho bạn một mẫu mà tôi sử dụng cho 98% các chương trình chứa hộp thoại mà tôi viết. Tôi bắt đầu với chương trình cơ bản này và tạo dựng tiếp từ đó. Điều đó cũng như điều tương tự chúng ta đã từng làm trước đấy. Tôi đặt một hộp thoại. cho phép người sử dụng nhấn các nút và thao tác với các dữ liệu. Điều khác duy nhất là khi người sử dụng nhấn nút Okay tôi sẽ lưu lại những điều đặt trong tập tin hộp thoại trước khi đóng nó. Tôi thực hiện điều này trong khóa hành động kèm theo hàm được gọi là hàm saveVar . Ngoài ra chẳng còn gì khác nữa, và đây là mẫu đó.

The DCL File: Tập tin DCL

```
EXAMPLE : dialog {
```

```
        label = "EXAMPLE.lsp";     \\ Puts a label on the dialog box (Đặt nhãn cho hộp thoại)
        initial_focus = "textval";   \\ Sets the initial focus (Đặt mục tiêu ban đầu)
        : column {
          : row {
            : boxed_column {

              : edit_box {        \\ The edit_box control - Something new! (Cái này mới nè:
                                     Nút điều khiển hộp thoại hiệu chỉnh)

                key = "textval";
                label = "Text Value to Find";
                edit_width = 30;
                value = "";
              }
            }
          }
          : row {
            : boxed_row {
              : button {
                key = "accept";
                label = " Okay ";
                is_default = true;
              }
              : button {
                key = "cancel";
                label = " Cancel ";
                is_default = false;
                is_cancel = true;
              }
            }
          }
        }
}
```

The AutoLisp File: (Tập tin Autolisp)

```
;;;--- EXAMPLE.lsp - Text Find (Tìm dòng văn bản)

(defun saveVars()
  ;;;--- Save the input from the dialog box (Lưu lại đầu vào của hộp thoại)
  (setq textVal (get_tile "textval"))
)


(defun C:EXAMPLE()

  ;;;--- Load the dcl file ( Tải tập tin DCL)
  (setq dcl_id (load_dialog "EXAMPLE.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded (Tải định
```

nghĩa hộp thoại nếu nó chưa được tải)

```
  (if (not (new_dialog "EXAMPLE" dcl_id) ) (exit))

  ;;;--- If an action event occurs, do this function (Nếu một sự hành động xảy ra, thực hiện hàm này)
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")

  ;;;--- Display the dialog box (Hiển thị hộp thoại)
  (start_dialog)

  ;;;--- If the cancel button was pressed - display message (Hiển thị nếu nút Cancel được nhấn)
  (if (= ddiag 1)
    (princ "\n \n ...EXAMPLE Cancelled. \n ")
  )

  ;;;--- If the "Okay" button was pressed (Nếu nút Okay được nhấn)
  (if (= ddiag 2)
    (princ "\n \n ...Example Complete!")
  )

  ;;;--- Suppress the last echo for a clean exit (Triệt tiêu việc lặp lại thao tác cuối và thoát êm)
  (princ)

)
```

Screen Shot: Hiển thị trên màn hình:

If you could not follow some of the autolisp functions, you can ignore them for now or read the Autolisp Tutorial. (Nếu bạn không hiểu chỗ nào trong chương trình Autolisp trên, hãy đọc lại phần tự học Autolisp)

This is the model. Very little will change from this setup. You may add some controls to the DCL file which in turn will add Action calls and influence the number of lines inside the saveVars routine. The only other thing that will change is inside the OKAY button function. [When ddiag = 2]. You will have to tell the program what to do when all of the information is gathered. We will cover these later.

Đây là một mẫu . Có rất ít thay đổi so với ban đầu. Bạn có thể thêm vài nút điều khiển vào tập tin DCL mà nó sẽ thêm vào các việc gọi lệnh hành động và ảnh hưởng đến số dòng bên trong của việc lưu biến saveVar. Điều khác duy nhất là có sự thay đổi bên trong chưc năng của nút Okay.[Khi ddiag = 2]. Bạn phải bảo

chương trình điều nó cần làm khi tất cả các thông tin được nhóm lại. Ta sẽ học những điều này sau.

Back

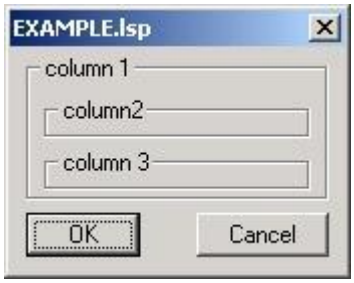
Rows and Columns (Các hàng và các cột)

Let's get a feel for laying out a dialog box using nested rows and columns. I'll do this by showing you the DCL code next to a picture of the dialog box. I can't think of any easier way to do it. *Ignore all of the Labels and TEXT controls. I will use these to visually show the location of things.*

Ta hãy hình dung sự bố trí một hộp thoại sử dụng các tổ hợp hàng và cột. Tôi sẽ thực hiện điều này bằng cách trình bày cho bạn mã DCL bên cạnh hình thể hiện của hộp thoại. Tôi không thể nghĩ ra cách nào dễ dàng hơn cách này. Hãy quên đi tất cả các nhãn và các text điều khiển. Tôi sử dụng chúng chỉ để thể hiện cái chỗ đặt chúng mà thôi.

Example 1:

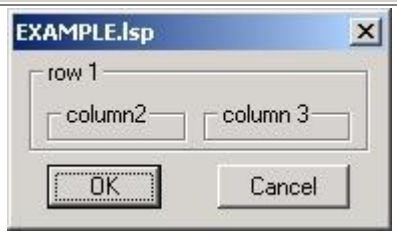
```
EXAMPLE : dialog {
  label = "EXAMPLE.lsp";
  : column {
    : boxed_column {
      label = "column 1";
      : boxed_column {
        label = "column2";
      }
      : boxed_column {
        label = "column 3";
      }
    }
    ok_cancel;
  }
}
```



Column (Một cột chính bao gồm)
  Column (Một cột hộp 1 chứa)
  Column<\p> (Các cột hộp bên trong 2,3)

  Tổ hợp hàng gồm hai nút OK và Cancel

```
EXAMPLE : dialog {
  label = "EXAMPLE.lsp";
  : column {
    : boxed_row {
      label = "row 1";
      : boxed_column {
        label = "column2";
      }
      : boxed_column {
        label = "column 3";
      }
    }
    ok_cancel;
  }
}
```



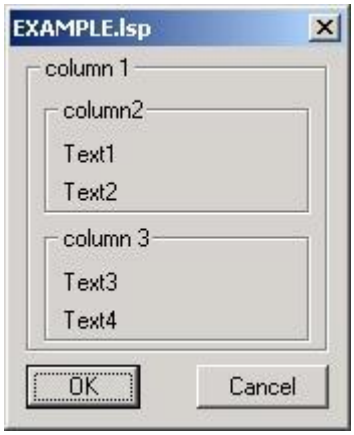
Notice I changed the first boxed column into a boxed row. (Chú ý sự thay đổi cột hộp thứ nhất thành hàng hộp)

Cột chính bao gồm:

Row (Hàng hộp 1 chứa)
  Column (Cột hộp 2 bên trong)
  Column (Cột hộp 3 bên trong)

Hàng chứa hai nút OK và Cancel

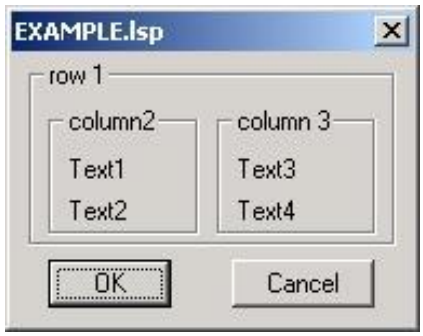
```
EXAMPLE : dialog {
  label = "EXAMPLE.lsp";
  : column {
    : boxed_column {
      label = "column 1";
      : boxed_column {
        label = "column2";
: text {
          key = "t1";
          value = "Text1";
        }
        : text {
          key = "t2";
          value = "Text2";
        }
      }
      : boxed_column {
        label = "column 3";
         : text {
          key = "t3";
          value = "Text3";
        }
        : text {
          key = "t4";
          value = "Text4";
        }
      }
    }
    ok_cancel;
  }
}
```



Column (Cột chính bao gồm)
  Column (Cột hộp 1 chứa)
  Column<\p> (Các cột hộp bên trong 2,3 chứa các khoá t1, t2 và các khóa t3,t4)

Hàng hai nút OK và Cancel

```
EXAMPLE : dialog {
  label = "EXAMPLE.lsp";
  : column {
    : boxed_row {
      label = "row 1";
      : boxed_column {
        label = "column2";
: text {
          key = "t1";
          value = "Text1";
        }
        : text {
          key = "t2";
          value = "Text2";
        }
      }
      : boxed_column {
        label = "column 3";
         : text {
          key = "t3";
          value = "Text3";
        }
        : text {
          key = "t4";
          value = "Text4";
        }
      }
    }
    ok_cancel;
  }
}
```

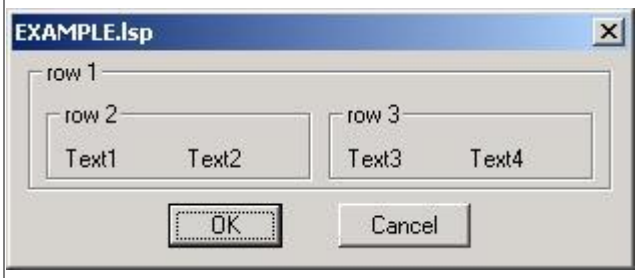


Cột chính bao gồm

Row (Hàng hộp 1 chứa hai cột hộp 2,3)
  Column (Cột hộp 2 chứa hai khoá t1, t2)
  Column (Cột hộp 3 chứa hai khóa t3,t4)

Hàng chứa hai nút OK và Cancel

```
EXAMPLE : dialog {
  label = "EXAMPLE.lsp";
  : column {
    : boxed_row {
      label = "row 1";
      : boxed_row {
        label = "row 2";
       : text {
          key = "t1";
          value = "Text1";
        }
        : text {
          key = "t2";
          value = "Text2";
        }
      }
      : boxed_row {
        label = "row 3";
         : text {
          key = "t3";
          value = "Text3";
        }
        : text {
          key = "t4";
          value = "Text4";
        }
      }
    }
    ok_cancel;
  }
}
```

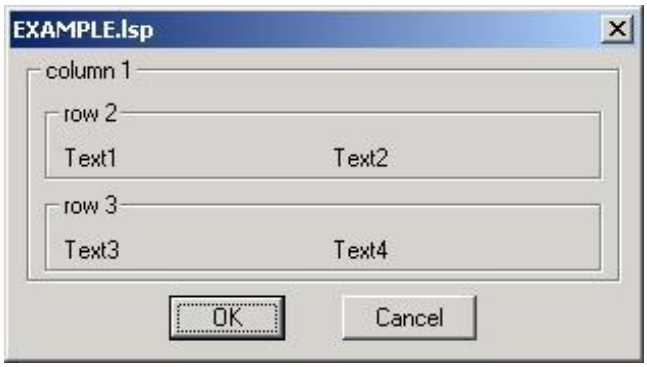


Cột chính bao gồm:

Row (Hàng hộp 1 chứa các hàng hộp 2,3)
  Row (Hàng hộp 2 chứa các khóa t1,t2)
  Row Hàng hộp 3 chứa các khoá t3,t4)

Hàng chứa hai núT OK va Cancel

```
EXAMPLE : dialog {
  label = "EXAMPLE.lsp";
  : column {
    : boxed_column {
      label = "row 1";
      : boxed_row {
        label = "row 2";
       : text {
          key = "t1";
          value = "Text1";
        }
        : text {
          key = "t2";
          value = "Text2";
        }
      }
      : boxed_row {
        label = "row 3";
         : text {
          key = "t3";
          value = "Text3";
        }
        : text {
          key = "t4";
          value = "Text4";
        }
      }
    }
    ok_cancel;
  }
}
```



Cột chính bao gồm:

Column (Cột hộp 1 chứa các hàng hộp 2,3)
Row (Hàng hộp 2 chứa các khóa t1, t2)
Row (Hàng hộp 3 chứa các khoá t3, t4)

Hàng chứa hai nút OK và Cancel

I hope this puts an end to the rows and columns issue. If not, you know where to find me. (Tôi hy vọng đến đây có thể kết thúc cái món hàng và cột này. Nếu không chắc bạn biết tìm tôi ở đâu rồi.

Back

Dialog Control Language–Controls (Các nút điều khiển trong DCL)

Let's take a look at the different controls you can put on the dialog box. You've seen a few of them in the overview portion but let's go over the majority of them now. I will only show the most used properties for each control.

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

Ta hãy đi kiếm các nút điều khiển khác nhau mà bạn có thể đặt vào hộp thoại. Bạn đã từng ngó thấy chúng trong phần tổng quan nhưng bây giờ ta sẽ tập trung ngâm cứu chúng. Tôi sẽ chỉ trình bày những thuộc tính sử dụng chính của mỗi nút .

LAYOUT CONTROLS:

```
: column {
  label = "My Column";
}
```

Column (Cột)

You can omit the label. Bạn có thể bỏ qua việc gắn nhãn

```
: boxed_column {
  label = "My Column";
}
```

Boxed Column (Cột hộp)

You can omit the label. Bạn có thể bỏ qua việc gắn nhãn

```
: row {
  label = "My Row";
}
```

Row (Hàng)

You can omit the label. Bạn có thể bỏ qua việc gắn nhãn

```
: boxed_row {
  label = "My Row";
}
```

Boxed Row (Hàng hộp)

You can omit the label. Bạn có thể bỏ qua việc gắn nhãn

```
: spacer {
  width = 10;
  height = 15;
}
```

A normal spacer to help align other controls. You can omit the width and height properties. (Một khoảng không gian bình thường để giúp cho việc căn chỉnh vị trí của nút điều khiển khác. Bạn có thể bỏ qua thuộc tính độ rộng và chiều cao)

```
: spacer_0;
```

A spacer that usually has no width. It simply means, if you have to stretch something in this row to make things fit, stretch this spacer instead of the other items.. (Một không gian có độ rộng luôn bằng 0. Nó đơn giản có nghĩa là nếu bạn phải duỗi dài nút nào đó trên hàng này để tạo một sự hài hoà thì hãy dãn không gian này thay cho các không gian khác)

```
: spacer_1;
```

The smallest spacer that's visually noticable. (Không gian nhỏ nhất có thể thấy được)

BUTTON (NÚT)

```
: button {
  key = "button1;
  label = "Okay";
  is_tab_stop = true;
  is_cancel = false;
  is_default = true;
  width = 15;
  height = 10;
  mnemonic = "B";
}
```

key = The name you assign to the button. (Tên bạn gán cho nút)

label = The text displayed on the button. Dòng chữ thể hiện trên nút)

is_cancel = Determines if this is the cancel button. One control must be assigned to be the cancel action. (Xác định đây có phải là nút xóa không. Một quá trình điều khiển được gán cho hành động xoá này)

is_default = Determines if this button activates when the user presses the enter key. (Xác định nút này có bị kích hoạt khi người sử dụng nhấn khóa Enter hay không)

mnemonic = Determines if you can press ALT+B to move to this action key. You assign the letter and DCL will underscore the letter on the dialog box to let the user know it is ALT-able. *If that is a word!* (Xác định bạn có thể sử dụng tổ hợp phím Alt +B để chuyển tới khoá hành động này hay không. Bạn gán chữ cái và DCL sẽ gạch đít chữ cái đó trong hộp thoại để báo cho người sử dụng biết nó có khả năng kết hợp với phím Alt. *Nếu đó là một từ)*

BOXED RADIO COLUMN, RADIO COLUMN, & RADIO BUTTON

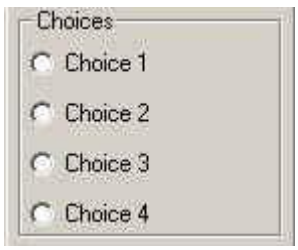
CỘT LỰA CHỌN HỘP, CỘT LỰA CHỌN, VÀ NÚT LỰA CHỌN

Cái tên này tớ tự bịa ra vì cóc biết nêndịch nó là gì, thôi thì cứ cho nó một cái tên Tí Tèo như vậy để mà đọc tiếp. Ai có cái tên hay hơn, đúng hơn thì chỉ giùm tớ, xin cám ơn trước nhé.

```
: radio_column {
  label = "Choices";
  key = "choices";

  : radio_button {
```

// Use boxed_radio_column if a box is required. (Sử dụng cột lựa chọn hộp nếu một hộp được yêu cầu)
// Label for the column or boxed_column (Gắn nhãn cho cột hay cột hộp)
// Action key for the radio column (Khóa hành động cho cột lựa chọn)

// First radio button (Nút lựa chọn thứ nhất)



```
    label = "Choice 1";
    key = "choice1";
  }
  : radio_button {
    label = "Choice 2";
    key = "choice2";
  }
  : radio_button {
    label = "Choice 3";
    key = "choice3";
  }
  : radio_button {
    label = "Choice 4";
    key = "choice4";
  }
}
```

// Second radio button (Nút lựa chọn thứ hai)

// Third radio button (Nút lựa chọn thứ ba)

// Fourth radio button (Nút lựa chọn thứ tư)

// Close the radio_column (Đóng cột lựa chọn)

BOXED RADIO ROW, RADIO ROW, & RADIO BUTTON

HÀNG LỰA CHỌN HỘP, HÀNG LỰA CHỌN, VÀ NÚT LỰA CHỌN

```
: radio_row {
  label = "Choices";
  key = "choices";

  : radio_button {
    label = "1";
    key = "choice1";
  }
  : radio_button {
    label = "2";
    key = "choice2";
  }
  : radio_button {
    label = "3";
    key = "choice3";
  }
  : radio_button {
    label = "4";
    key = "choice4";
  }
```

// Use boxed_radio_row for box. (Sử dụng hàng lựa chọn hộp cho một hộp)
// Label for the row or boxed_row (Gắn nhãn cho hàng hay hàng hộp)
// Action key for the radio row (Khóa hành động cho hàng lựa chọn)

// First radio button (Nút lựa chọn thứ nhất)

// Second radio button (Nút lựa chọn thứ hai)

// Third radio button (Nút lựa chọn thứ ba)

// Fourth radio button (Nút lựa chọn thứ tư)

```
}
```

// Close the radio_row (Đóng hàng lựa chọn)

EDIT BOX (HỘP HIỆU CHỈNH)

```
: edit_box {
  key = "myval";
  label = "Value:";
  edit_width = 10;
  value = "";
}
```

// Action key (Khóa hành động)
// Label for the edit box (Nhãn gắn cho hộp hiệu chỉnh)
// Character width (Độ rộng ký tự)
// Initial value (Giá trị ban đầu)

LIST BOX (HỘP DANH SÁCH)

```
: list_box {
  label ="Choose Items";
  key = "mylist";
  height = 15;
  width = 25;
  multiple_select = true;
  fixed_width_font = true;
  value = "0";
}
```

// Label for the list box (Nhãn cho hộp danh sách)
// Action key (Khoá hành động)
// Height of list box (Độ cao hộp)
// Width of list box (Độ rộng hộp)
// Multiple Select = TRUE (nhiều lựa chọn)
// Use fixed with font = TRUE (Sử dụng font chữ đậm cố định)
// Initial selection = 1st item (Lựa chọn ban đầu là khoản mục đầu tiên)

Fixed Width Font = TRUE

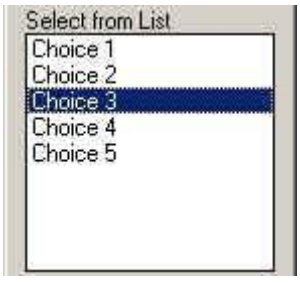
Multiple Select = TRUE

View of list box with... (Hình hiển thị hộp danh sách với:)

Fixed Width Font = FALSE. (Font chữ mảnh)

Multiple Select = TRUE. (Cho chọn nhiều khoản mục)

View of list box with... (Hình hiển thị hộp danh sách với:)

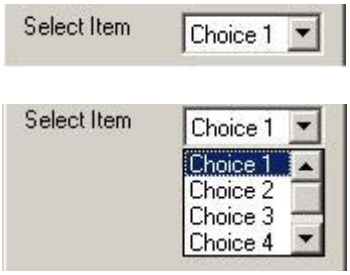


Fixed Width Font = FALSE. (Font chữ mảnh)

Multiple Select = FALSE. (Không cho chọn nhiều khoản mục)

POPUP LIST (DANH SÁCH THẢ XUỐNG)

```
: popup_list {                    // Action key (Khoá hành động)
  key = "mylist";                 // Label  (nhãn gắn)
  label = "Select Item";          // Use fixed with font = false (Không
  fixed_width_font =              sử dụng font chữ đậm)
false;                            // Width of list box  (Độ rộng hộp
  width = 15;                     danh sách)
  value = 0;                      // Initial selection = 1st item (Lựa
}                                 chọn ban đầu là khoản mục thứ nhất)
```



TEXT (VĂN BẢN)

```
: text {
  key = "mytext";                 // Action key (Khóa hành động)
  value = "This is text!";        // Value  (Giá trị văn bản)
}
```

Okay. That's it for controls. There are others but, these are the most used controls. Let's move on.(OK. Đó là các nút điều khiển. Còn có các nút khác nhưng đây là những nút sử dụng chủ yếu. Ta sang mục khác nhé.

Back

Dialog Control Language – Image (Hình ảnh trong DCL)

Image (Các hình ảnh)

Image (Hình ảnh)

An Image is used to display a vector graphic picture inside a rectangle. To create the image you can use three different methods. (Một hình ảnh được sử dụng để hiển thị một bứchình hoạ đồ vector trong một khung chữ nhật. Để tạo một hình ảnh bạn có thể sử dụng 3 phương pháp khác nhau)

- Vector_image - Draws a single straight line in the current image. (Ảnh Vector - Vẽ một đoạn thẳng trong ảnh hiện tại)
- Fill_image - Draws a filled rectangle in the current image.(Ảnh đặc- Vẽ một khung chữ nhật đặc trong ảnh hiện tại)
- Slide_image - Draws an AutoCAD slide in the image. (Ảnh Silse- Vẽ một ảnh AutoCad Slide trong ảnh hiện tại)

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

For this tutorial we will concentrate on the Slide_Image function. Everyone knows how to create a slide right? The MSLIDE command. Okay. We will assume our image is ready.

Để học điều này ta sẽ tập trung vào hàm ảnh Slide. Mọi người đã biết cách tạo một ảnh Slide rồi, đúng không? Lệnh Mslide nữa. RồI Ok. Ta giả sử rằng ta đã có ảnh sẵn sàng rồi.

Let's look at the DCL Image definition and the AutoLisp functions required to display the image. (Hãy xem định nghĩa hình ảnh trong DCL và các hàm Autolisp càn có để hiển thị hình ảnh này.)

DCL - You must supply either the width and height or one of these plus an aspect_ratio. (Bạn phải áp dụng hoặc là độ cao và độ rộng hoặc là một trong chúng và một tỉ lệ ảnh)

```
: image {
  key = "sld";
  height = 30;
  width = 30;
  color = 0;
  is_enabled = false;
  is_tab_stop = false;
}
```

AutoLisp

```
;;;--- First we need a slide name  (Trước hết bạn cần đặt tên ảnh silde)


(setq mySlideName "c:/acad/myslide.sld")

;;;--- Second we need the key to the image control (Thứ hai bạn cần đặt khoá điều khiển hình ảnh)
(setq myKey "sld")

;;;--- Next we send the slide name and the key to the update function  (Tiếp theo ta gửi tên và khóa này vào hàm update hình ảnh)
(upDateImage mySlideName myKey)

; NOTE: Notice mySlideName becomes sldName and myKey becomes key when passed as
; parameters to the upDateImage function. ( Chú ý:Biến mySlideName trở thành sdlName và biến myKey trở thành key khi đi qua hàm update hình ảnh như các tham số)

;;;--- Function to update the slide (Hàm update hình ảnh Slide)

(defun upDateImage(sldName key)

  ;;;--- Get the width of the slide (Lấy độ rộng của hình ảnh)
  (setq width (dimx_tile key))

  ;;;--- Get the height of the slide (Lấy độ cao của hình ảnh)
  (setq height (dimy_tile key))
```

```
  ;;;--- Start the slide definition (Bắt đầu định nghĩa ảnh slide)
  (start_image key)

  ;;;--- Wipe out the background (Xóa sạch nền)
  (fill_image 0 0 width height 0)

  ;;;--- Put the slide in the image area (Đặt ảnh Slide vào vùng ảnh)
  (slide_image 0 0 width height sldName)

  ;;;--- Finish the slide definition (Kết thúc định nghĩa ảnh slide)
  (end_image)

)
```

This function can be used over and over to display any slide image in an image control. Use the upDateImage function after the new_dialog call and before the action_tile statements. You can also use this function while the dialog box is displayed in one of your action calls. We will use this later on in the tutorial so you can see it in action.

Hàm này có thể sử dụng lần lượt để hiển thị bất kỳ ảnh slide nào trong một khóa điều khiển. Việc sử dụng hàm update hình ảnh sau khi gọi hộp thoại mới và trước các thông báo của phần tử hành động. Bạn cũng có thể sử dụng hàm này trong khi hộp thoại hiển thị trong một lần gọi hành động của bạn. Ta sẽ sử dụng điều này trong phần sau và bạn có thể quan sát nó làm việc.

Back

Dialog Control Language – Action (Hành động trong DCL)

Action (Các hành động)

In order for the dialog box to respond to user interaction, you must set up an action event. A trigger, to fire when the user selects an item from a list or presses a button. This works off of the KEY defined for each control inside the DCL file. If the key for a button is named "mybutton" then you must have an action named "mybutton". Otherwise the user will press the button and nothing will happen. Let's take a look at the action_tile.

Để hộp thoại tương tác với người sử dụng, bạn phải tạo ra một sự kiện hành động. Một cái lẫy để khai hỏa khi người sử dụng chọn một khoản mục trong danh sách hoặc nhấn một nút. Điều này sẽ ngắt một khoá được xác định cho mỗi thao tác điều khiển trong tập tin DCL. Nếu khóa cho một nút được đặt tên là “mybutton” thì bạn phải có một hành động được đặt tên là “mybutton”. Nếu không khi người sử dụng nhấn nút đó sẽ chẳng có gì xảy ra. Ta hãy xem xét phần tử hành động này.

```
(action_tile   "key"    "action")
```

The action_tile has two parameters. "Key" and "action". (Phần tử hành động có hai tham số là key và action)

"Key" - The name of the key you defined with the control inside the DCL file. (Tên của khoá mà bạn đã xác định với khoá điều khiển trong tập tin DCL)

"Action"- The action you want taken when the action event is fired. You can set a variable or run a function. (Hành động mà bạn muốn thực hiện khi sự kiện hành động được khai hỏa. Bạn có thể tạo một biến hoặc thực thi một hành động)

Examples: Các ví dụ:

Let's take for example, you have a button named "test" and you want to set the variable named myTEST to 1 if the user presses the button: (Lấy ví dụ, bạn có một nút đặt tên là "test" và bạn muốn đặt một biến tên là "myTEST" về 1 khi người sử dụng nhấn nút này:)

```
(action_tile   "test"   "(setq myTest 1)")
```

Notice the key is in quotes and the setq function is in quotes. This is standard procedure for the action_tile statement. Next let's say you have the same thing except you want to run a function named "saveVars" when the user presses the test button. (Chú ý là tên khoá và hàm setq phải được đặt trong ngoặc kép. Đó là thủ tục tiêu chuẩn cho các hàm thông báo phần tử hành động. Tiếp theo bạn có một hàm thông báo phần tử hành động tương tự như trên trừ việc bạn muốn chạy hàm saveVar khi người sử dụng nhấn nút test)

```
(action_tile   "test"    "(saveVars)")
```

Notice is is the same as the above. The (saveVars) is inside quotes. (Chú ý là cũng như ở trên hàm saveVar phải đặt trong ngoặc kép)

What if you wanted to do both, run the function and set a variable? (Vậy khi bạn muốn thực hiện cả hai điều trên chạy một chương trình và đặt một biến thì sao?)

```
(action_tile   "test"   "(setq myTest 1)(saveVars)")
```

Simply put both inside the quotes. (Đơn giản là đặt cả hai hàm setq và saveVar vào trong ngoặc kép.)

One more thing....What if the button is set to be the cancel or accept button for the dialog box? No problem...

Một điều nữa, nếu nút này được đặt là nút xóa hay chấp nhận đối với hộp thoại thì sao? Không có sao cả…

```
(action_tile   "test"   "(setq myTest 1)(saveVars)(done_dialog)")
```

Add the done_dialog statement to the end. [Inside the quotes](Thêm thông báo done_dialog vào cuối trước khi đóng ngoặc kép.)

That is about all there is to an action_tile. Remember, anything with a key can have an action call.

Đó là tất cả về một phần tử hành động. Nhớ rằng, bất kỳ khóa nào cũng có thể có một lệnh hành động.

Back

Dialog Control Language - Set and Mode Tile (Đặt và tạo chức năng cho phần tử trong DCL)

Set_Tile and Mode_Tile (Đặt phần tử và tạo chức năng phần tử)

Set_Tile - is used to set the value of a control. (Sử dụng để đặt giá trị cho một nút điều khiển)

```
(set_tile  "key"   "value")
```

Set_Tile has two parameters. "Key" and "Value". (Hàm Set_tile có hai tham số là "key" và "value")

"Key" - The name of the key you defined with the control inside the DCL file. (là tên của khóa mà bạn xác định với nút điều khiển trong tập tin DCL)

"Value"- The new value for the control. (Giá trị mới của nút điều khiển)

- Edit_box (Hộp hiệu chỉnh)
  - `(set_tile "mybox" "Jeff")` Displays Jeff in the edit box. (Hiển thị Jeff trong hộp hiệu chỉnh)
  - `(set_tile "mybox" "4'-3 1/2")` Displays 4'-3 1/2 in the edit box. (Hiển thị 4'-3 ½ trong hộp hiệu chỉnh)
- List_box (Hộp danh sách)
  - `(set_tile "mylist" "0")` Selects the first item in the list. (Chọn khoản mục thứ nhất trong danh sách)
  - `(set_tile "mylist" "5")` Selects the sixth item in the list. (Chọn khoản mục thứ sáu trong danh sách)
  - `(set_tile "mylist" "")` No Items are selected. (Không chọn khoản mục nào)
- PopUp_List (Danh sách thả xuống)
  - `(set_tile "mylist" "0")` Selects the first item in the list. (Chọn khoản mục thứ nhất trong danh sách)
  - `(set_tile "mylist" "5")` Selects the sixth item in the list. (Chọn khoản mục thứ sáu trong danh sách)
  - `(set_tile "mylist" "")` No Items are selected.(Không chọn khoản mục nào)

- Toggle (Bật hoặc tắt kiểm soát hộp điều khiển)
  - `(set_tile "mytog" "0")` Removes the check from the box. (Xóa kiểm soát)
  - `(set_tile "mytog" "1")` Checks the box. (Kiểm soát)
- Radio_Button (Nút lựa chọn)
  - `(set_tile "myradio1" "1")` Moves the selection to the first radio button. (Chuyển việc lựa chọn về nút lựa chọn 1)
  - `(set_tile "myradio2" "1")` Moves the selection to the 2nd radio button. (Chuyển việc lựa chọn về nút lựa chọn 2)

Use Set_Tile after the new_dialog and before the action_tiles. Set_Tile can also be used as a function while the dialog box is displayed. (Sử dụng hàm Set_Tile sau thông báo new_dialog và trước thông báo action_tile. Hàm Set_tile cũng có thể sử dụng như một hàm trong lúc hộp thoại được hiển thị)

Mode_Tile - is used to enable or disable a control. (Hàm được sử dụng để bật hoặc tắt một nút điều khiển)

```
(mode_tile  "key"   value)
```

Mode_Tile has two parameters. "Key" and Value. (Hàm có hai tham số là "Key" và Value)

"Key" - The name of the key you defined with the control inside the DCL file. (là tên của khóa mà bạn đã xác định với nút điều khiển trong tập tin DCL)

Value- The new value for the control. 0 = Enabled 1 = Disabled (Giá trị mới của nút điều khiển 0 là bật 1 là tắt)

`(mode_tile "mylist" 0)` Enables the list box (Kích hoạt hộp danh sách)

`(mode_tile "mylist" 1)` Disables the list box (Tắt hộp danh sách)

Use Mode_Tile after the new_dialog call and before the action_tiles. Mode_Tile can also be used as a function while the dialog box is displayed.(Hàm Mode_tile sử dụng sau hàm lệnh new_dialog và trước hàm action_tile. HàmMode_tile cũng có thể được sử dụng trong lúc hộp thoại được hiển thị)

That is all there is to set_tile and mode_tile. We will used them later on in the tutorial so you can see them in action.(Đó là tất cả mọi điều về hàm Set_tile và hàm Mode_tile. Ta sẽ sử dụng chúng trong phần sau và bạn sẽ thấy chúng hoạt động ra sao.)

Back

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

Dialog Control Language – List (Danh sách trong DCL)

List and how to handle them.(Danh sách và cách điều khiển chúng)

List_box and popup_list handle a list of items in basically the same manner. You have to load your list into the list box and at some point you have to decide which item or items were selected. In the "Saving data from a dialog box" I cover how to find the selected items from a list_box and a popup_list. How do we get the list inside the list_box? That is what we need to cover in this section. So let's get started.

Hộp danh sách và danh sách thả xuống về cơ bản điều khiển một danh sách theo cùng một phương cách. Bạn đã tải danh sách của bạn thành một hộp danh sách tại một điểm nào đó và bạn phải quyết định khoản mục nào hay những khoản mục nào sẽ được lựa chọn. Trong phần "Lưu giữ liệu từ một hộp thoại" tôi sẽ trình bày cách để tìm những khoản mục được lựa chọn từ một hộp danh sách hay một danh sách thả xuống. Làm sao để lấy đượcdanh sách đó bên trong một hộp danh sách? Đó là điều bạn sẽ học trong phần này. Bắt đầu thôi.

If you looked at the previous sections you know the AutoLisp and DCL basic model. Let's get a copy of that in this section so we can look at it. I will replace the edit_box with a list_box control and add the code to handle the list. The list handling code is shown in the "Saving data from a dialog box" section of this tutorial.

Ở các phần trước, bạn đã biết về các mẫu cơ bản về Autolisp và DCL. Ở phần này ta sẽ copy chúng lại để xem xét. Ta sẽ thay thế hộp hiệu chỉnh bằng một hộp danh sách và thêm vào các mã điều khiển danh sách đó. Mã điều khiển danh sách này được trình bày trong phần "Lưu dữ liệu từ một hộp thoại " của tài liệu này

I will show the revised and new code in red below. All of the **black code is unchanged** from the Basic Model and the instructions from the "list_box multiple selection" area of the "Saving data from a Dialog box" page.

Tôi sẽ trình bày các phần mả chỉnh sửa và mã mới bằng màu đỏ. Tất cả các mã màu đen là không thay đổi so với mẫu cơ bản và so với các chỉ dẫn ở mục "hộp danh sách đa lựa chọn" của phần "Lưu dữ liệu từ một hộp thoại"

The Basic Revised Model (Mẫu cơ bản được chỉnh sửa)

The DCL File: (Tập tin DCL)

```
EXAMPLE : dialog {
          label = "EXAMPLE.lsp";        \\ Puts a label on the dialog box (Gắn nhãn hộp thoại)
          : column {
            : row {
              : boxed_column {
                : list_box {                \\ I replaced the edit_box with a list_box. (Thay thế hộp hiệu chỉnh bằng hộp danh sách)
```

```
                    label ="Choose Items";
                    key = "mylist";
                    height = 15;
                    width = 25;
                    multiple_select = true;   \\ Multiple selection is on (Bật đa lựa chọn)
                    fixed_width_font = true; \\ Fixed Width Font is on (Bật font đậm)
                    value = "0";
                  }
                }
              }
              : row {
                : boxed_row {
                  : button {
                    key = "accept";
                    label = " Okay ";
                    is_default = true;
                  }
                  : button {
                    key = "cancel";
                    label = " Cancel ";
                    is_default = false;
                    is_cancel = true;
                  }
                }
              }
            }
}
```

The AutoLisp File: Tập tin Autolisp:

```
;;;--- EXAMPLE.lsp

;;;--- I replaced the saveVars routine with the one from the "Save data from a list" section of this tutorial.
;;;      I also changed the things marked in red. (Tôi thay thế vòng lưu biến saveVar bằng một trong các
;;;      cách được trình bày trong phần " Lưu dữ liệu từ một hộp thoại" của tài liệu này. Tôi cũng thay đổi
;;;;     những điều được đánh dấu màu đỏ.)


(defun saveVars(/ readlist count item)

;;;--- Notice the "mylist" is the action key associated with the DCL file and
  ;;;    the myList is the variable for the list built below.(Chú ý rằng "mylist" là khoá hành
;;;;     động tương ứng với tập tin DCL và myList là biến cho danh sách được tạo thành dưới đây)

  ;;;--- Setup a list to hold the selected items (Tạo một danh sách để giữ các khoản mục chọn)
  (setq retList(list))

  ;;;--- Save the list setting (Lưu việc đặt danh sách)
  (setq readlist(get_tile "mylist"))
```

```
  ;;;--- Setup a variable to run through the list (Đặt biến để chạy qua toàn bộ danh sách)
  (setq count 1)

  ;;;--- cycle through the list getting all of the selected items (Lặp qua toàn bộ danh sách
để lấy các khoản mục được lựa chọn)
  (while (setq item (read readlist))
    (setq retlist(append retList (list (nth item myList))))
    (while
      (and
        (/= " " (substr readlist count 1))
        (/= ""   (substr readlist count 1))
      )
      (setq count (1+ count))
    )
    (setq readlist (substr readlist count))
  )
)


(defun C:EXAMPLE()

  ;;;--- I need to build a list of data ( Tôi cần tạo một danh sách các dữ liệu)
  (setq myList(list "1" "2" "3" "4" "5" "6" "7" "8"))

  ;;;--- Load the dcl file (Tải tập tin DCL)
  (setq dcl_id (load_dialog "EXAMPLE.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded (Tải định nghĩa hộp thoại nếu
nó chưa được tải)
  (if (not (new_dialog "EXAMPLE" dcl_id) ) (exit))

  ;;;-- Here, I add the data to the list_box control (Ở đây tôi thêm dữ liệu vào hộp danh sách)
  ;;;    I do this after the new_dialog call and before the action_tiles.( Điều này thực
;;;;     hiện sau hàm new_dialog và trước hàm action_tile.)
  (start_list "mylist" 3)
  (mapcar 'add_list myList)
  (end_list)

;;;--- Notice the "mylist" is the action key associated with the DCL file and
  ;;;    the myList is the variable for the list built above. (Chú ý rằng "mylist" là khoá
;;;;     hành động tương ứng với tập tin DCL và myList là biến cho danh sách được tạo thành dưới đây)

  ;; If an action event occurs, do this function (Nếu một thao tác xảy ra, thực hiện hàm này)
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")


  ;;; If an action event occurs, do this function (Nếu một thao tác xảy ra, thực hiện hàm này)
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")

  ;;;--- Display the dialog box (Hiển thị hộp thoại)
  (start_dialog)
```

```
  ;;;- If the cancel button was pressed - display message (Hiển thị nếu nút Cancel được nhấn)
  (if (= ddiag 1)
    (princ "\n \n ...EXAMPLE Cancelled. \n ")
  )

  ;;;--- If the "Okay" button was pressed (Nếu nút "Okay" được nhấn)
  (if (= ddiag 2)

    ;;;--- Here, I decide what to do with my data (Ở đây tôi xác định điều cần thực hiện với biến)
    ;;;     I'm going to print each selected item on the command line.(In từng khoản mục được chọn ra dòng lệnh)
    (foreach a retList
      (princ "\n")(princ a)
    )
  )

  ;;;--- Suppress the last echo for a clean exit (Triệt tiêu sự lặp lại lệnh cuối và thoát êm)
  (princ)

)
```

So, in order to display a list in a dialog box, you must first build the list. Then use the start_list, add_list, and end_list functions just after the new_dialog call and before the action_tiles. This is the same for both a list_box and a popup_list. That wasn't so bad. Was it?

Vậy đó, để hiển thị một danh sách trong một hộp thoại, trước hết bạn phải tạo danh sách đó. Sau đó sử dụng các hàm start_list, add_list, và end_list ngay sau hàm lệnh new_dialog và trước hàm action_tile. Điều này là như nhau đối với hộp danh sách và danh sách thả xuống. Không tồi chứ, phải không?

Back

Dialog Control Language–SaveVars(Lưu biến trong DCL-SaveVars)

In this section we are going to take a look at the SaveVars routine and disect it for each type of control. The types of controls that we sometimes need to save are :

Trong phần này, ta sẽ xem xét vòng lưu biến SaveVars và **disect** nó với từng loại nút điều khiển. Các loại nút điều khiển mà đôi khi chúng ta cần lưu lại là:

Edit_box - List_box - Popup_List - Radio_Buttons - Radio_Column - Radio_Row - Toggle

The SaveVars routine is executed just before the dialog box is issued a done_dialog call. You cannot get

the data from the dialog box once it has been shut down. If you remember our action calls in the AutoLisp Program, they looked something like this:

Vòng lưu biến SaveVars được thực hiện ngay trước khi hộp thoại thực hiện lệnh gọi done_dialog. Bạn không thể lấy được dữ liệu từ hộp thoại khi nó đã đóng lại. Trừ phi bạn nhớ gọi lệnh hành động trong Autolisp. Chương trình sẽ giống như sau:

```
;;;-- If an action event occurs, do this function (Nếu một thao tác xảy ra, thực hiện hàm này)
(action_tile "accept" "(saveVars)(done_dialog)")
```

Notice the (saveVars) is just before the (done_dialog). This is the proper order. The settings from the dialog box will be saved just before it is shut down. (Chú y rằng hàm (saveVars) được đặt ngay trước hàm (done_dialog). Đó là trật tự đúng. Điều này sẽ khiến hộp thoại phải lưu lại biến trước khi đóng lại.)

The SaveVars routine always has the same definition: (Vòng lưu biến saveVars luôn có cùng một định nghĩa là:)

```
(defun saveVars()

  ;stuff here (Thực hiện ba cái lăng nhăng ở đây)

)
```

That's the shell. We need to work on the "stuffing". Let's get to it. (Đó mới chỉ là cái vỏ. Ta cần phải lảm việc với "ba cái lăng nhăng" nữa. Hãy tiếp tục đi nữa nào.)

Stuffing (Các hành động thực hiện trong vòng lưu biến SaveVars)

There are three parts to a control with an action call back statement. (Một nút điều khiển với một hành động gọi lại hàm thông báo có ba phần gồm:)

1. The DCL definition (Định nghĩa DCL)
2. The action call in the AutoLisp file. (Lệnh gọi hành động trong tập tin Autolisp)
3. The code inside the SaveVars routine. (Mã bên trong vòng lưu biến SaveVars)

In each of the following controls I will show you all three parts so we know how to put it all together later. Although, all of the parts are important, try to focus on the SaveVars routine.

Trong mỗi nút điều khiển dưới đây, tôi sẽ trình bày cả ba phần để bạn có thể biết cách đặt chúng kết hợp với nhau. Mặc dầu cả ba phần đều quan trọng nhưng hãy cố gắng tập trung vào vòng lưu biến SaveVars

Edit_box - List_Box_Single_Selection - List_Box_Multiple_Selection - Popup_List

Edit_Box (Hộp hiệu chỉnh)

1	`: edit_box {`
	`  key = "edbox";`
	`  label = "Bolt Circle Diameter:";`
	`  edit_width = 8;`
	`  value = "10";`
	`}`

2	**No Action call required.** We will get the data the user typed when we use the SaveVars routine. But, if your deadset about doing something when the user types in the edit box, here you go: **Không yêu cầu lệnh gọi hành động.** Ta sẽ lấy dữ liệu mà người sử dụng nhập vào khi ta sử dụng vòng lưu biến Savevars. Nhưng nếu bạn muốn cố định thực hiện một điều gì đó khi người sử dụng nhập vào hộp hiệu chỉnh , bạn sẽ thực hiện: `(action_tile "edbox" "(doThisFunction)")`
3	To get the data as **a real number:** (Để có dữ liệu là **một số thực**:) (defun saveVars() `  (setq edBox(distof(get_tile "edbox")))`)
	To get the data as **a string:** (Để có dữ liệu là một **chuỗi:**) (defun saveVars() `  (setq edBox(get_tile "edbox"))`)

NOTE: Notice I used the key name as the variable to store the data in. You will find this method makes it easier to keep things straight. DCL is case sensitive so, I never capitalize key names, only autolisp variable names if they are made up of more than one word. Like: thisIsMyVariable.

CHÚ Ý: Tôi sử dụng tên khoá như một biến để lưu dữ liệu vào. Bạn sẽ tìm thấy phương pháp này là dễ hơn để giữ trực tiếp mọi thứ của khoá. DCL là rất nhậy với kiểu chữ in hoa hay in thường, tôi không bao giờ sử dụng chữ in hoa cho các tên khóa, mà chỉ sử dụng trong tên biến củA Autolisp khi tên đó dài hơn một từ. Ví dụ: thisIsMyVariable.

List_Box Single Choice Only (Hộp danh sách với sự lựa chọn đơn duy nhất)

1	`: list_box {`
	`  key = "mylist";`
	`  label = "Available Choices";`
	`  multiple_select = "FALSE";` // Sets single selection (Đặt sự lựa chọn đơn)
	`  width = 40;`
	`  height = 20;`

```
  fixed_width_font = true;      // Equally spaced characters (Các ký tự cách đều nhau)
  value = "";                   // Start with no item selected (Không chọn trước)
}
```

2

No Action call required. We will get the selected item when we use the SaveVars routine. But, sometimes you need to change things in the dialog box if the user makes a selection, so:
Không yêu cầu lệnh gọi hành động. Ta sẽ lấy khoản mục được chọn khi sử dụng vòng lưu biến SaveVars. Nhưng đôi khi bạn cần thay đổi những điều trong hộp thoại nếu người sử dụng thực hiện sự lựa chọn, vậy thì:

```
(action_tile "edbox" "(doThisFunction)")  ;;;
```
Chỗ này chắc cụ Jeff muốn kiểm tra mấy thằng học mót đây, phải thay "edbox' bằng "mylist" mới phải chớ???

3

```
(defun saveVars()

  ;;;--Get the selected item from the list (Lấy các khoản mục được chọn từ danh sách)
  (setq sStr(get_tile "mylist"))

  ;;;--- If the index of the selected item is not "" then something was selected
(Nếu số thứ tự của khoản mục được chọn là khác nil "" thì chọn một điều gì đó)
  (if(/= sStr "")
    (progn

      ;;;--- Something is selected, so convert from string to integer (Có đối
tượng được chọn, chuyển đổi tượng từ chuỗi thành số nguyên)
      (setq sIndex(atoi sStr))

      ;;;--- And get the selected item from the list (Và lấy đối tượng được chọn từ
danh sách)
      (setq sName(nth sIndex myList))
    )

    ;;;--- Else, nothing is selected (ngược lại, không có đối tượng nào được chọn)
    (progn

      ;;;--- Set the index number to -1 (Đặt số thứ tự về -1)
      (setq sIndex -1)

      ;;;--- And set the name of the selected item to nil (Và đặt tên đối tượng được
chọn về nil)
      (setq sName nil)
    )

  )
)
```

Notes: Chú ý:

1. This should only be used when single selection is required. This will not work on multiple selection.(Điều này chỉ được sử dụng với việc lựa chọn đơn. Nó không làm việc nếu là đa lựa chọn)

2. This is assuming there is a list of items called myList. (Giả sử đã có danh sách tên là myList)

List_Box Multiple Choice (Hộp danh sách đa lựa chọn)

1	`: list_box {`
	`  key = "mylist";`
	`  label = "Available Choices";`
	`  multiple_select = "TRUE";` // Sets multiple selection (Đặt đa lựa chọn)
	`  width = 40;`
	`  height = 20;`
	`  fixed_width_font = false;` // Use default font [no alignment](Sử dụng font mặc định, Không căn hàng)
	`  value = "4";` // Start with item 5 selected.(Bắt đầu với khoản mục thứ 5 được chọn)
	`}`

2	**No Action call required.** We will get the selected items when we use the SaveVars routine. But, sometimes you need to change things in the dialog box if the user makes a selection, so: **Không yêu cầu lệnh gọi hành động.** Ta sẽ lấy các khoản mục được chọn khi sử dụng vòng lưu biến SaveVars. Nhưng đôi khi bạn cần thay đổi những điều trong hộp thoại khi người sử dụng tạo một lựa chọn, Thế thì: `(action_tile "mylist" "(doThisFunction)")`

3	`(defun saveVars(/ readlist count item)`
	
	`  ;;;--- Setup a list to hold the selected items` (Đặt một danh sách để giữ các khoản mục được chọn)
	`  (setq retList(list))`
	
	`  ;;;--- Save the list setting` (Lưu lại danh sách chọn)
	`  (setq readlist(get_tile "files"))`
	
	`  ;;;-- Setup a variable to run through the list` (Đặt biến để lặp qua toàn bộ danh sách)
	`  (setq count 1)`
	
	`  ;;;--- cycle through the list getting all of the selected items` (Lặp qua toàn bộ danh sách các khoản mục được chọn)
	`  (while (setq item (read readlist))`
	`  (setq retlist(append retList (list (nth item fileNames))))`
	`  (while`
	`  (and`
	`  (/= " " (substr readlist count 1))`
	`  (/= ""  (substr readlist count 1))`
	`  )`
	`  (setq count (1+ count))`
	`  )`
	`  (setq readlist (substr readlist count))`
	`  )`
	`)`
	
	Note: This method can be used for a single or multiple selection list_box. (Chú ý rằng phương pháp này chỉ sử dụng cho hộp danh sách đa lựa chọn)

PopUp_List (Danh sách thả xuống)

1	`: popup_list {`
	`  key = "mylist";` // action key (Khoá hành động)
	`  fixed_width_font = false;` // fixed width font off (Tắt font rộng đều)
	`  width = 20;` // width of popup list (Độ rộng danh sách thả)
	`  height = 20;` // height of popup list (Độ cao danh sách thả)
	`}`

2	**No Action call required.** We will get the selected item when we use the SaveVars routine. But, sometimes you need to change things in the dialog box if the user makes a selection, so: **Không yêu cầu lệnh gọi hành động.** Ta sẽ lấy các khoản mục được chọn khi sử dụng vòng lưu biến SaveVars. Nhưng đôi khi bạn cần thay đổi những điều trong hộp thoại khi người sử dụng tạo một lựa chọn, Thế thì: `(action_tile "mylist" "(doThisFunction)")`

3	`(defun saveVars()`
	
	`  ;;;--- Get the selected item from the list` (Lấy khoản mục lựa chọn từ danh sách)
	`  (setq sStr(get_tile "mylist"))`
	
	`  (if(= sStr "")(alert "No Item was Selected!"))`
	
	`)`

Radio_Column and Radio_Buttons (Cột lựa chọn và các nút lựa chọn)

1	`: radio_column {` // Use boxed_radio_column if a box is required. (Sử dụng cột hộp lựa chọn nếu yêu cầu một hộp)
	`  label = "Choices";` // Label for the column or boxed_column (Gắn nhãn cho cột hay cột hộp)
	`  key = "choices";` // Action key for the radio column (Khoá hành động của cột lựa chọn)
	
	`  : radio_button {` // First radio button (Nút lựa chọn thứ nhất)
	`  label = "Choice 1";`
	`  key = "choice1";`
	`  }`
	
	`  : radio_button {` // Second radio button (Nút lựa chọn thứ hai)
	`  label = "Choice 2";`
	`  key = "choice2";`
	`  }`
	
	`  : radio_button {` // Third radio button (Nút lựa chọn thứ ba)
	`  label = "Choice 3";`
	`  key = "choice3";`

```
  }

  : radio_button {          // Fourth radio button (Nút lựa chọn thứ tư)
    label = "Choice 4";
    key = "choice4";
  }

}                           // Close the radio_column (Đóng cột lựa chọn)
```

2

No Action call required. We will get the selected item when we use the SaveVars routine. But, sometimes you need to change things in the dialog box if the user makes a selection, so:
Không yêu cầu lệnh gọi hành động. Ta sẽ lấy các khỏan mục được chọn khi sử dụng vòng lưu biến SaveVars. Nhưng đôi khi bạn cần thay đổi những điều trong hộp thoại khi người sử dụng tạo một lựa chọn, Thế thì:

```
(action_tile "choices" "(doThisFunction)")      // If any choice is made (Nếu một khoản mục bất kỳ được lựa chọn)

(action_tile "choice1" "(doThatFunction)")      // If choice1 is made (Nếu khoản mục thứ nhất được chọn)

(action_tile "choice2" "(doDisFunction)")       // If choice2 is made (Nếu khoản mục thứ hai được chọn)

etc.
```

3

```
 (defun saveVars()

  ;;;--- Get the key of the choice made (Lấy khóa của sự lựa chọn đã thực hiện)
  ;;;    [ returns (Trả về giá trị) "choice1" "choice2" "choice3" or "choice4" ]

  (setq choicesVal(get_tile "choices"))

)

OR (Hoặc)

(defun saveVars()

;;;--- Get the value of each item (Lấy giá trị của mỗi một khoản mục)
 (setq choice1(atoi(get_tile "choice1")))   // 0 = not chosen   1 = chosen (0 là không chọn, 1 là chọn)
 (setq choice2(atoi(get_tile "choice2")))   // 0 = not chosen   1 = chosen (0 là không chọn, 1 là chọn)
 (setq choice3(atoi(get_tile "choice3")))   // 0 = not chosen   1 = chosen (0 là không chọn, 1 là chọn)
 (setq choice4(atoi(get_tile "choice4")))   // 0 = not chosen   1 = chosen (0 là không chọn, 1 là chọn)

)
```

Radio_Row And Radio_Buttons (Hàng lựa chọn và các nút lựa chọn)

1

```
: radio_row {                    // Use boxed_radio_row if a box is required. (Sử dụng hàng hộp lựa chọn nếu yêu cầu một hộp)
  label = "Choices";             // Label for the row or boxed_row (Gắn nhãn cho hàng hay hàng hộp)
  key = "choices";     // Action key for the radio row (Khoá hành động cho hàng lựa chọn)

  : radio_button {               // First radio button (Nút lựa chọn thứ nhất)
    label = "Choice 1";
    key = "choice1";
  }

  : radio_button {               // Second radio button (Nút lựa chọn thứ hai)
    label = "Choice 2";
    key = "choice2";
  }

  : radio_button {               // Third radio button  (Nút lựa chọn thứ ba)
    label = "Choice 3";
    key = "choice3";
  }

  : radio_button {               // Fourth radio button (Nút lựa chọn thứ tư)
    label = "Choice 4";
    key = "choice4";
  }

}                                // Close the radio_row (Đóng hàng lựa chọn)
```

2

No Action call required. We will get the selected item when we use the SaveVars routine. But, sometimes you need to change things in the dialog box if the user makes a selection, so:
Không yêu cầu lệnh gọi hành động. Ta sẽ lấy các khoản mục được chọn khi sử dụng vòng lưu biến SaveVars. Nhưng đôi khi bạn cần thay đổi những điều trong hộp thoại khi người sử dụng tạo một lựa chọn, Thế thì:

```
(action_tile "choices" "(doThisFunction)")     // If any choice is made (Nếu bất kỳ khoản mục nào được chọn)

(action_tile "choice1" "(doThatFunction)")     // If choice1 is made (Nếu choice1 được chọn)

(action_tile "choice2" "(doDisFunction)")      // If choice2 is made  (Nếu choice2 được chọn)

etc.
```

3

```
(defun saveVars()

  ;;;--- Get the key of the choice made (Lấy khoá của lựa chọn được thực hiện)
  ;;;    [ returns (Trả về giá trị) "choice1" "choice2" "choice3" or "choice4" ]
```

```
  (setq choicesVal(get_tile "choices"))

)

OR (HoặC)

(defun saveVars()

;;;--- Get the value of each item (Lấy giá trị của mỗi khoản mục)
 (setq choice1(atoi(get_tile "choice1"))) // 0 = not chosen  1 = chosen (0 là không chọn, 1 là chọn)
 (setq choice2(atoi(get_tile "choice2"))) // 0 = not chosen  1 = chosen (0 là không chọn, 1 là chọn)
 (setq choice3(atoi(get_tile "choice3"))) // 0 = not chosen  1 = chosen (0 là không chọn, 1 là chọn)
 (setq choice4(atoi(get_tile "choice4"))) // 0 = not chosen  1 = chosen (0 là không chọn, 1 là chọn)

)
```

Toggle (Bật và tắt kiểm soát nút điều khiển)

1

```
: toggle {
  key = "tog1";                    // Action key (Khoá hành động)
  label = "Name";                  // Label (Nhãn)
}
```

2

No Action call required. We will get the value of the toggle when we use the SaveVars routine. But, sometimes you need to change things in the dialog box if the user makes a selection, so:
Không yêu cầu lệnh gọi hành động. Ta sẽ lấy các khoản mục được chọn khi sử dụng vòng lưu biến SaveVars. Nhưng đôi khi bạn cần thay đổi những điều trong hộp thoại khi người sử dụng tạo một lựa chọn, Thế thì:

```
(action_tile "tog1" "(doThisFunction)")
```

3

```
(defun saveVars()

  ;;;--- Get the selected item from the radio column (Lấy khoản mục lựa chọn từ cột lựa chọn)
  (setq tog1Val(atoi(get_tile "tog1")))   // 0 = unchecked  1 = checked (0 là không chọn, 1 là chọn)

)
```

Ok_Cancel

1	`ok_cancel;` // Only one line required (Yêu cầu một dòng duy nhất) Note: I usually define my own okay and cancel buttons using two standard buttons, but this works fine. (Chú ý là tôi thường xác định hai nút Okay và Cancel của mình bằng cách sử dụng hai nút tiêu chuẩn mà chúng làm việc rất tốt)

2	You will need two action calls for this button...both close the dialog box but the accept or "Okay" key will save the dialog box settings before shutting the dialog box down.
	Bạn cần có hai lệnh gọi hành động cho nút này, cả hai đều đóng hộp thoại nhưng khóaAccept hay Okay sẽ lưu việc đặt hộp thoại lại trước khi đóng hộp thoại.
	
	`(action_tile "cancel" "(setq ddiag 1)(done_dialog)")`
	
	`(action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")`
	
	Ahh....finally the SaveVars routine shows up. (A ha, vậy là xong với vòng lưu biến SaveVars rồi)

3	Nothing to do in the saveVars routine for these buttons. (Chả còn gì làm thêm với các nút này trong vòng lưu biến SaveVars)

Back

Dialog Control Language - Part 1 (Phần thực hành 1 về DCL)

Part 1 - Buttons (Phần 1- Các nút)

Let's build a working DCL file showing us exactly how to handle buttons. (Ta sẽ xây dựng tập tin DCL hoạt động để thấy được chính xác cách điều khiển các nút)

We will build a DCL file containing three buttons plus a Close [Cancel] button. Each of the three buttons will display a message when pressed. (Ta sẽ tạo một tập tin DCL gồm ba nút và thêm một nút Close [Cancel]. Mỗi một trong ba nút sẽ hiển thị một thông báo khi bị nhấn.)

Layout thoughts: I will place the buttons in a column, (stacked on top of each other). Then I'll put the Close button at the bottom on a row of it's own. So...I'll need something like this:

Thiết kế hộp thoại: Ta sẽ đặt các nút trong một cột (Các nút chồng lên nhau). Sau đó ta sẽ đặt nút Close ở hàng cuối cùng của hộp thoại. Do đó, ta sẽ cần một thứ đại loại như thế này:

```
: column {
  : boxed_column {
```

```
    : button {
      // Put code for button 1 here (Đặt mã cho nút 1 ở đây)
    }
    : button {  ]
      // Put code for button 2 here (Đặt mã cho nút 2 ở đây)
    }
    : button {  ]
      // Put code for button 3 here  (Đặt mã cho nút 3 ở đây)
    }
  }
  : boxed_row {
    : button {
       // Put code for the Close button here (Đặt mã cho nút Close ở đây)
    }
  }
}
```

Let's copy in the code for the header and all of the controls above. I'll show them in red. Notice the key names and labels had to be changed. (Ta sẽ copy các mã phần tiêu đề và tất cả các mả điều khiển ở trên. Tôi sẽ trình bày chúng màu đỏ. Chú ý các tên khoá và các nhãn có sự thay đổi.)

```
SAMPLE1 : dialog {
          label = "Sample Dialog Box Routine - Part 1";
          : column {
            : boxed_column {
              : button {
                key = "but1";
                label = "Button 1";
                is_default = false;
              }
              : button {
                key = "but2";
                label = "Button 2";
                is_default = false;
              }
              : button {
                key = "but3";
                label = "Button 3";
                is_default = false;
              }
            }
            : boxed_row {
              : button {
                key = "cancel";
                label = "Close";
                is_default = true;
                is_cancel = true;
              }
```

```
            }

}
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE1.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Click chuột phải và copy tập tin trên. Mở NotePad và dán nó vào. Lưu lại nó dưới tên SAMPLE1.DCL ***Phải đảm bảo là bạn đã thay đổi giá trị của hộp thoại "Save as Type" thành "All Files" trước khi lưu tập tin, nếu không nó sẽ bị lưu với kiểu tập tin là txt. Lưu tập tin này trong đường dẫn tìm kiếm của AutoCad.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. (Tiếp theo ta sẽ copy chương trình Autolisp mẫu và chỉnh sửa lại. Các mã mới sẽ được trình bày màu đỏ.)

```
(defun C:SAMPLE1()

  ;;;--- Load the dcl file (Tải tập tin DCL)
  (setq dcl_id (load_dialog "SAMPLE1.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded (Tải định nghĩa hộp thoại nếu nó chưa được tải)
  (if (not (new_dialog "SAMPLE1" dcl_id))
    (progn
      (alert "The SAMPLE1.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function (Nếu có một sự hành động, thực hiện hàm)
  (action_tile "cancel" "(done_dialog)")

  ;;;--- Display the dialog box (Hiển thị hộp thoại)
  (start_dialog)

  ;;;--- Unload the dialog box (Thoát khỏi hộp thoại)
  (unload_dialog dcl_id)

  ;;;--- Suppress the last echo for a clean exit (Triệt tiêu việc lặp lại lệnh cuối và thoát êm)
  (princ)

)
```

I removed several lines from the model. I took out the part that checked to see if we hit the Cancel or Okay buttons. We don't need either in this program. I also removed the action tile for the okay button and removed "(setq ddiag 1)" from the cancel button.

Tôi xóa đi một vài dòng từ chương trình mẫu. Tôi lấy đi phần kiểm soát xem liệu ta có kích các nút Okay hay Cancel không. Ta không cần điều đó trong chương trình này. Tôi cũng xóa đi phần tử hành động đối với nút Okay và xoá đi "setq ddiag 1" từ phần tử hành động đối với nút Cancel.

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE1.LSP ***Be sure to***

change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.

Click chuột phải và copy tập tin trên. Mở NotePad và dán nó vào. Lưu lại nó dưới tên SAMPLE1.LSP . ***Phải đảm bảo là bạn đã thay đổi giá trị của hộp thoại "Save as Type" thành "All Files" trước khi lưu tập tin, nếu không nó sẽ bị lưu với kiểu tập tin là txt. Lưu tập tin này trong đường dẫn tìm kiếm của AutoCad.***

Let's load the program and see what the DCL file looks like. On the command line type this: (Ta hãy tải chương trình và xem tập tin DCL ra sao. Trên dòng lệnh nhập như sau:)

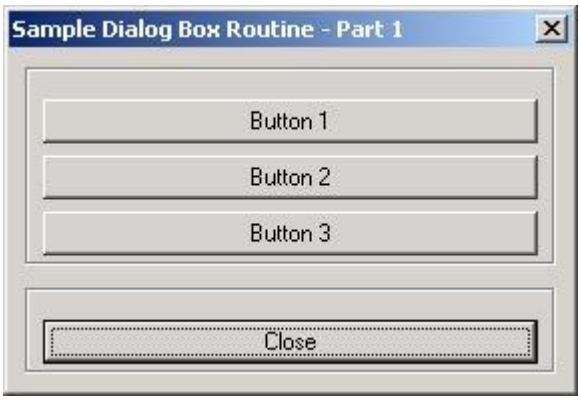
Command: (load "sample1") and press enter (Nhập (load "sample1") rồi nhấn Enter)

You should see this (Bạn sẽ thấy như sau:)

C:Sample1
Command:

Now type Sample1 and press enter. If everything went according to plan you should see this on your screen:

Bây giờ nhập Sample1 và nhấn Enter. Nếu mọi thứ đúng như kế hoach, bạn sẽ thấy trên màn hình:



The buttons do not work yet. Except for the close button. It should work fine. We need to add the function to print three different messages when the user presses each button. Let's send a parameter to the function to decide which message to display. If the parameter equals 1 we will print the first message. If 2 then print the second message. If 3 print the third message. Something like this:

Các nút vẫn chưa làm việc ngoại trừ nút Close. Chúng sẽ phải làm việc thôi. Ta cần thêm hàm để in ba thông báo khác nhau khi người sử dụng nhấn vào mỗi nút. Ta sẽ gửi thông số vào hàm để quyết định thông báo nào sẽ hiển thị. Nếu thông số bằng 1 ta sẽ in thông báo thứ nhất. Nếu thông số bằng 2 ta sẽ in thông báo thứ hai. Nếu thông số là 3 ta sẽ in thông báo thứ ba. Hàm sẽ như sau:

```
(defun doButton(a)
```

```
  (cond
    ((= a 1)(alert "Button 1 was pressed!"))
    ((= a 2)(alert "Button 2 was pressed!"))
    ((= a 3)(alert "Button 3 was pressed!"))
  )
)
```

Now we need to add the action calls for each of the buttons: (Bây giờ ta cần phải thêm các lệnh gọi phần tử hành động cho mỗi một nút)

```
  (action_tile "but1" "(doButton 1)")
  (action_tile "but2" "(doButton 2)")
  (action_tile "but3" "(doButton 3)")
```

This will send a parameter of 1,2, or 3 to the doButton function depending on which button is pressed. (Điều này sẽ gửi các thông số 1,2 và 3 vào hàm doButton tùy theo nút nào được nhấn)

Let's add the doButton function and the action calls to the autolisp program. It should look like this:

Ta hãy thêm hàm doButton và các lệnh gọi hành động vào chương trình Autolisp. Nó sẽ giống như sau:

```
(defun doButton(a)
  (cond
    ((= a 1)(alert "Button 1 was pressed!"))
    ((= a 2)(alert "Button 2 was pressed!"))
    ((= a 3)(alert "Button 3 was pressed!"))
  )
)

(defun C:SAMPLE1()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE1.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE1" dcl_id))
    (progn
      (alert "The SAMPLE1.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "but1" "(doButton 1)")
  (action_tile "but2" "(doButton 2)")
  (action_tile "but3" "(doButton 3)")
  (action_tile "cancel" "(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- Suppress the last echo for a clean exit
```

```
    (princ)
)
```

Save it and test it out. Everything working okay? (Lưu nó lại và thử chạy. Mọi thứ làm việc tốt chứ?)

When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE1()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

Khi bạn đã chạy thử chương trình và mọi thứ đều làm việc tốt, hãy chuyển dòng chữ màu xanh [(defun C:Sample1()] lên trên cùng của tập tin. Điều này sẽ làm cho tất cả các biến của bạn thành biến cục bộ và đặt lại chúng về nil khi chương trình kết thúc.

That's it. We're done. Vậy đó, Ta đã hoàn thành.

Back

Dialog Control Language - Part 2 (Phần thực hành 2 về DCL)

Part 2 - Edit_Box (Phần 2 - Hộp hiệu chỉnh)

Let's build a working DCL file showing us exactly how to handle edit boxes. (Ta sẽ xây dựng tập tin DCL để trình bày chính xác cách điều khiển hộp hiệu chỉnh)

We will build a DCL file containing two edit boxes plus a Okay and Cancel button. The information in the two edit boxes will be displayed after selecting the Okay button. (Ta sẽ tạo một tập tin DCL chứa hai hộp hiệu chỉnh và bộ nút Okay và Cancel. Thông tin trong hai hộp hiệu chỉnh sẽ được hiển thị sau khi nhấn nút Okay.)

Layout thoughts: I will place the edit boxes in a column, (stacked on top of each other). Then I'll put the Okay and Cancel buttons at the bottom in a row. So...I'll need something like this:

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

Thiết kế hộp thoại: Tôi sẽ đặt các hộp hiệu chỉnh trong một cột (các hộp chồng lên nhau). Sau đó đặt các nút Okay và Cancel vào hàng dưới cùng. Vậy nên tôi cần một thứ như sau:

```
: column {
  : boxed_column {
    : edit_box {
      // Put code for edit_box 1 here (Đặt mã của hộp hiệu chỉnh 1 ở đây)
    }
    : edit_box {
      // Put code for edit_box 3 here (Đặt mã của hộp hiệu chỉnh 3 ở đây)
    }
  }
  : boxed_row {
    : button {
       // Put code for the Okay button here (Đặt mã của nút Okay ở đây)
    }
    : button {
       // Put code for the Cancel button here (Đặt mã của nút Cancel ở đây)
    }
  }
}
```

Let's copy in the code for the header and all of the controls above. I'll show them in red. Notice the key names and labels had to be changed. (Ta sẽ copy phần tiêu đề và tất cả các mã điều khiển ở mẫu trên. Tôi sẽ thể hiện chúng màu đỏ. Chú ý là các tên khoá và các nhãn đã được thay đổi.)

```
SAMPLE2 : dialog {
          label = "Sample Dialog Box Routine - Part 2";
          : column {
            : boxed_column {
              : edit_box {
                key = "username";
                label = "Enter your Name:";
                edit_width = 15;
                value = "";
                initial_focus = true;
              }
              : edit_box {
                key = "userage";
                label = "Enter your Age:";
                edit_width = 15;
                value = "";
              }
            }
            : boxed_row {
              : button {
                key = "accept";
                label = " Okay ";
```

```
                is_default = true;
              }
              : button {
                key = "cancel";
                label = " Cancel ";
                is_default = false;
                is_cancel = true;
              }
            }
          }

}
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE2.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. (Tiếp theo ta sẽ copy chương trình Autolisp mẫu và chỉnh sửa lại. Các mã mới được thể hiện bằng màu đỏ.)

```
(defun C:SAMPLE2()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE2.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE2" dcl_id))
    (progn
      (alert "The SAMPLE2.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")  ;Chú ý thằng SaveVars
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample2 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )
```

```
  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE2.LSP ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

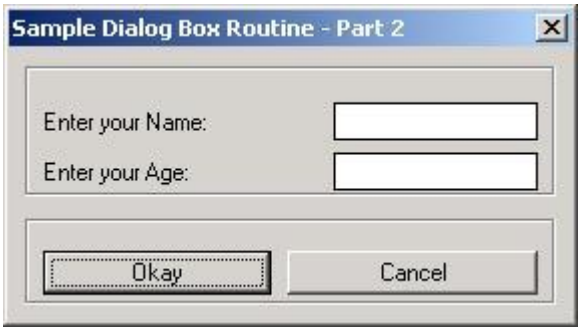
Let's load the program and see what the DCL file looks like. On the command line type this:

Command: (load "sample2") and press enter

You should see this

C:Sample2
Command:

Now type Sample2 and press enter. If everything went according to plan you should see this on your screen:



Looking good so far. We need to add the SaveVars function to save the strings in the edit boxes when the Okay button is pressed. *Look at the blue text in the Sample2.lsp program above.*(Trông quá đẹp. Ta cần thêm hàm SaveVars để lưu lại chuỗi trong hộp hiệu chỉnh khi nút Okay được nhấn. *Hãy xem dòng chữ màu xanh trong chương trình Autolisp Sample2.lsp ở trên)*

The edit box for the name needs to be a string. Dialog boxes return strings, so we do not need to modify it. All we have to do is use the get_tile function. (Hộp hiệu chỉnh dùng cho tên cần nhập một chuỗi. Các hộp thoại trả về giá trị là các chuỗi, vì thế ta không cần phải sửa nó. Mọi thứ ta cần làm là sử dụng hàm get_tile)

```
(setq userName(get_tile "username"))    ; Cú pháp này tớ chưa thông, tạm hiểu là hàm
                                        ;;get_tile trả về một chuỗi được nhập trong
                                        ;;;edit_box có khóa là đối số của nó.
```

The edit box for Age needs to be an integer. We will have to modify the get_tile results by using the ATOI function. This function converts a string to an integer. (Hộp hiệu chỉnh dùng cho tuổi cần nhập một số

nguyên. Ta sẽ phải sửa kết quả của hàm get_tile bằng cách sử dụng hàm ATOI. Hàm này đổi một chuỗi thành một số nguyên)

```
  (setq userAge( atoi (get_tile "userage")))
```

If we needed to convert to an Real number we would use DISTOF function instead of the ATOI function.
(Nếu ta cần đổi thành một số thực thì phải sử dụng hàm DISTOF thay cho hàm ATOI)

Our SaveVars routine would look like this: (Vòng lưu biến của ta sẽ trông giống như sau:)

```
(defun saveVars()
  (setq userName(get_tile "username"))
  (setq userAge( atoi (get_tile "userage")))
)
```

Our program would now look like this: (Chương trình lúc này sẽ như sau:)

```
(defun saveVars()
  (setq userName(get_tile "username"))
  (setq userAge(atoi(get_tile "userage")))
)

(defun C:SAMPLE2()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE2.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE2" dcl_id))
    (progn
      (alert "The SAMPLE2.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample2 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
```

```
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Last item. We need to replace the line in the program: `(princ "\n The user pressed Okay!")` with something to modify and display the userName and userAge data. Let's do something simple. We will find out how many days old this person is. (Tí cuối cùng. Ta cần thay thế dòng thông báo (princ "\n The user pressed Okay!") trong chương trình bằng tí sửa đổi và hiển thị Tên người sử dụng cùng với số ngày tuổi của hắn. Ta sẽ làm tí đơn giản này. Hãy tìm ra số ngày tuổi của gã.)

```
;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn

      ;;;--- Multiply the users age x 365 to get the number of days.(Nhân số tuổi với 365)
      (setq userAge(* userAge 365))

      ;;;--- Display the results (Hiển thị kết quả)
      (alert (strcat userName " is " (itoa userAge) " days old."))
    )
  )
```

Add the above to the file, save it and test it out. Everything working okay? (Thêm những điều trên vào chương trình. Lưu lại và chạy thử. Mọi thứ chạy tốt chứ?)

When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE2()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

That's it. We're done. (Biết rồi, khổ lắm…)

Back

Dialog Control Language - Part 3 (Phần thực hành 3 về DCL)

Part 3 - List_Box (Phần 3 - Hộp danh sách)

Let's build a working DCL file showing us exactly how to handle list boxes. (Ta sẽ xây dựng một tập tin DCL hoạt động để chỉ ra chính xác cách điều khiển một hộp danh sách)

We will build a DCL file containing two list boxes plus an Okay and Cancel button. We will set the first list box to single selection and the second to multiple selection. The selected items will be displayed on the screen after the user presses the Okay button. (ta sẽ tạo ra một tập tin DCL chứa hai hộp danh sách với bộ nút Okay và Cancel. Ta sẽ đặt hộp danh sách thứ nhất là lựa chọn đơn và hộp danh sách thứ hai là đa lựa chọn. Các khoản mục được lựa chọn sẽ được hiển thị trên màn hình sau khi người sử dụng nhấn nút Okay.)

Layout thoughts: I will place the list boxes in a row, (side by side). Then I'll put the Okay and Cancel buttons in a row at the bottom of the dialog box. So...I'll need something like this:

Thiết kế hộp thoại: Tôi sẽ đặt các hộp danh sách trong một hàng (Các hộp cạnh nhau). Sau đó đặt các nút Okay và Cancel vào hàng dưới cùng của hộp thoại. Vì thế tôi sẽ cần một thứ như sau:

```
: column {
  : boxed_row {
    : list_box {
      // Put code for list_box 1 here (Đặt mã cho hộp danh sách 1 ở đây)
    }
    : list_box {
      // Put code for list_box 2 here  (Đặt mã cho hộp danh sách 2 ở đây)
    }
  }
  : boxed_row {
    : button {
       // Put code for the Okay button here (Đặt mã cho nút Okay ở đây)
    }
    : button {
       // Put code for the Cancel button here (Đặt mã cho Nút Cancel ở đây)
    }
  }
}
```

Let's copy in the code for the header and all of the controls above from the "Controls" section of this tutorial. I'll show them in red. Notice the key names and labels had to be changed. (Ta hãy copy các mã tiêu đề và tất cả các mã điều khiển ở trên từ phần các nút điều khiển trong tài liệu này. Tôi sẽ thể hiện chúng bằng màu đỏ. Chú ý là các tên khóa và các nhãn đã được thay đổi)

```
SAMPLE3 : dialog {
          label = "Sample Dialog Box Routine - Part 3";
          : column {
            : boxed_row {
              : list_box {
                label ="Choose Item";
                key = "mylist1";
                height = 15;
                width = 25;
                multiple_select = false;
                fixed_width_font = true;
                value = "";
              }
              : list_box {
                label ="Choose Items";
                key = "mylist2";
                height = 15;
                width = 25;
                multiple_select = true;
                fixed_width_font = true;
                value = "";
              }
            }
            : boxed_row {
              : button {
                key = "accept";
                label = " Okay ";
                is_default = true;
              }
              : button {
                key = "cancel";
                label = " Cancel ";
                is_default = false;
                is_cancel = true;
              }
            }
          }

}
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE3.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. (Tiếp theo ta sẽ lấy một phiên bản của chương trình Autolisp mẫu và chỉnh sửa lại. Tất cả mã mới được thể hiện bởi màu đỏ)

```
(defun C:SAMPLE3()
```

```
  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE3.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE3" dcl_id))
    (progn
      (alert "The SAMPLE3.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)") ;Chú ý hàm SaveVars
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample3 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE3.LSP ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

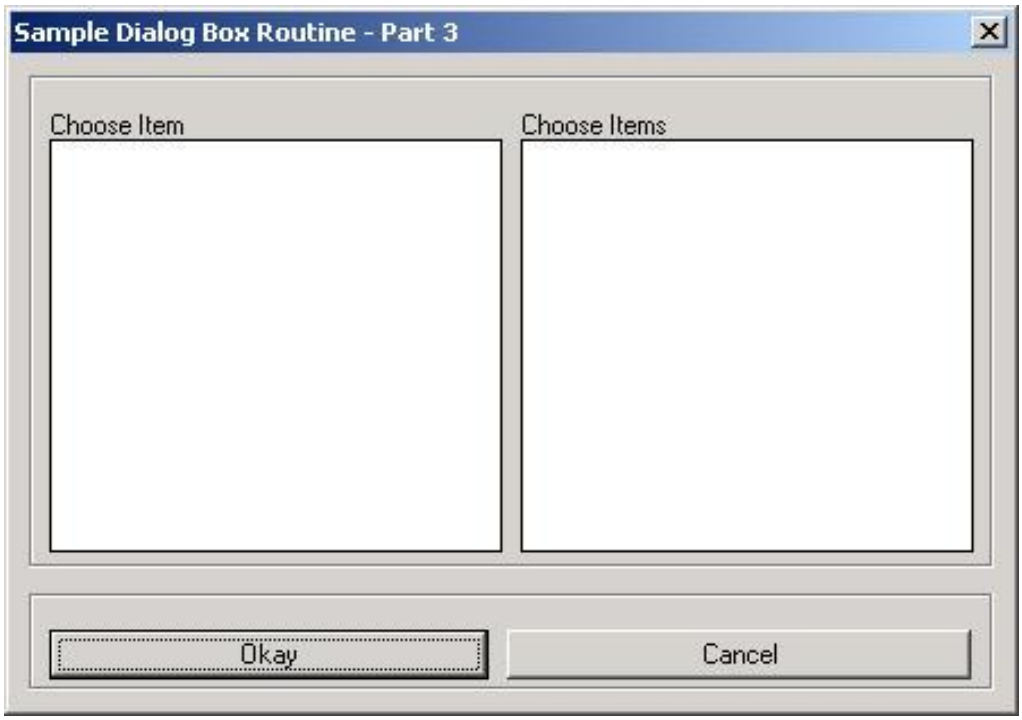
Let's load the program and see what the DCL file looks like. On the command line type this:

Command: `(load "sample3")` and press enter

You should see this

C:Sample3
Command:

Now type Sample3 and press enter. If everything went according to plan you should see this on your screen:



Looks good but, there is nothing to choose. Let's add some data to the list boxes. We will need two list. We will call the list for the first list box myList1 and the second myList2. (Trông đẹp đấy nhưng chưa có gì để chọn. Ta sẽ thêm dữ liệu vào hộp danh sách. Ta cần hai danh sách. Ta sẽ gọi hàm list để tạo danh sách thứ nhất là myList1 và danh sách thứ hai là myList2.)

```
(setq myList1(list "Electrical" "Structural" "Plumbing" "Foundation"))

(setq myList2(list "Plastic" "Steel" "Aluminum" "Concrete"))
```

Alrighty then, we have our list built. All we have to do is put them in the dialog box. We will use the start_list, add_list, and end_list functions. Start_list tells DCL which list_box we are going to edit. The identification is made by using the list_box KEY. The first list has a key of "mylist1" and the second has a key of "mylist2". So the start_list function would look like this:

Vậy là ta đã tạo các danh sách của mình. Điều phải làm là đặt nó vào trong hộp danh sách. Ta sẽ sử dụng các hàm start_list, add_list và end_list. Hàm start_list sẽ bảo DCL hộp danh sách nào sẽ phải sửa lại. Việc chỉ định được thực hiện bằng cách sử dụng khóa của hộp danh sách. Danh sách thứ nhất có khoá là "mylist1" và danh sách thứ hai có khoá là "mylist2". Do đó hàm start_list sẽ như sau:

```
(start_list "mylist1" 3) ;The 3 means we want to delete the old contents and start new. (Tham
                          số 3 có nghĩa là ta muốn xoá các nội dung cũ và bắt đầu danh sách mới)
```

Next we use the add_list function to tell DCL which list to put in the list_box. We use the mapcar function to apply add_list to each member in the list. Our list for the first list_box is named myList1. So... (Tiếp theo sử dụng hàm add_list để bảO DCL danh sách nào sẽ được đặt vào trong hộp danh sách. Ta sử dụng hàm mapcar để áp dụng hàm add_list cho mỗi thành viên trong danh sách đó. Danh sách của chúng ta dùng cho hộp danh sách thứ nhất là myList1. Vì thế ta có:

```
(mapcar 'add_list myList1)
```

Finally we use the end_list function to tell DCL to display the new contents because we are through editing the list. (Cuối cùng ta dùng hàm End_list để bảO DCL hiển thị nội dung mới vì ta đã sửa xong danh sách đó)

```
(end_list)
```

To look at it all together: Hãy xem toàn bộ công việc này:

```
(start_list "mylist1" 3)
(mapcar 'add_list myList1)
(end_list)

(start_list "mylist2" 3)
(mapcar 'add_list myList2)
(end_list)
```

Let's add all of this to the AutoLisp program and see what it looks like. (Thêm các mã này vào chương trình)

```
(defun C:SAMPLE3()

 (setq myList1(list "Electrical" "Structural" "Plumbing" "Foundation"))
 (setq myList2(list "Plastic" "Steel" "Aluminum" "Concrete"))

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE3.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE3" dcl_id))
    (progn
      (alert "The SAMPLE3.DCL file could not be loaded!")
      (exit)
    )
  )

  (start_list "mylist1" 3)
  (mapcar 'add_list myList1)
  (end_list)

  (start_list "mylist2" 3)
  (mapcar 'add_list myList2)
  (end_list)

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)") ;Chú ý hàm SaveVars
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
```

```
    (princ "\n Sample3 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Notice the location of the red lines. We create the list before loading the dialog box. We add the list to the dialog box after it is loaded with new_dialog and before the action_tile statements. This is the order you should use.

Chú ý tới vị trí của các dòng màu đỏ. Ta tạo danh sách trước khi tải hộp thoại. Ta thêm danh sách vào hộp thoại sau khi nó được tải với hàm new_dialog và trước hàm thông báo phần tử hanh động. Đây là trật tự bạn nên dùng

---

Looking good so far. We need to add the SaveVars function to save the selected items from the list boxes when the Okay button is pressed. *Look at the blue text in the Sample3.lsp program above.*

Tiếp tục thêm chút nữa. Ta cần thêm hàm SaveVars để lưu các khoản mục được chọn từ hộp danh sách khi nút Okay được nhấn. *Xem dòng chữ màu xanh trong chương trình Autolisp Sample3.lsp ở trên.*

Let's steal the saveVars routine from the list_box control on the "List and how to handle them" page of this tutorial and modify it. I'll show the modifications in red.

Ta sẽ ăn cắp vòng lưu biến Savevars từ chương trình điều khiển hộp danh sách trong trang "Danh sách và cách điều khiển chúng" của tài liệu tự học này và sửa đổi nó. Tôi sẽ thể hiện phần sửa đổi bằng màu đỏ.

```
(defun saveVars(/ readlist count item)

  ;;;--- Setup a list to hold the selected items
  (setq retList(list))

  ;;;--- Save the list setting
  (setq readlist(get_tile "mylist1"))

  ;;;--- Setup a variable to run through the list
  (setq count 1)

  ;;;--- cycle through the list getting all of the selected items
  (while (setq item (read readlist))
    (setq retlist(append retList (list (nth item myList1))))
    (while
      (and
        (/= " " (substr readlist count 1))
        (/= ""   (substr readlist count 1))
```

```
      )
      (setq count (1+ count))
    )
    (setq readlist (substr readlist count))
  )
)
```

Wow! That was easy. Wait a minute, we have two list boxes. We will have to create a function out of this or simply copy this and do it twice. For now, let's just do it twice. (Ồ , quá dễ rồi. Hượm đã, ta có hai hộp danh sách. Ta sẽ phải tạo một hàm bao gồmcả hai hộp danh sách này hoặc đơn giản là copy nó thành hai lần. Bây giờ ta sẽ làm như vậy.)

Our program would now look like this: (Chương trình của chúng ta sẽ trông giống như sau:)

```
(defun saveVars(/ readlist count item)

  ;;;--- Setup a list to hold the selected items
  (setq retList(list))

  ;;;--- Save the list setting
  (setq readlist(get_tile "mylist1"))

  ;;;--- Setup a variable to run through the list
  (setq count 1)

  ;;;--- cycle through the list getting all of the selected items
  (while (setq item (read readlist))
    (setq retlist(append retList (list (nth item myList1))))
    (while
      (and
        (/= " " (substr readlist count 1))
        (/= ""   (substr readlist count 1))
      )
      (setq count (1+ count))
    )
    (setq readlist (substr readlist count))
  )

  ;;;--- Setup a list to hold the selected items
  (setq retList2(list))

  ;;;--- Save the list setting
  (setq readlist(get_tile "mylist2"))

  ;;;--- Setup a variable to run through the list
  (setq count 1)

  ;;;--- cycle through the list getting all of the selected items
  (while (setq item (read readlist))
    (setq retlist2(append retList2 (list (nth item myList2))))
    (while
      (and
        (/= " " (substr readlist count 1))
        (/= ""   (substr readlist count 1))
      )
      (setq count (1+ count))
```

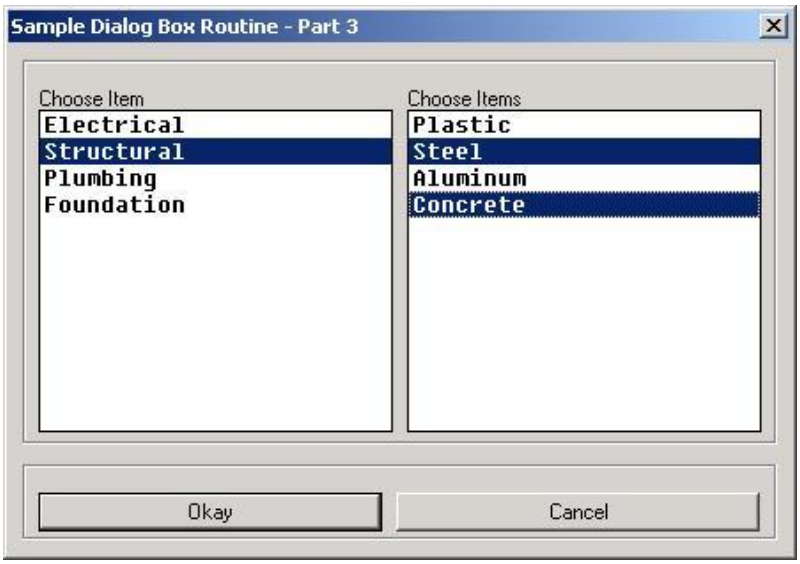
```
    )
    (setq readlist (substr readlist count))
  )
)

(defun C:SAMPLE3()

 (setq myList1(list "Electrical" "Structural" "Plumbing" "Foundation"))
 (setq myList2(list "Plastic" "Steel" "Aluminum" "Concrete"))

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE3.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE3" dcl_id))
    (progn
      (alert "The SAMPLE3.DCL file could not be loaded!")
      (exit)
    )
  )

  (start_list "mylist1" 3)
  (mapcar 'add_list myList1)
  (end_list)

  (start_list "mylist2" 3)
  (mapcar 'add_list myList2)
  (end_list)

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample3 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Last item. We need to replace the line in the program: `(princ "\n The user pressed Okay!")` with

something to modify and display the selected items. Let's do something simple. We will tell the user what was selected out of each list.. (Khoản cuối cùng. Ta cần thay thế dòng (princ "\n The user pressed Okay!") trong chương trình bằng điều gì đó để sửa đổi và hiển thị các khoản mục được chọn. Ta hãy làm điều đơn giản sau. Bảo cho người sử dụng biết cái gì đã được chọn ra từ mỗi một danh sách.)

```
;;;--- If the user pressed the Okay button
(if(= ddiag 2)
  (progn

    ;;;--- Inform the user of his selection from the first list
    (princ (strcat "\n You chose " (car retList) " from the first list box."))

    ;;;--- Inform the user of his selections from the second list
    (princ "\n Your choice(s) from the second list box :")
    (foreach a retList2
      (princ "\n ")
      (princ a)
    )
  )
)
```

Add the above to the file, save it and test it out. Everything working okay? (Thêm những điều trên vào tập tin, lưu lại và chạy thử. Mọi thứ làm việc tốt cả chứ?)




```
Command: (load "sample3")
C:SAMPLE3

Command: Sample3

 You chose Structural from the first list box.
 Your choice(s) from the second list box :
 Steel
 Concrete

Command:
```

When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE3()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

That's it. We're done.

Back

Dialog Control Language - Part 4 (Phần thực hành 4 về DCL)

Part 4 - PopUp_List (Phần 4- Danh sách thả xuống)

Let's build a working DCL file showing us exactly how to handle popup list. (Ta sẽ xây dựng một tập tin DCL hoạt động để chỉ cho thấy chính xác cách điều khiển danh sách thả xuống)

We will build a DCL file containing two pop_up list plus an Okay and Cancel button. The selected items will be displayed on the screen after the user presses the Okay button. (Ta sẽ tạo một tập tin DCL chứa hai danh sách thả xuống với bộ nút Okay và Cancel. Các khoản mục được chọn sẽ hiển thị trên màn hình sau khi người sử dụng nhấn nút Okay)

Layout thoughts: I will place the popup_list in a row, (side by side). Then I'll put the Okay and Cancel buttons in a row at the bottom of the dialog box. So...I'll need something like this:

Thiết kế hộp thoại: Tôi sẽ đặt các danh sách thả trong một hàng, (nằm cạnh nhau). Sau đó tôi đặt các nút Okay và Cancel vào hàng dưới cùng của hộp thoại. Vì thế tôi sẽ cần như sau:

```
: column {
  : boxed_row {
    : popup_list {
      // Put code for popup_list 1 here (Đặt mã cho danh sách thả xuống 1 ở đây)
    }
    : popup_list {
      // Put code for popup_list 2 here (Đặt mã cho danh sách thả xuống 2 ở đây)
    }
  }
  : boxed_row {
    : button {
       // Put code for the Okay button here  (Đặt mã cho nút Okay ở đây)
    }
```

```
    : button {
       // Put code for the Cancel button here   (Đặt mã cho nút Cancel ở đây)
    }
  }
}
```

Let's copy in the code for the header and all of the controls above from the "Controls" section of this tutorial. I'll show them in red. Notice the key names and labels had to be changed. (Ta hãy copy các mã phần tiêu đề và tất cả các nút điều khiển trên đây từ phần "Các nút điều khiển" của tài liệu tự học này. Tôi sẽ trình bày chúng bằng màu đỏ. Chú ý là các tên khoá và các nhãn đã được thay đổi.)

```
SAMPLE4 : dialog {
          label = "Sample Dialog Box Routine - Part 4";
          : column {
            : boxed_row {
              : popup_list {
                key = "mylist1";
                label = "Select Item";
                fixed_width_font = true;
                width = 30;
                value = "";
              }
              : popup_list {
                key = "mylist2";
                label = "Select Item";
                fixed_width_font = true;
                width = 30;
                value = "";
              }
            }
            : boxed_row {
              : button {
                key = "accept";
                label = " Okay ";
                is_default = true;
              }
              : button {
                key = "cancel";
                label = " Cancel ";
                is_default = false;
                is_cancel = true;
              }
            }
          }

}
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE4.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. (Tiếp theo ta sẽ lấy một phiên bản của chương trình Autolisp mẫu và sửa đổi lại nó. Tất cả mã mới đều thể hiện bởi màu đỏ.)

```
(defun C:SAMPLE4()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE4.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE4" dcl_id))
    (progn
      (alert "The SAMPLE4.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)") ; Chú ý hàm SaveVars
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample4 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE4.LSP ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Let's load the program and see what the DCL file looks like. On the command line type this:

Command: `(load "sample4")` and press enter

You should see this

C:Sample4
Command:

Now type Sample4 and press enter. If everything went according to plan you should see this on your screen:

Looks good but, there is nothing to choose. Let's add some data to the popup list boxes. We will need two list. We will call the list for the first popup list box myList1 and the second myList2. (Trông thì ngon đấy nhưng chưa có gì để chọn cả. Ta sẽ thêm dữ liệu vào các hộp danh sách thả xuống. Ta cần hai danh sách. Gọi danh sách cho hộp danh sách thả xuống thứ nhất là myList1 và cho hộp thứ hai là myList2.)

```
(setq myList1(list "Electrical" "Structural" "Plumbing" "Foundation"))

(setq myList2(list "Plastic" "Steel" "Aluminum" "Concrete"))
```

Alrighty then, we have our list built. All we have to do is put them in the dialog box. We will use the start_list, add_list, and end_list functions. Start_list tells DCL which popup list box we are going to edit. The identification is made by using the popup list box KEY. The first popup list box has a key of "mylist1" and the second has a key of "mylist2". So the start_list function would look like this: (Rồi, vậy là ta đã tạo xong các danh sách. Ta chỉ việc đặt chúng vào hộp thoại. Ta sẽ sử dụng các hàm start_list, add_list, và end_list. Hàm start_list dùng để xác định hộp danh sách nào sẽ được chỉnh sửa. Để chỉ định hộp, sử dụng khoá của hộp. Hộp danh sách thả xuống thứ nhất có khóa là "mylist1" và hộp thứ hai có khoá là "mylist2". Vì thế hàm start_list sẽ như sau:

```
(start_list "mylist1" 3)  ; The 3 means we want to delete the old contents and start new.
```
(Tham số 3 có nghĩa là ta sẽ xóa toàn bộ nội dung cũ và bắt đầu danh sách mới.)

Next we use the add_list function to tell DCL which list to put in the popup list box. We use the mapcar function to apply add_list to each member in the list. Our list for the first popup list box is named myList1. So... (Tiếp theo ta sử dụng hàm add_list để xác định mục nào cần đặt vào trong hộp danh sách thả xuống. Ta sử dụng hàm mapcar để áp dụng hàm ADD_list cho từng thành viên của danh sách. Danh sách ta dùng cho hộp danh sách thả xuống thứ nhất là myList1. Vì thế…)

```
(mapcar 'add_list myList1)
```

Finally we use the end_list function to tell DCL to display the new contents because we are through editing the popup list box. (Cuối cùng ta sử dụng hàm end_list để hiển thị nội dung mới của hộp danh sách thả xuống vì ta đã chỉnh sửa xong.)

```
(end_list)
```

To look at it all together: (Gộp chúng lại với nhau:)

```
(start_list "mylist1" 3)
(mapcar 'add_list myList1)
(end_list)

(start_list "mylist2" 3)
(mapcar 'add_list myList2)
(end_list)
```

Let's add all of this to the AutoLisp program and see what it looks like. (Thêm tất cả các mã trên vào chương trình Autolisp và ta thấy như sau:

```
(defun C:SAMPLE4()

 (setq myList1(list "Electrical" "Structural" "Plumbing" "Foundation"))
 (setq myList2(list "Plastic" "Steel" "Aluminum" "Concrete"))

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE4.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE4" dcl_id))
    (progn
      (alert "The SAMPLE4.DCL file could not be loaded!")
      (exit)
    )
  )

  (start_list "mylist1" 3)
  (mapcar 'add_list myList1)
  (end_list)

  (start_list "mylist2" 3)
  (mapcar 'add_list myList2)
  (end_list)

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample4 cancelled!")
  )
```

```
  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Notice the location of the red lines. We create the list before loading the dialog box. We add the list to the dialog box after it is loaded with new_dialog and before the action_tile statements. This is the order you should use. (Chú ý tới vị trí của các dòng màu đỏ. Ta tạo danh sách trước khi tải hộp thoại. Thêm danh sách vào hộp thoại sau khi nó đã được tải bởi hàm new_dialog và trước hàm thông báo phần tử hành động action_tile. Đó là trật tự mà bãn nên sử dụng.)

Looking good so far. We need to add the SaveVars function to save the selected items from the popup list boxes when the Okay button is pressed. *Look at the blue text in the Sample4.lsp program above.*(Tiếp tục nữa nhé. Ta cần phải thêm hàm SaveVars để lưu lại các khoản mục đã được chọn từ hộp danh sách thả xuống khi nút Okay bị nhấn. *Hãy xem dòng chữ màu xanh trong chương trình Autolisp Sample4.lsp ở trên.)*

Let's steal the saveVars routine from the popup list box control on the "Saving data from the dialog box" page of this tutorial and modify it. I'll show the modifications in red. (Ta sẽ chôm vòng lưu biến SaveVars từ mục điều khiển hộp danh sách thả xuống trong phần "Lưu dữ liệu từ hộp thoại" của tài liệu này và sửa đổi lại. Tôi sẽ thể hiện các phần sửa đổi bằng màu đỏ.)

```
(defun saveVars()

  ;;;--- Get the selected item from the first list
  (setq sStr1(get_tile "mylist1"))

  ;;;--- Make sure something was selected...
  (if(= sStr1 "")>
    (setq myItem1 nil)
    (setq myItem1 (nth (atoi sStr1) myList1))
  )

)
```

Wow! That was easy. Wait a minute, we have two pop_up list boxes. We will have to create a function out of this or simply copy this and do it twice. For now, let's just do it twice. (Ồ, dễ quá nhỉ. Khoan đã đừng tưởng bở. Ta có hai hộp danh sách thả xuống cơ mà. Ta sẽ phải tạo một hàm cho cả hai hộp danh sách này hay đơn giản hơn ta copy điều này hai lần. Ở đây ta sẽ thực hiện hai lần copy.)

Our program would now look like this: (Chương trình của ta trông giống như sau:)

```
(defun saveVars()

  ;;;--- Get the selected item from the first list
  (setq sStr1(get_tile "mylist1")) ;;; Hàm (get_tile "mylist1") trả về số thứ tự của
```

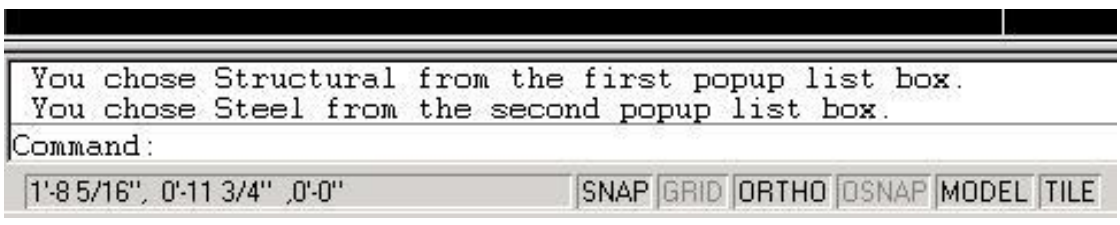
```
khoản mục được chọn.

  ;;;--- Make sure something was selected...
  (if(= sStr1 "")>  ;;; === Symbol ">" is excess and needs to obmit ===;;;
    (setq myItem1 "Nothing")
    (setq myItem1 (nth (atoi sStr1) myList1))
  )

  ;;;--- Get the selected item from the second list
  (setq sStr2(get_tile "mylist2"))

  ;;;--- Make sure something was selected...
  (if(= sStr2 "")>  ;;; === Symbol ">" is excess and needs to obmit ===;;;
    (setq myItem2 "Nothing")
    (setq myItem2 (nth (atoi sStr2) myList2))
  )
)

(defun C:SAMPLE4()

 (setq myList1(list "Electrical" "Structural" "Plumbing" "Foundation"))
  (setq myList2(list "Plastic" "Steel" "Aluminum" "Concrete"))

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE4.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE4" dcl_id))
    (progn
      (alert "The SAMPLE4.DCL file could not be loaded!")
      (exit)
    )
  )

  (start_list "mylist1" 4)
  (mapcar 'add_list myList1)
  (end_list)

  (start_list "mylist2" 4)
  (mapcar 'add_list myList2)
  (end_list)

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample4 cancelled!")
  )
```

```
  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Last item. We need to replace the line in the program: `(princ "\n The user pressed Okay!")` with something to modify and display the selected items. Let's do something simple. We will tell the user what was selected out of each popup list.. (Cuối cùng, ta cần thay thế dòng (princ "\n The user pressed Okay!") bằng cái gì đó để sửa đổi và hiển thị các khoản mục được chọn. Hãy làm điều đơn giản này. Báo cho người sử dụng biết cái gì đã được chọn ra từ danh sách thả xuống)

```
  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn

      ;;;--- Inform the user of his selection from the first list
      (princ (strcat "\n You chose " myItem1 " from the first popup list box."))

      ;;;--- Inform the user of his selections from the second list
      (princ (strcat "\n You chose " myItem2 " from the second popup list box."))

    )
  )
```

Add the above to the file, save it and test it out. Everything working okay? (Bổ sung những điều này vào tập tin, lưu lại và chạy thử nó. Mọt thứ làm việc tốt cả chứ ?)



When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE4()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

That's it. We're done.

Back

Dialog Control Language - Part 5 (Phần thực hành 5 về DCL)

Part 5 - Radio Buttons (Phần 5 – Các nút lựa chọn)

Let's build a working DCL file showing us exactly how to handle radio buttons. (Ta sẽ xây dựng một tập tin DCL hoạt động để thể hiện chính xác cách điều khiển các nút lựa chọn)

The first thing you need to know about a radio button is how stupid they are. They have no brains. They do not know what the other radio buttons are doing. You can layout six radio buttons and select everyone of them like they were toggles. That's not the way we use radio buttons. They are supposed to be smart. They should know what to do if a radio button around them is selected. They should turn themselves off because only one radio button in a group is supposed to be checked. That's where radio_column and radio_row come in to play. They are the brains for the radio buttons. They watch all the buttons in their row or column to make sure only one is turned on. Okay..moving on. (Trước tiên, bạn cần biết là các nút lựa chọn rất ngu ngốc. Chúng không có óc. Chúng không cần biết tới các nút lựa chọn khác đang làm gì. Bạn có thể bố trí sáu nút lựa chọn và chọn một trong số đó như kiểu bật hay tắt chúng. Đó không phải cách mà ta sử dụng các nút lựa chọn. Chúng có nhiệm vụ là linh hoạt. Chúng phải biết làm gì nếu một nút lựa chọn quanh chúng được lựa chọn. Chúng sẽ tự tắt bởi vì chỉ duy nhất một nút lựa chọn trong nhóm có nhiệm vụ kiểm soát. Đó là nơi để các cột lựa chọn và các hàng lựa chọn hoạt động. Chúng là bộ óc của các nút lựa chọn. Chúng theo dõi tất cả các nút lựa chọn trong hàng hay cột của chúng để đảm bảo rằng chỉ có duy nhất một nút được kích hoạt. Rồi, tiếp tục.

We will build a DCL file containing 4 radio_buttons plus an Okay and Cancel button. The selected item will be displayed on the screen after the user presses the Okay button. (Ta sẽ tạo một tập tin DCL chứa bốn nút lựa chọn với bộ nút Okay và Cancel. Khoản mục được chọn sẽ được hiển thị trên màn hình sau khi người sử dụng nhấn nút Okay)

Layout thoughts: I will place the radio_buttons in a column, (stacked on top of each other). Then I'll put the Okay and Cancel buttons in a row at the bottom of the dialog box. So...I'll need something like this:
Thiết kế hộp thoại: Tôi sẽ đặt các nút lựa chọn trong một cột (Các nút chồng lên nhau). Rồi đặt các nút Okay và Cancel trong một hàng ở đáy hộp thoại. Vậy nên tôi sẽ cần một thứ như sau:

```
: column {
  : radio_column {
    // Put code for radio_column here (Đặt mã của cột lựa chọn ở đây)
    : radio_column {
      // Put code for radio_button 1 here (Đặt mã của nút lựa chọn 1 ở đây)
    }
```

```
    : radio_button {
      // Put code for radio_button 2 here   (Đặt mã của nút lựa chọn 2 ở đây)
    }
    : radio_button {
      // Put code for radio_button 3 here   (Đặt mã của nút lựa chọn 3 ở đây)
    }
    : radio_button {
      // Put code for radio_button 4 here   (Đặt mã của nút lựa chọn 4 ở đây)
    }
  }
  : boxed_row {
    : button {
       // Put code for the Okay button here   (Đặt mã của nút Okay ở đây)
    }
    : button {
       // Put code for the Cancel button here    (Đặt mã của nút Cancel ở đây)
    }
  }
}
```

Let's copy in the code for the header and all of the controls above from the "Controls" section of this tutorial. I'll show them in red. Notice the key names and labels had to be changed. (Ta sẽ copy các mã của phần tiêu đề và tất cả các nút điều khiển trên đây từ phần "Các nút điều khiển" của tài liệu này. Tôi sẽ thể hiện chúng bằng màu đỏ. Chú ý rằng các tên khoá và các nhãn đã được thay đổi.)

```
SAMPLE5 : dialog {
          label = "Sample Dialog Box Routine - Part 5";
          : column {
            : radio_column {
              key = "mychoice";
              : radio_button {
                key = "but1";
                label = "Apples";
              }
              : radio_button {
                key = "but2";
                label = "Oranges";
              }
              : radio_button {
                key = "but3";
                label = "Bananas";
              }
              : radio_button {
                key = "but4";
                label = "Lemons";
              }
            }
            : boxed_row {
```

```
                key = "accept";
                label = " Okay ";
                is_default = true;
              }
              : button {
                key = "cancel";
                label = " Cancel ";
                is_default = false;
                is_cancel = true;
              }
            }
          }

}
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE5.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. Tiếp theo ta lấy một phiên bản của chương trình Autolisp mẫu và sửa đổi lại nó. Tất cả các mã mới được thể hiện màu đỏ.

```
(defun C:SAMPLE5()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE5.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE5" dcl_id))
    (progn
      (alert "The SAMPLE5.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample5 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
```

```
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE5.LSP ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

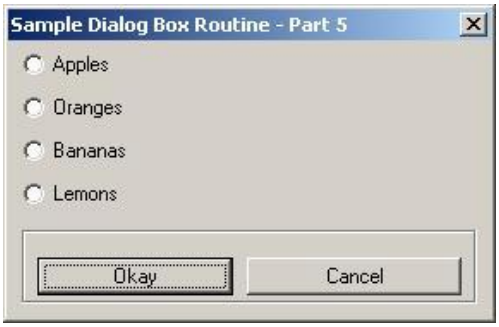
Let's load the program and see what the DCL file looks like. On the command line type this:

Command: `(load "sample5")` and press enter

You should see this

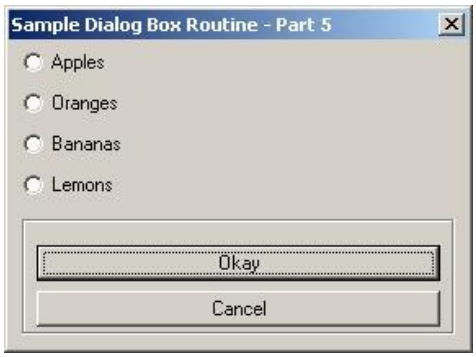
C:Sample5
Command:

Now type Sample5 and press enter. If everything went according to plan you should see this on your screen:



That doesn't look very good does it? Let's change the boxed_row into a boxed_column in our DCL file. *(See the blue text in the DCL file above)* Make the changes then Save the Sample5.DCL file. No need to load the autolisp program again, it's loaded. Just run the Sample5 program again. Now it should look like this:

Trông nó không đẹp lắm phải không? Ta sẽ thay đổi hàng hộp thành cột hộp trong tập tin DCL.*(Xem dòng chữ màu xanh trong tập tin DCL ở trên).*Thực hiện sự thay đổi trong tập tin Sample5.DCL. Không cần tải lại nó vì nó đã được tải rồi.Chỉ cần chạy chương trình Sample5.lsp một lần nữa. Bây giờ kết quả sẽ như sau:



It still doesn't look right. It's our label "Sample Dialog Box Routine - Part 5" that is causing the problem. Let's shorten it to "SDBR - Part 5" and try it again: (Trông vẫn chưa đẹp lắm. Nguyên nhân là do cái nhãn "Sample Dialog Box Routin – Part 5" của ta quá dài. Cắt ngắn nó thành "SDBR – Part 5" và thử lại:)

Looks better! (Trông bô trai rồi chứ?)

Looking good so far. We need to add the SaveVars function to save the selected items from the radio_column when the Okay button is pressed. *Look at the blue text in the Sample5.lsp program above.*(Ta hãy rặn thêm tí nữa nha. Ta cần thêm hàm SaveVars để lưu lại các khoản mục được chọn từ cột lựa chọn khi nút Okay bị nhấn. *Xem dòng chữ màu xanh trong chương trình Sample.lsp ở trên.*

Let's steal the saveVars routine from the radio_column control on the "Saving data from the dialog box" page of this tutorial and modify it. I'll show the modifications in red. (Chôm vòng lưu biến SaveVars từ mục điều khiển cột lựa chọn trên trang "Lưu dữ liệu từ hộp thoại" của tài liệu này và sửa đổi lại. Tôi sẽ thể hiện những phần sửa đổi bằng màu đỏ.)

We can do this two different ways. We can check each radio_button to find out which one is on or we can check the entire column of radio_buttons by getting the value of the radio_column. (Ta có thể làm điều này bằng hai cách khác nhau. Ta sẽ kiểm tra từng nút lựa chọn để xác định nút nào được bật hoặc ta có thể kiểm tra cả cột các nút lựa chọn bằng cách lấy giá trị của cột lựa chọn.)

First method: Checking the Radio_Column: (Phương pháp 1: Kiểm tra Cột lựa chọn)

```
(defun saveVars()

  ;;;--- Get the key of the choice made
  ;;;     [ returns "but1" "but2" "but3" or "but4" whichever is selected.]
```

```
  (setq myChoice(get_tile "mychoice"))

)
```

Second method: Checking each Radio_Button: (Phương pháp 2: Kiểm tra từng nút lựa chọn)

```
(defun saveVars()

;;;--- Get the value of each item
  (setq choice1(atoi(get_tile "but1")))   // 0 = not chosen   1 = chosen
  (setq choice2(atoi(get_tile "but2")))   // 0 = not chosen   1 = chosen
  (setq choice3(atoi(get_tile "but3")))   // 0 = not chosen   1 = chosen
  (setq choice4(atoi(get_tile "but4")))   // 0 = not chosen   1 = chosen

)
```

Wow! That was easy. So...Which one do we use? For this tutorial, let's use both. Why not? (Ô, quá dễ nhỉ. Vậy bạn muốn sử dụng cách nào? Để học,ta sẽ chơi cả hai thứ. Sao lại không nhỉ?

```
(defun saveVars()

  ;;;--- Get the key of the choice made
  ;;;     [ returns "but1" "but2" "but3" or "but4" whichever is selected.]

  (setq myChoice(get_tile "mychoice"))

  ;;;--- Get the value of each item
  (setq choice1(atoi(get_tile "but1")))   // 0 = not chosen   1 = chosen
  (setq choice2(atoi(get_tile "but2")))   // 0 = not chosen   1 = chosen
  (setq choice3(atoi(get_tile "but3")))   // 0 = not chosen   1 = chosen
  (setq choice4(atoi(get_tile "but4")))   // 0 = not chosen   1 = chosen

)
```

Add this to the original Sample5.lsp program and we should have something that looks like this: (Thêm điều đó vào chương trình Sample5.lsp và ta sẽ có chương trình như SAU:

```
(defun saveVars()

  ;;;--- Get the key of the choice made
  ;;;     [ returns "but1" "but2" "but3" or "but4" whichever is selected.]

  (setq myChoice(get_tile "mychoice"))

  ;;;--- Get the value of each item
  (setq choice1(atoi(get_tile "but1")))   // 0 = not chosen   1 = chosen
  (setq choice2(atoi(get_tile "but2")))   // 0 = not chosen   1 = chosen
  (setq choice3(atoi(get_tile "but3")))   // 0 = not chosen   1 = chosen
  (setq choice4(atoi(get_tile "but4")))   // 0 = not chosen   1 = chosen

)
```

```
(defun C:SAMPLE5()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE5.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE5" dcl_id))
    (progn
      (alert "The SAMPLE5.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample5 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Last item. We need to replace the line in the program: `(princ "\n The user pressed Okay!")` with something to display the selected item. (Cuối cùng, ta cần thay thế dòNG (princ "\n The user pressed Okay!") bằng cái gì đó để hiển thị khoản mục được chọn.)

```
 ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn

      ;;;--- Inform the user of his selection using the radio_column data
      (princ "\n Using Radio_column data...You chose ")
      (cond
        ((= myChoice "but1")(princ "Apples!"))
        ((= myChoice "but2")(princ "Oranges!"))
        ((= myChoice "but3")(princ "Bananas!"))
        ((= myChoice "but4")(princ "Lemons!"))
      )

      ;;;--- Inform the user of his selection using the radio_buttons data
```

```
      (princ "\n Using Radio_buttons data...You chose ")
      (cond
        ((= Choice1 1)(princ "Apples!"))
        ((= Choice2 1)(princ "Oranges!"))
        ((= Choice3 1)(princ "Bananas!"))
        ((= Choice4 1)(princ "Lemons!"))
      )

    )
  )
```

Add the above to the autolisp file, save it and test it out. Everything working okay? (Thêm những điều trên vào tập tin Autolisp, lưu lại và chạy thử chương trình. Mọi việc ổn cả chứ?)


```
Command: (load "sample5")
C:SAMPLE5

Command: sample5

 Using Radio_column data...You chose Oranges!
 Using Radio_buttons data...You chose Oranges!

Command:
```

When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE5()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

That's it. We're done.

Back

Dialog Control Language - Part 6 (Phần thực hành 6 về DCL)

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

Part 6 - Text and Toggles (Phần 6 - Văn bản và các nút kích hoạt)

Let's build a working DCL file showing us exactly how to handle text and toggles. (Ta sẽ xây dựng một tập tin DCL hoạt động để chỉ ra chính xác cách điều khiển văn bản và các nút kích hoạt)

We will build a DCL file containing 4 toggles, one text, plus a Cancel button. The selected item will be displayed on the screen in the text control.(Ta sẽ tạo một tập tin DCL gồm bốn nút kích hoạt, một nút văn bản với một nút Cancel. Khoản mục được chọn sẽ được hiển thị trên màn hình ở nút điều khiển văn bản)

Layout thoughts: I will place the text control on the top of the box. Then I'll put the toggles in a column, (stacked on top of each other). Last, I'll put the Cancel button at the bottom of the dialog box. So...I'll need something like this:

Thiết kế hộp thoại: Ta sẽ đặt nút điều khiển văn bản trên cùng của hộp thoại. Sau đó đặt các nút kích hoạt trong một cột (các nút chồng lên nhau). Cuối cùng đặt nút Cancel ở dưới cùng của hộp thoại. Do vậy ta cần một thứ giống như sau:

```
: column {
  : column {
    : text {
      // Put code for text here (Đặt mã của nút văn bản vào đây)
    }
  }
  : boxed_column {
    : toggle {
      // Put code for toggle 1 here (Đặt mã của nút kích hoạt 1 vào đây)
    }
    : toggle {
      // Put code for toggle 2 here (Đặt mã của nút kích hoạt 2 vào đây)
    }
    : toggle {
      // Put code for toggle 3 here (Đặt mã của nút kích hoạt 3 vào đây)
    }
    : toggle {
      // Put code for toggle 4 here (Đặt mã của nút kích hoạt 4 vào đây)
    }
  }
  : boxed_row {
    : button {
       // Put code for the Cancel button here (Đặt mã của nút Cancel vào đây)
    }
  }
}
```

Let's copy in the code for the header and all of the controls above from the "Controls" section of this tutorial. I'll show the changes that needed to be made in red. Notice the key names and labels had to be changed. (Ta sẽ copy các mã của tiêu đề và tất cả các nút điều khiển trên từ phần "Các nút điều khiển" của tài liệu này. Tôi sẽ thể hiện những sự thay đổi cần thiết bằng màu đỏ. Chú ý tên các khoá và các nhãn đã được thay đổi.)

```
            label = "Sample Dialog Box Routine - Part 6";
            : column {
              : column {
                : text {
                  key = "text1";
                   value = "Nothing selected.";
                }
              }
              : boxed_column {
                label = "Choose your lucky charms:";
                : toggle {
                  key = "tog1";
                  label = "Hearts";
                   value = "0";
                }
                : toggle {
                  key = "tog2";
                  label = "Moons";
                   value = "0";
                }
                : toggle {
                  key = "tog3";
                  label = "Stars";
                   value = "0";
                }
                : toggle {
                  key = "tog4";
                  label = "Clovers";
                   value = "0";
                }
              }
              : boxed_row {
                : button {
                  key = "cancel";
                  label = "Cancel";
                  is_default = true;
                  is_cancel = true;
                }
              }
}
```

(Ở đây thiếu mất 1 Ký tự ngoặc móc đóng kết thúc hộp thọai dialog đang mở. Chắc cụ Jeff buồn ngủ…)

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE6.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. (Tiếp theo ta lấy một phiên bản của chương trình Autolisp mẫu và sửa đổi lại nó. Tất cả các mã mới được thể hiện màu đỏ)

```
(defun C:SAMPLE6()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE6.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE6" dcl_id))
    (progn
      (alert "The SAMPLE6.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the user pressed the Cancel button
  (if(= ddiag 1)
    (princ "\n Sample6 cancelled!")
  )

  ;;;--- If the user pressed the Okay button
  (if(= ddiag 2)
    (progn
      (princ "\n The user pressed Okay!")
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Remove everything listed in orange above. We do not need an Okay button. Thus we do not need to check to see if the user pressed Okay or Cancel. We also do not need the **SaveVars** routine in this program. Remove the orange items so your program looks like the one below.

(Xoá đi tất cả các mã được thể hiện bằng màu cam. Ta không cần nút Okay. Do vậy cũng không cần kiểm soát xem liệu người sử dụng có nhấn nút Okay hay Cancel không. Ta cũng không cần tới hàm SaveVars trong chương trình này. Sau khi xoá các mã màu cam chương trình của chúng ta sẽ như sau:)

```
(defun C:SAMPLE6()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE6.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE6" dcl_id))
    (progn
      (alert "The SAMPLE6.DCL file could not be loaded!")
```

```
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "cancel" "(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE6.LSP ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Let's load the program and see what the DCL file looks like. On the command line type this: (Ta hãy tải chương trình này và xem tập tin DCL giống như thế nào. Trên dòng lệnh nhập:)

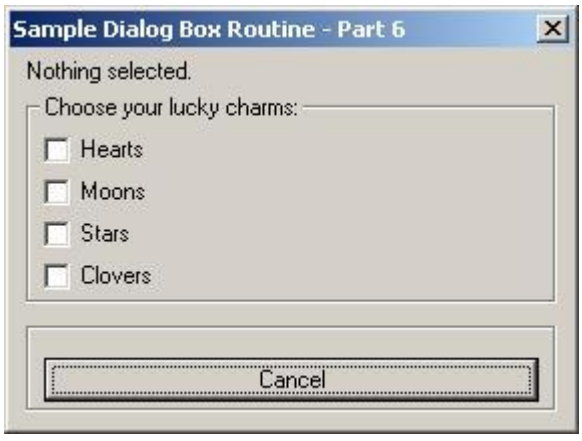
Command: `(load "sample6")` and press enter (và nhấn Enter)

You should see this (Bạn sẽ thấy như sau)

C:Sample6
Command:

Now type Sample6 and press enter. If everything went according to plan you should see this on your screen:

Bây giờ nhập Sample6 và nhấn Enter. Nếu mọi thứ tuân theo dự kiến bạn sẽ thấy trên màn hình như sau:



Doesn't look right. The dialog box is too wide. It's our label "Sample Dialog Box Routine - Part 6" that is causing the problem. I would shorten it but, my text control will need the room if everything is selected. I'll have to display "Hearts Moons Stars Clovers" all on one line. I'll leave it the way it is.

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp

Trông chưa đẹp lắm, Hộp thoại quá rộng. Nguyên nhân là do nhãn "Sample Dialog Box Routine –Part 6" quá dài. Ta có thể cắt ngắn nó, nhưng nút điều khiển văn bản của chúng ta cần một không gian đủ rộng nếu một nút được chọn. Ta sẽ phải hiển thị toàn bộ "Hearts Moons Stars Clovers" trên một dòng. Vì vậy ta cứ để nó như thế.

Notice you can select and deselect items without a problem but the text control doesn't change. We need to add the action_tiles and function to accomplish this. (Chú ý là bạn có thể chọn hay không chọn các khoản mục tùy ý mà nút điều khiển văn bản không cần thay đổi. Ta cần thêm hàm các phần tử hành động action _tiles và hàm để hoàn thiện chúng)

First let's write a routine to check each toggle and build a string representing all of the selected items.(Trước hết ta sẽ viết một vòng kiểm soát từng nút kích hoạt và tạo một chuỗi hiển thị tất cả các mục được chọn.)

```
(defun chkToggle()

  (setq tog1(atoi(get_tile "tog1")))   // 0 = not chosen   1 = chosen
  (setq tog2(atoi(get_tile "tog2")))   // 0 = not chosen   1 = chosen
  (setq tog3(atoi(get_tile "tog3")))   // 0 = not chosen   1 = chosen
  (setq tog4(atoi(get_tile "tog4")))   // 0 = not chosen   1 = chosen

  (setq myStr "")
  (if(= tog1 1)(setq myStr(strcat myStr " Hearts")))
  (if(= tog2 1)(setq myStr(strcat myStr " Moons")))
  (if(= tog3 1)(setq myStr(strcat myStr " Stars")))
  (if(= tog4 1)(setq myStr(strcat myStr " Clovers")))

  ;;;--- If nothing was selected...
  (if(= myStr "")(setq myStr "Nothing Selected!"))

  ;;;--- Now set the text control to display the string
  (set_tile "text1" myStr)
)
```

Alrighty then. I used the get_tile function to get the value of each toggle. I used the atoi function to convert that data from a string to an integer. *(I could have left it a string and checked to see if tog1 equalled "1" instead of the number 1.)* I set the variable myStr to an empty string and then appended all the checked toggle labels to it. I then changed the value of the text control by using the **set_tile** function.

Vậy là xong. Ta sử dụng hàm **get_tile** để lấy giá trị của từng nút kích hoạt . Sử dụng hàm **atoi** để đổi các dữ liệu từ chuỗi thành số nguyên. *(Ta có thể để nó ở dạng chuỗi và kiểm tra xem liệu tog1 có phải là chuỗi "1" thay vì là số 1).* Tôi đặt biến myStr là một chuỗi rỗng "" và sau đó gán tất cả các nhãn của các nút kích hoạt được kiểm soát cho nó. Tôi thay đổi giá trị của nút điều khiển văn bản bằng cách sử dụng hàm **set_tile**.

Add this to the top of your autolisp program and save it. (Thêm điều này vào bên trên chương trình của bạn và lưu lại.)

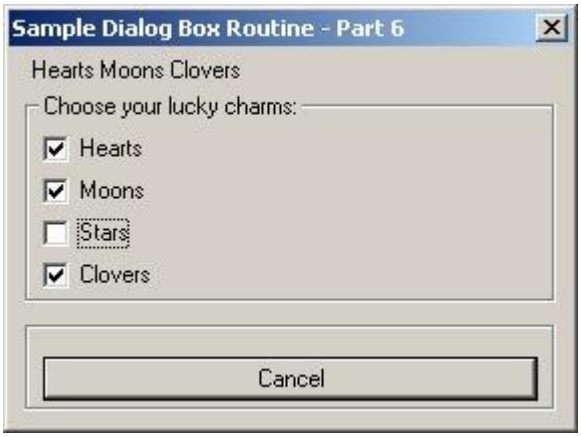
The last step is to add the action calls to the AutoLisp program. We need one action call per toggle switch. Each action call should run the chkToggle function we just created. (Bước cuối cùng là thêm các lệnh gọi hành động vào chương trình Autolisp. Ta cần một lệnh gọi hành động cho mỗi một nút kích hoạt. Mỗi lệnh gọi hành động sẽ chạy hàm chkToggle mà ta vừa tạo ra.)

```
  (action_tile "tog1" "(chkToggle)")
  (action_tile "tog2" "(chkToggle)")
  (action_tile "tog3" "(chkToggle)")
  (action_tile "tog4" "(chkToggle)")
```

Let's add this to the AutoLisp program. I'll show the new chkToggle function and the action calls in red. It should look like this: (Thêm điều này vào chương trình. Tôi sẽ thể hiện hàm chkToggle mới và các lệnh gọi hành động bởi màu đỏ. Nó sẽ trông giống như sau:)

```
(defun chkToggle()

  (setq tog1(atoi(get_tile "tog1")))   // 0 = not chosen   1 = chosen
  (setq tog2(atoi(get_tile "tog2")))   // 0 = not chosen   1 = chosen
  (setq tog3(atoi(get_tile "tog3")))   // 0 = not chosen   1 = chosen
  (setq tog4(atoi(get_tile "tog4")))   // 0 = not chosen   1 = chosen

  (setq myStr "")
  (if(= tog1 1)(setq myStr(strcat myStr " Hearts")))
  (if(= tog2 1)(setq myStr(strcat myStr " Moons")))
  (if(= tog3 1)(setq myStr(strcat myStr " Stars")))
  (if(= tog4 1)(setq myStr(strcat myStr " Clovers")))

  ;;;--- If nothing was selected...
  (if(= myStr "")(setq myStr "Nothing Selected!"))

  ;;;--- Now set the text control to display the string
  (set_tile "text1" myStr)
)

(defun C:SAMPLE6()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE6.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE6" dcl_id) ) (exit))

  ;;;--- If an action event occurs, do this function
  (action_tile "tog1" "(chkToggle)")
  (action_tile "tog2" "(chkToggle)")
  (action_tile "tog3" "(chkToggle)")
  (action_tile "tog4" "(chkToggle)")
  (action_tile "cancel" "(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- Suppress the last echo for a clean exit
  (princ)
)
```

Save it and test it out. Everything working okay? (Lưu chương trình lại và chạy thử. Mọi việc tốt chứ?)



When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE6()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

That's it. We're done.

Back

Dialog Control Language - Part 7 (Phần thực hành 7 về DCL)

Now it is time to take all of the parts and pieces and put them together. Let's build a working DCL file that will let us see most of the things we've learned and yet not get to deep into autolisp. (Bây giờ ta sẽ lấy toàn bộ các phần và các đoạn rồi ghép chúng lại với nhau. Ta sẽ xây dựng một tập tin DCL mà nó sẽ cho phép ta xem hầu hết những điều ta đã học nhưng còn chưa sâu về Autolisp)

Lets draw a polygon or circle, on a selected layer with the option to save the settings last used for defaults next time we run the program. If polygon is chosen, ask for the number of sides. So....we will need a list_box to hold all of the available layer names. We will need a radio_column to select a circle or polylgon. We will need an popup_list to hold the number of sides for the polygon. We will need a toggle to check the "Save Settings" option. And last, we will need an Okay and Cancel button.

Ta sẽ vẽ một đa giác hay một vòng tròn, trên một lớp đã chọn với tùy chọn để lưu các tham số đặt được sử dụng cuối cùng làm mặc định cho lần tiếp theo khi ta chạy chương trình. Nếu đa giác được chọn, nó sẽ yêu cầu số cạnh. Vì thế ta sẽ cần một hộp danh sách để lưu giữ tất cả các tên lớp có thể. Ta sẽ cần một cột lựa chọn để chọn đa giác hay vòng tròn. Ta cũng cần một danh sách thả xuống để lưu giữ số cạnh của đa giác. Ta cũng cần một nút kích hoạt để kiểm soát tùy chọn Save Setting. Cuối cùng ta cần một bộ nút Okay và Cancel.

Layout thoughts: The list will need to have several layers showing so the user won't have to scroll forever. So, it will probably be the tallest item on the dialog box. I'll put it in a column by itself. I'll try to fit the rest of the items in a column beside it. Then I'll put the Okay and Cancel buttons at the bottom in a

row. So...I'll need something like this:

Thiết kế hộp thoại: Danh sách cần có vài lớp thể hiện để người sử dụng không phải cuốn mãi. Vì thế nó sẽ là khoản mục lớn nhất trong hộp thoại. Tôi đặt nó vào một cột riêng. Tôi sẽ cố gắng sắp xếp các khoản mục còn lại trong một cột bên cạnh nó. Sau đó tôi đặt các nút Okay và Cancel vào hàng dưới cùng của hộp thoại. Vì vậy tôi sẽ cần một thứ giống như sau:

```
: column {
  : row {
    : boxed_column {
      : radio_column  {
        : radio_button {

          // Put code for radio button 1 here [ Circle ]
        }
        : radio_button {

          // Put code for radio button 2 here [ Polygon ]
        }
      }
      : popup_list {

         // Put code for popup_list here [ Number of Sides ]
      }
      : toggle {

         // Put code for toggle here  [ Save Settings ]
      }
    }
    : row  {
      : list_box {

         // Put code for list here [ Layer Names ]
      }
    }
  }
  : row {
    : button {

       // Put code for button 1 here [ Okay ]
    }
    : button {

       // Put code for button 2 here [ Cancel ]
    }
  }
}
```

names and labels had to be changed.(Ta sẽ copy tất cả mã cho tiêu đề và các nút điều khiển ở trên. Tôi sẽ thể hiện chúng bằng màu đỏ. Chú ý là các tên khoá và các nhãn đã được thay đổi)

```
SAMPLE7 : dialog {
          label = "Sample Dialog Box Routine - Part 7";
           : column {
             : row {
               : boxed_column {
                 : radio_column  {
                   key = "radios";
                   : radio_button {
                     label = "Draw Circle";
                     key = "drawcir";
                     value = "1";
                   }
                   : radio_button {
                     label = "Draw Polygon";
                     key = "drawpol";
                     value = "0";
                   }
                 }
                 : popup_list {
                   key = "numsides";
                   label = "Number of Sides";
                   width = 25;
                   fixed_width_font = true;
                 }
                 : toggle {
                   key = "saveset";
                    label = "Save settings";
                 }
               }
               : row  {
                 : list_box {
                   label ="Select Layer";
                   key = "layerList";
                   height = 5;
                   width = 15;
                   multiple_select = false;
                   fixed_width_font = true;
                   value = "";
                 }
               }
             }
             : row {
               : button {
                 key = "accept";
                 label = " Okay ";
                 is_default = true;
               }
```

```
            : button {
              key = "cancel";
              label = " Cancel ";
              is_default = false;
              is_cancel = true;
            }
          }
        }

}
```

Chú ý : Ở đây khóa của list_box tên "Select Layer" bị sai do khóa của list_box được xác định cả về kiểu chữ in và chữ thường .

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE.DCL ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

Next we will get a copy of the AutoLisp model and revise it. All new code is shown in red. (Bây giờ ta lấy một phiên bản của chương trình Autolisp mẫu và sửa lại nó. Tất cả mã mới được thể hiện bởi màu đỏ.)

```
(defun C:SAMPLE7()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE7.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE7" dcl_id))
    (progn
      (alert "The SAMPLE7.DCL file could not be loaded!")
      (exit)
    )
  )

  ;;;--- If an action event occurs, do this function
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the cancel button was pressed - display message
  (if (= ddiag 1)
    (princ "\n \n ...SAMPLE7 Cancelled. \n ")
  )

  ;;;--- If the "Okay" button was pressed
  (if (= ddiag 2)
    (princ "\n \n ...SAMPLE7 Complete!")
  )
```

```
  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Right click and copy the above. Open NotePad and paste it. Save the file as SAMPLE7.LSP ***Be sure to change the "Save as Type" drop down box to "All Files" before saving it or it will put a ".txt" extension on the file name. Save this file somewhere in the AutoCAD search path.***

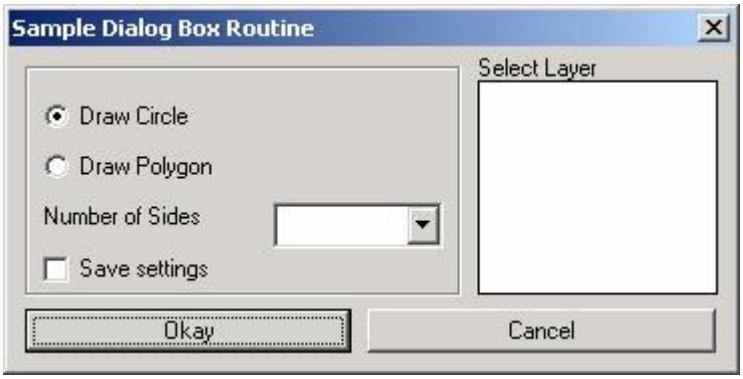
Let's load the program and see what the DCL file looks like. On the command line type this:

Command: (load "sample7") and press enter

You should see this

C:Sample7
Command:

Now type Sample and press enter. If everything went according to plan you should see this on your screen: (Bây giờ nhập Sample7 và nhấn Enter. Nếu mọi thứ được thực hiện đúng bạn sẽ thấy như sau:)



Notice there are no items in either list. We haven't added that part yet. Also notice the Draw Circle is selected. We did that in the DCL file. We set the value of that radio_button to "1". We did not set the value of the toggle, so the default is unchecked. (Chú ý là không có khoản mục nào trong các hộp danh sách. Ta vẫn chưa thêm phần này vào. Và cũng lưu ý rằng nút lựa chọn "Draw Circle" được chọn. Điều này đã được làm trong tập tin DCL. Ta đã đặt giá trị của nút lựa chọn này là "1". Ta đã không đặt giá trị cho nút kích hoạt, vì thế giá trị mặc định là không kiểm soát.)

If you push the Cancel button the program will exit normally. If you press the Okay button an error will occur because we have not defined the saveVars routine. (Nếu bạn nhấn nút Cancel chương trình sẽ thoát bình thường. Nếu bạn nhấn nút Okay, một lỗi sẽ xảy ra do ta vẫn chưa xác định vòng lưu biến SaveVars.)

We have two things left to do. First we need to build the list for the popup_list control and the list_box control. Then we need to add the list to the dialog box. (Ta có hai điều phải làm. Trước hết bạn phải tạo các danh sách cho nút điều khiển danh sách thả xuống và nút điều khiển hộp danh sách. Sau đó ta phải thêm các danh sách này vào hộp thoại.)

Instead of building a function to get all of the layer names into a list, let's just build the list manually to keep things simple. Okay...you talked me into it. I'll do it both ways and you can use the one you want. I'm going to use the short version for this tutorial. (Thay vì xây dựng một hàm để lấy tất cả các tên lớp làm danh sách, ta sẽ tạo danh sách đó một cách thủ công để cho đơn giản. Rồi, bạn bảo tôi đấy nhé. Tôi sẽ làm cả hai cách và bạn có thể sử dụng cách mà bạn muốn. Trong bài này tôi sẽ xài cách ngắn)
Long way: (Cách dài:)

```
;;;--- Set up a list to hold the layer names (lập một danh sách để lưu các tên lớp)
(setq layerList(list))

;;;--- Get the first layer name in the drawing (Lấy tên lớp đầu tiên trong bản vẽ)
(setq layr(tblnext "LAYER" T)); Chán mớ đời cái cụ Jeff này, tự dưng thò ra cái thằng tblnext mà chả thấy giới thiệu gì cả, chả biết nó con cái nhà ai mà lần, Thôi thì kệ thây nó rồi làm quen sau vậy.

;;;--- Add the layer name to the list (Thêm tên lớp vào danh sách)
(setq layerList(append layerList (list(cdr(assoc 2 layr)))))

;;;--- Step though the layer table to get all layers(Thông qua bảng lớp để lấy tất cả các lớp)
(while (setq layr(tblnext "LAYER"))

  ;;;--- Save each layer name in the list (Lưu mỗi tên lớp vào danh sách)
  (setq layerList(append layerList (list(cdr(assoc 2 layr)))))
)
```

Short Way: (Cách ngắn)

```
(setq layerList(list "0" "DIM" "HIDDEN" "STR" "TX" "WELD"))
```

We also need a list for the polgon's number of sides: (Ta cũng cần một danh sách số cạnh của đa giác:)

```
(setq numSides(list "4" "6" "8" "12" "16"))
```

We need to add these lines to our autolisp program. Add it just below the new_dialog call and above the action_tile statements.(Ta cần thêm các dòng này vào chương trình Autolisp. Thêm nó vào ngay dưới lệnh gọi hàm **new_dialog** và trước hàm thông báo phần tử hành động **action_tile)**

We now need to upload the list into the dialog box. So put these lines just below the lines you just added. (Bấy giờ ta cần đưa danh sách vào trong hộp thoại. Vì thế , đặt các dòng này ngay dưới các dòng mà bạn vừa thêm)

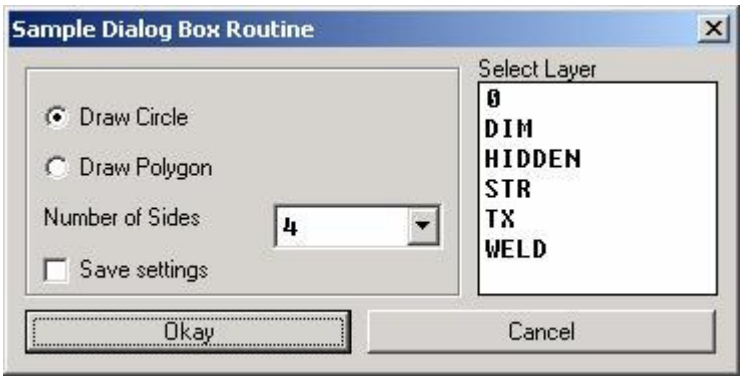
```
;;;--- Add the layer names to the dialog box (Thêm các tên lớp vào hộp thoại)
(start_list "layerlist" 3)
(mapcar 'add_list layerList)
(end_list) ; Chú ý tên khoá là "layerlist" chứ không phải là "layerList" đâu nhé

;;;--- Add the number of sides to the dialog box (Thêm số cạnh đa giác vào hộp thoại)
(start_list "numsides" 3)
(mapcar 'add_list numSides)
(end_list)
```

When you are done it should look like this: (Khi bạn làm xong chương trình sẽ như sau:)

```
(defun C:SAMPLE7()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE7.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE7" dcl_id))
    (progn
      (alert "The SAMPLE7.DCL file could not be loaded!")
      (exit)
    )
  )

  (setq layerList(list "0" "DIM" "HIDDEN" "STR" "TX" "WELD"))

  (setq numSides(list "4" "6" "8" "12" "16"))

  ;;;--- Add the layer names to the dialog box
  (start_list "layerlist" 3)
  (mapcar 'add_list layerList)
  (end_list)

  ;;;--- Add the number of sides to the dialog box
  (start_list "numsides" 3)
  (mapcar 'add_list numSides)
  (end_list)

  ;;;--- If an action event occurs, do this function
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")

  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the cancel button was pressed - display message
  (if (= ddiag 1)
    (princ "\n \n ...SAMPLE7 Cancelled. \n ")
  )

  ;;;--- If the "Okay" button was pressed
  (if (= ddiag 2)
    (princ "\n \n ...SAMPLE7 Complete!")
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Save, Load, and Run. You should see this: (Lưu lại, tải chương trình và chạy. Bạn sẽ thấy như sau:)

Gửi Mr. DUÂN – Hướng dẫn viết và sử dụng Dialog trong AutoLisp



Next, let's build the SaveVars routine. We will do this by starting with the saveVars routine in the AutoLisp Model, then copy and paste from the "Save Data from Dialog Box" section of this tutorial. We will then modify the names to match the key names in the DCL file. *I also changed a variable name to numStr to be used later to set the default settings. More on that later.* I'll show you the changes I made in red: (Tiếp theo ta sẽ xây dựng vòng lưu biến SaveVars. Ta sẽ thực hiện điều này bằng cách bắt đầu với vòng lưu biến trong chương trình Autolisp mẫu, sau đó cópy và dán từ phần "Lưu dữ liệu từ hộp thoại" của tài liệu này. Sau đó ta sẽ sửa đổi các tên cho phù hợp tên khoá trong tập tin DCL. Ta cũng thay đổi tên biến thành numStr để sử dụng về sau khi đặt các giá trị mặc định. Những điều khác sẽ nói sau. Tôi sẽ thể hiện những sự thay đổi đó bằng màu đỏ.)

```
(defun saveVars()

  (setq radios(get_tile "radios"))

  ;;;--- Get the number of sides selected from the list
  (setq numStr(get_tile "numsides"))
  (if(= numStr "")
    (setq numSides nil)
    (setq numSides(nth (atoi numStr) numSides))
  )

  ;;;--- See if the user wants to save the settings
  (setq saveSet(atoi(get_tile "saveset")))

  ;;;--- Get the selected item from the layer list
  (setq sStr(get_tile "layerlist"))

  ;;;--- If the index of the selected item is not "" then something was selected
  (if(/= sStr "")
    (progn

      ;;;--- Something is selected, so convert from string to integer
      (setq sIndex(atoi sStr))

      ;;;--- And get the selected item from the list
      (setq layerName(nth sIndex layerList))
    )

    ;;;--- Else, nothing is selected
    (progn

      ;;;--- Set the index number to -1
      (setq sIndex -1)
```

```
      ;;;--- And set the name of the selected item to nil
      (setq layerName nil)
    )
  )
)
```

Note: Since we did not specify a default value for the list_box, layerName will be set to nil if the user does not select a layer from the list before hitting the Okay button. (**Chú ý:** Do chúng ta không mô tả giá trị mặc định của hộp danh sách, tên lớp sẽ được đặt về nil khi người sử dụng không chọn một lớp nào từ danh sách lớp trước khi kích hoạt nút Okay.)

Save, Load, and Run. Check the values after pressing the Okay button by typing an exclamation point and then the variable name on the command line. Examples: To get the layer name type `!layerName` and press enter. To get the value of the toggle type `!saveSet` and press enter. Everything working okay? Alrighty then...let's move on. (Lưu lại, tải và chạy thử. Kiểm tra các giá trị sau khi nhấn nút Okay bằng cách nhập một dấu chấm than rồi đến tên biến trên dòng lệnh. Ví dụ: Để lấy giá trị biến tên lớp, nhập !layerName rồi nhấn Enter. Để lấy giá trị của nút kích hoạt, nhập !saveSet rồi nhấn Enter. Mọi thứ chạy tốt chứ? Rồi, tiếp tục…)

The only work left on the dialog box is to disable the "Number of Sides" popup_list when a circle is selected. Let's add two action_tiles on both of the radio_buttons.(Việc duy nhất còn lại với hộp thoại là tắt danh sách thả xuống "Number of sides" khi chọn vẽ vòng tròn. Ta sẽ thêm hai hàm phần tử hành động vào cả hai nút lựa chọn.)

```
;;;--- If an action event occurs, do this function
(action_tile "drawcir" "(toggleRadio 1)")
(action_tile "drawpol" "(toggleRadio 2)")
```

This will send a parameter of 1 to the toggleRadio function if the Circle is selected. It will send a parameter of 2 to the toggleRadio function if the Polygon is selected. What the hell is a toggleRadio function? (Điều này sẽ gửi tham số 1 vào hàm lựa chọn kích hoạt nếu việc vẽ vòng tròn được chọn. Gửi tham số 2 vào hàm lựa chọn kích hoạt nếu việc vẽ Đa giác được chọn. Vậy chứ hàm lựa chọn kích hoạt là thằng chó nào vậy?)

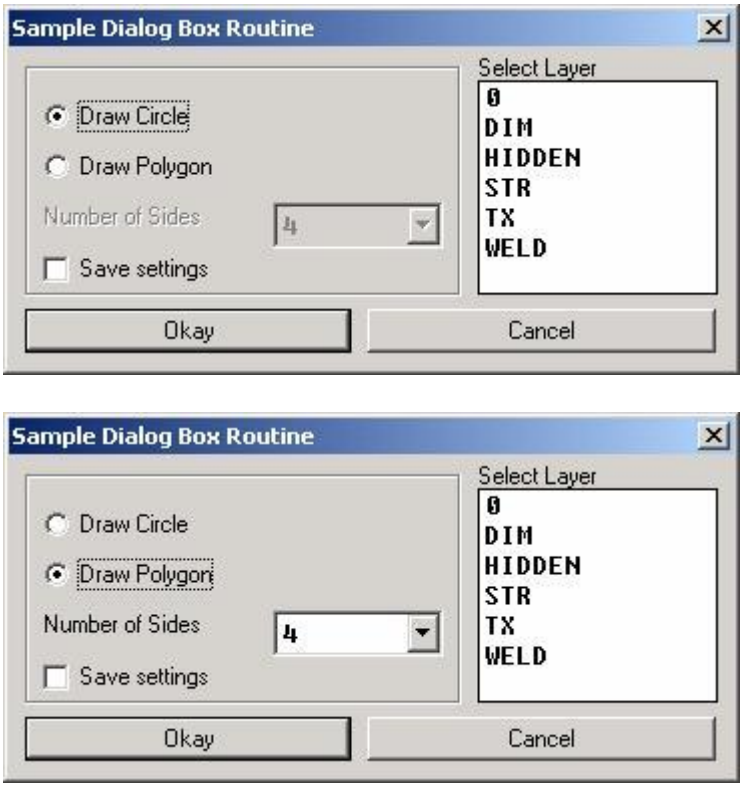
We have to create it and put it in our AutoLisp program. Like this: (Ta sẽ tạo nó và đặt nó vào chương trình Autolisp. Nó như sau:

```
(defun toggleRadio(a)

   ;if circle is selected
   (if(= a 1)
      (mode_tile "numsides" 1)   ;disable

      ;else
      (mode_tile "numsides" 0)   ;enable
    )
)
```

Since our default for the radio buttons is to draw a circle, the numSides popup_list should be disabled before the dialog box starts. So just before the action_tile statements we need to add this line: (Do mặc định cho các nút lựa chọn của chúng ta là vẽ vòng tròn, danh sách thả xuốNG "number of sides" sẽ biến mất trước khi hộp thoại bắt đầu. Vì thế, ngay trước các hàm thông báo phần tử hành động ta phải thêm các dòng sau:)

```
(mode_tile "numsides" 1)
```

Copy, Save, and Run. You should see this: (Copy, lưu lại và chạy thử. Bạn sẽ thấy như sau:)



The numSides popup_list is disabled when the circle is selected. Enabled when polygon is selected. Cool. (Danh sách thả xuống "Number of sides" không hoạt động khi việc vẽ vòng tròn được chọn và hoạt động khi vẽ đa giác được chọn. Tuyệt cú mèo.)

Here is the AutoLisp program after the lines above were added: (Đây là chương trình Autolisp sau khi đã thêm các dòng trên.)

```
(defun saveVars()
  (setq radios(get_tile "radios"))

  ;;;--- Get the number of sides selected from the list
  (setq sStr(get_tile "numsides"))
  (if(= sStr "")
    (setq numSides nil)
    (setq numSides(nth (atoi sStr) numSides))
  )

  ;;;--- See if the user wants to save the settings
  (setq saveSet(atoi(get_tile "saveset")))

  ;;;--- Get the selected item from the layer list
  (setq sStr(get_tile "layerlist"))

  ;;;--- If the index of the selected item is not "" then something was selected
  (if(/= sStr "")
    (progn
```

```
      ;;;--- Something is selected, so convert from string to integer
      (setq sIndex(atoi sStr))

      ;;;--- And get the selected item from the list
      (setq layerName(nth sIndex layerList))
    )

    ;;;--- Else, nothing is selected
    (progn

      ;;;--- Set the index number to -1
      (setq sIndex -1)

      ;;;--- And set the name of the selected item to nil
      (setq layerName nil)
    )
  )
)

(defun toggleRadio(a)
   ;if circle is selected
   (if(= a 1)
      (mode_tile "numsides" 1) ;disable
      ;else
      (mode_tile "numsides" 0) ;enable
    )
)

(defun C:SAMPLE7()

  ;;;--- Load the dcl file
  (setq dcl_id (load_dialog "SAMPLE7.dcl"))

  ;;;--- Load the dialog definition if it is not already loaded
  (if (not (new_dialog "SAMPLE7" dcl_id))
    (progn
      (alert "The SAMPLE7.DCL file could not be loaded!")
      (exit)
    )
  )

  (setq layerList(list "0" "DIM" "HIDDEN" "STR" "TX" "WELD"))
  (setq numSides(list "4" "6" "8" "12" "16"))

  ;;;--- Add the layer names to the dialog box
  (start_list "layerlist" 3)
  (mapcar 'add_list layerList)
  (end_list)

  ;;;--- Add the number of sides to the dialog box
  (start_list "numsides" 3)
  (mapcar 'add_list numSides)
  (end_list)

  (mode_tile "numsides" 1)

  ;;;--- If an action event occurs, do this function
  (action_tile "drawcir" "(toggleRadio 1)")
  (action_tile "drawpol" "(toggleRadio 2)")
  (action_tile "cancel" "(setq ddiag 1)(done_dialog)")
  (action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")
```

```
  ;;;--- Display the dialog box
  (start_dialog)

  ;;;--- Unload the dialog box
  (unload_dialog dcl_id)

  ;;;--- If the cancel button was pressed - display message
  (if (= ddiag 1)
    (princ "\n \n ...SAMPLE7 Cancelled. \n ")
  )

  ;;;--- If the "Okay" button was pressed
  (if (= ddiag 2)
    (princ "\n \n ...SAMPLE7 Complete!")
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

Now the only thing remaining is to do something when the user presses the okay button. We have all of the dialog box data stored in variable names.... (Bây giờ điều duy nhất còn lại là phải làm điều gì đó khi người sử dụng nhấn nút Okay. Ta có tất cả các dữ liệu của hộp thoại được lưu dưới các tên biến .)

Variable Name	DCL Control Item	Action Key	Type of Data Stored
radios	Radio_Column	"radios"	String [Action Key Name]
numSides	Popup_List	"numsides"	Integer [Number of sides]
saveSet	Toggle	"saveset"	Integer [0 or 1]
layerName	List	"layerlist"	String [Name of Layer]

So now all we have to do is write the AutoLisp code inside the "Okay button was pressed" IF statement. (Where the blue line is above.) (Ví thế bây giờ toàn bộ điều ta phải làm là viết mã Autolisp bên trong hàm thông báo If "Okay button was pressed". (Nơi có dòng chữ màu xanh ở trên))

Now we will replace the blue line above: `(princ "\n \n ...SAMPLE Complete!")` with new code (Bây giờ ta sẽ thay thế dòng chữ màu xanh (princ "\n \n …Sample7 Complete!") bằng các mã mới.)

(Cần chú ý bổ sung thông báo có nhiều hành động được thực hiện trong hàm if)

First, let's go ahead and change the layer.(Trước hết thay đổi các lớp ở phần đầu chương trình)

```
(setq oldLay(getvar "clayer"))
(setvar "clayer" layerName)
```

That was easy. Now, Let's draw the selected entity (Điều đó quá dễ, bây giờ ta vẽ đối tượng được chọn)

```
(if(= radios "drawcir")
```

```
  (progn
    (setq pt(getpoint "\n Center point: "))
    (command "circle" pt pause)
  )
  ;;;--- Else draw a polygon
  (progn
    (setq pt(getpoint "\n Center Point: "))
    (command "polygon" numSides pt "C" pause)
  )
)
```

Add the above to your program and save it. Test it out. Everything working okay? (Thêm tất cả các điều trên vào chương trình của chúng ta và lưu nó lại. Chạy thử. Mọi thứ làm việc tốt cả chứ?)

And finally all we have left is the code for the default settings. Since everything in the dialog box is a string, why don't we save all of the data as strings. Lucky for us, AutoDesk included 15 setvars for us to store data in. USERR1 thru USERR5 is used to store real numbers. USERI1 thru USERI5 is used to store integers. USERS1 thru USERS5 are used to store strings. We will use the system variables USERS1 thru USERS4 to save our data since we are saving the data as strings. All of these setvars are stored inside the drawing file. They will be there everytime you open your drawing. How convenient! So let's get to it.
Và cuối cùng ta còn phải làm là các mã cho việc đặt mặc định. Do mọi thứ trong hộp thoại đều là dạng chuỗi, vậy tại sao ta lại không lưu các dữ liệu dưới dạng chuỗi. Rất may mắn là Autodesk bao gồm 15 hàm đặt biến **setvars** để chúng ta có thể lưu các dữ liệu lại. Từ biến **USERR1** đến **USERR5** được sử dụng để lưu các số thực. Từ biến **USERI1** đến **USERI5** được sử dụng để lưu các số nguyên. Từ biến **USERS1** đến **USERS5** được dùng để lưu các chuỗi. Ta sẽ sử dụng các biến hệ thống **USERS1** đến **USERS4** để lưu dữ liệu của chúng ta do ta đang lưu dữ liệu dưới dạng chuỗi. Tất cả việc đặt biến **setvars** này được lưu bên trong tập tin bản vẽ. Chúng luôn ở đó mỗi khi bạn mở bản vẽ. Quá tiện lợi rồi! Vì thế ta sẽ lấy nó ra.

The first variable to store is RADIOS and it is already a string. So.... (Biến đầu tiên cần lưu là RADIOS và nó đã là một chuỗi . Cho nên)

```
(setvar "USERS1" radios)
```

The second variable to store is NUMSIDES and it is an integer representing the number of sides the polygon should have. I want to store the index of the selected item in the list not the actual number of sides. We saved the index as a variable named NUMSTR when we ran the saveVars routine. So... (Biến thứ hai phải lưu là NUMBERSIDES và nó là một số nguyên đại diện cho số cạnh của đa giác cần vẽ. Tôi muốn lưu số thứ tự của khoản mục được chọn trong danh sách đó chứ không phải là số các cạnh thật sự của đa giác. Ta sẽ lưu số thứ tự đó như một biến mang tên NUMSTR khi chúng ta chạy vòng lưu biến saveVars. Vì thế)

```
(setvar "USERS2" numStr)
```

The third variable to save is SAVESET. It is stored as an integer. We will have to convert it to a string to save it.(Biến thư ba phải lưu là SAVESET. Nó được lưu như một số nguyên. Ta sẽ đổi nó thành một chuỗi để lưu nó lại.)

```
(setvar "USERS3" (itoa saveSet))
```

The fourth and final variable to save is LAYERNAME. It is stored as a string. I want to save the index of the selected item. We saved that earlier with the saveVars routine in a variable named sSTR. (Thứ tư và cũng là biến cuối cùng phải lưu là LAYERNAME. Nó được lưu ở dạng chuỗi. Tôi muốn lưu số thứ tự của

khoản mục được chọn. Ta đã lưu điều này từ trước nhờ vòng lưu biến SaveVars khi đặt tên biến sSTR.)

```
(setvar "USERS4" sSTR)
```

The last thing we have to do is check for the default settings and load them if they exist. (Điều cuối cùng ta phải làm là kiểm tra các giá trị mặc định và tải nó nếu như nó tồn tại.)

```
(defun saveVars()

  (setq radios(get_tile "radios"))

  ;;;--- Get the number of sides selected from the list
  (setq numStr(get_tile "numsides"))
  (if(= numStr "")
    (setq numSides nil)
    (setq numSides(nth (atoi numStr) numSides))
  )

  ;;;--- See if the user wants to save the settings
  (setq saveSet(atoi(get_tile "saveset")))

  ;;;--- Get the selected item from the layer list
  (setq sStr(get_tile "layerlist"))

  ;;;--- If the index of the selected item is not "" then something was selected
  (if(/= sStr "")
    (progn

      ;;;--- Something is selected, so convert from string to integer
      (setq sIndex(atoi sStr))

      ;;;--- And get the selected item from the list
      (setq layerName(nth sIndex layerList))
    )

    ;;;--- Else, nothing is selected
    (progn

      ;;;--- Set the index number to -1
      (setq sIndex -1)

      ;;;--- And set the name of the selected item to nil
      (setq layerName nil)
    )
  )
)

(defun toggleRadio(a)
   ;if circle is selected
   (if(= a 1)
      (mode_tile "numsides" 1) ;disable
      ;else
      (mode_tile "numsides" 0) ;enable
    )
)

(defun C:SAMPLE7()

  ;;;--- Load the dcl file
```

```
(setq dcl_id (load_dialog "SAMPLE7.dcl"))

;;;--- Load the dialog definition if it is not already loaded
(if (not (new_dialog "SAMPLE7" dcl_id))
  (progn
    (alert "The SAMPLE7.DCL file could not be loaded!")
    (exit)
  )
)

;;; Build the list for the dialog box
(setq layerList(list "0" "DIM" "HIDDEN" "STR" "TX" "WELD"))
(setq numSides(list "4" "6" "8" "12" "16"))

;;;--- Add the layer names to the dialog box
(start_list "layerlist" 3)
(mapcar 'add_list layerList)
(end_list)

;;;--- Add the number of sides to the dialog box
(start_list "numsides" 3)
(mapcar 'add_list numSides)
(end_list)


;;;--- Add the code here to check for defaults
(if(/= (getvar "USERS1") "")
  (progn
    (setq radios (getvar "users1"))
    (setq numStr (getvar "users2"))
    (setq saveSet(getvar "users3"))
    (setq layerIndex(getvar "users4"))
    (set_tile "radios" radios)
    (set_tile "numsides" numStr)
    (set_tile "saveset" saveSet)
    (set_tile "layerlist" layerIndex)
  )
)

;;;--- Only disable the numSides popup_list if a circle is being drawn
(if(= radios "drawcir")
  (mode_tile "numsides" 1)
)

;;;--- If an action event occurs, do this function
(action_tile "drawcir" "(toggleRadio 1)")
(action_tile "drawpol" "(toggleRadio 2)")
(action_tile "cancel" "(setq ddiag 1)(done_dialog)")
(action_tile "accept" "(setq ddiag 2)(saveVars)(done_dialog)")

;;;--- Display the dialog box
(start_dialog)

;;;--- Unload the dialog box
(unload_dialog dcl_id)

;;;--- If the cancel button was pressed - display message
(if (= ddiag 1)
  (princ "\n \n ...SAMPLE7 Cancelled. \n ")
)
```

```
  ;;;--- If the "Okay" button was pressed
  (if (= ddiag 2)
    (progn

      ;;;--- Save the old layer and reset to new layer
      (setq oldLay(getvar "clayer"))
      (setvar "clayer" layerName)

      ;;;--- See what needs to be drawn
      (if(= radios "drawcir")
        (progn
          (setq pt(getpoint "\n Center point: "))
          (command "circle" pt pause)
        )

        ;;;--- Else draw a polygon
        (progn
          (setq pt(getpoint "\n Center Point: "))
          (command "polygon" numSides pt "C" pause)
        )
      )

      ;;;--- See if we need to save the settings
      (if(= saveSet 1)
        (progn

          ;;;--- Add code here to save the settings as defaults
          (setvar "USERS1" radios)
          (setvar "USERS2" numStr)
          (setvar "USERS3" (itoa saveSet))
          (setvar "USERS4" sSTR)

        )
      )
    )
  )

  ;;;--- Suppress the last echo for a clean exit
  (princ)

)
```

When you get your program tested and everything is working, move the blue line above, [`(defun C:SAMPLE7()`] all the way to the top of the file. This will make all of your variables local and will reset them all to nil when the program ends.

That's it. We're done.

Back

Programs

ADDLEN - Total the lengths of selected arcs, circles, ellipses, lines, lwpolylines, polylines, and/or splines.

AlgaCAD - Write AutoLisp programs to fill in standard drawings.

ALIAS - Shortcuts / Single Letter Commands on the fly.

ATTINC - Increment values of attributes.

ATTMAP - Map Attributes from drawing to drawing.

Autolisp Games - TicTacToe, HangMan, Mines, BattleShip, and Mummy.

Axcel - VB Application for Excel.

AxcelPts - VB Application for Excel.

BatchLisp - Run a selected autolisp program on multiple drawings.

BLKOUT - Extract nested blocks into individual files.

BLKTREE - Display blocks and nested blocks in a tree format.

BLOCKS - Count blocks filtered by name, attribute tag, or attribute value.

CAD2FILE - Get entity data from AutoCAD to a file or into Excel.

CASE Change text to upper case, lower case, or upper case the first letter of each word.

CHAIN Draw a chain like drawing a line. From point to point to point.

CLAY Create layer(s) from a master list.

CNTBLK Count blocks (Attribute Filter Also).

CSVTable Draw a table in autocad based on data from a selected CSV file.

CYL - Create hollow 3D circular, rectangular, or polygonal shapes.

Free AutoCAD Tools

GA Get the area and perimeter by selecting the interior of any area.

GETCELLS Function to get the values of a cell or cells from Excel into a list..

IMPORT XYZ Import coordinates from practically any type of file. Draw nodes, draw circles, draw lines, insert blocks, insert blocks and update attributes with the coordinates

ISODIM Create non-associative isometric dimensions.

LIBRARY Create your own slide library automatically.

LoadLSP AutoLisp loader. View all of your autolisp programs along with a description.

Match - Draw a matchline.

Num.lsp - Program to create charts automatically.

OverLapF - Remove Overlapping Lines.

REMF Remove Multiple Layer Filters.

RepAtt Replace the value of all matching attributes. Includes a value filter.

Rolling_ball Thanks to Guenther Bittner, the program is complete. Find the mid-boundary between two polylines. Verion 4+ now available.

SafeX - Explode a block containing attributes and replace the attributes with a text entity containing the original attribute value.

SBox Create a Shadow Box

SelectPlotter - Select a plotter/printer from a list box.

Simplex - Hatchable Balloon Font.

Words - Count words inside an autolisp drawing using filets. **Updated!**

XL Get data from Excel into AutoCAD using Visual Lisp. **Version 3.1**

XMINSERT Explode a MINSERT object.

Cogenerator

Make_LSP - AutoLisp Cogenerator. (AutoLisp program to write AutoLisp programs.)

Make_LSP Help and Tutorial

(Some of these programs were created with the Make_LSP program.)

AngX - Rotates entities by picking a line with the correct angle.

Au - Prefixes selected text with "%%u " and suffixes with " ".

BrkLine - Draws a break line.

Case - Reverses the case of text. (Upper to Lower and Lower to Upper.)

ChTxHt - Changes the height of selected text.

ChTxLy - Changes the Layer of selected text.

ChTxRt - Changes the Rotation of selected text.

ChTxSt - Changes the Style of selected text.

Cloud - Creates a revision cloud.

CPYXREF - Copies entities from an XREF or Block.

CrText - Circular Text

DrLim - Draw a line representing the limits of the drawing. (LimMax to LimMin)

EBT - Erase Blank Text. Removes all text with a value of " " or ""

ErasNode - Erase all nodes in the drawing.

FL - Select a line and this program will put the feet(decimal) and the angle(deg min sec) centered on top of the line.

FZ and UFZ FZ - FreeZes all layers including the current layer. UFZ - UnFreeZes all layers.

Ma - Match text by selecting the items.

MixTxt - Rotates words through a text entity. Why? I don't know.

NodeSert - Insert blocks on every node in a drawing.

NumberX - This programs makes numbers out of vertical text starting with number 1 being the highest vertical text location and working downward.

PipeLay - This program creates a layout drawing of a pipe to pipe connection to help the shop fabricate this unusual connection. The shop wraps the paper around the pipe and marks the cut lines. Click the link to find out more.

PreTx - Prefix all text with a string.

RRR - Text search and replace routine.

Sag - Draws the sag for conductor lines in 3D.

Simplex - Draws "ballooned" simplex characters with polylines.

Slot - Draws a solid slotted hole.

SpcText - Puts spaces between each character in a text string.

srtAN - Function to sort alpha-numerically. See inside file for instructions.

StText STX - Stretches a text entity. USTX - Un-stretches a text entity.

SwapEnt - Swaps two entities.

SwapTxt - Quickly swaps two text entities.

TextIn - Prints a text file in AutoCAD

TextOut - Prints AutoCAD text to a file.

TotalTx - Totals numeric text entities.
