AFRALISP

The AutoLisp Tutorials

DCL-Dialog Control Language

Written and Compiled by Kenny Ramage
afralisp@mweb.com.na
http://www.afralisp.com

afralisp@mweb.com.na
http://www.afralisp.com

Contents

Page Number	Chapter
Page 4	Getting Started
Page 13	Dialog Box Layout
Page 30	Dialog Boxes Step by Step
Page 55	Dialog Boxes in Action
Page 62	Nesting and Hiding Dialogs.
Page 74	Hiding Dialogs Revisited.
Page 82	AutoLisp Message Box.
Page 93	AutoLisp Input Box.
Page 95	Referencing DCL Files.
Page 103	Functional Synopsis of DCL Files.
Page 196	DCL Attributes.
Page 255	Dialog Box Default Data.
Page 266	Attributes and Dialog Boxes
Page 274	DCL without the DCL File
Page 283	DCL Model.
Page 317	Acknowledgements and Links

Getting Started With DCL

Dialog Control Language, or DCL, always seems to frighten off a lot of Lispers. I admit, it did me too until I was forced into a situation were I had to learn it and quickly. (make it work, or you're out boy!)
Well, I would hate for any of you to be in the same situation, so for the next few issues, I'll be taking you step by step, hand in hand, through the minefield of the DCL language. I will share your pain and misery, and will wipe away your tears, I will.....
("Hey Kenny, get on with it!")
Oops, sorry!
Anyway, where was I? Oh yeah, I will suffer.....(thump!!)
As I was saying before I got this black eye, there are a few terms that you need to familiarise yourself with before we get stuck into some coding.
Firstly, the dialog box itself is known as a "dialog definition".
Secondly, each "control" on the dialog is known as a "tile definition".
Thirdly, each "property" of a "tile" is known as a dialog "attribute".
And fourthly, each "method" of a "tile" is known as an "action expression".
Why? Who knows? Who cares? All I know is that it will help you immensely in understanding the AutoCAD DCL reference book if you have a basic knowledge of these terms.
Right, enough waffle, I'm bored and my eye's sore. Let's have a look at some DCL coding and design ourselves a simple dialog box.

Copy and paste this into Notepad and save it as "TEST_DCL1.DCL".
Oh, before I forget, please ensure that you save this file, and it's namesake AutoLisp file, into a directory that is within your AutoCAD search path.

```
//DCL CODING STARTS HERE
test_dcl1

: dialog

{
label = "Test Dialog No 1";

        : text
        {
        label = "This is a Test Message";
        alignment = centered;
        }

        : button
        {
        key = "accept";
        label = "Close";
        is_default = true;
        fixed_width = true;
        alignment = centered;
        }

}
//DCL CODING ENDS HERE
```

We'll have a closer look at what this all means a bit later. First, let's load some AutoLisp coding and try out our new dialog box.
Copy and paste this into Notepad and save it as "TEST_DCL.LSP".

```
;AUTOLISP CODING STARTS HERE
(prompt "\nType TEST_DCL1 to run.....")

(defun C:TEST_DCL1 ()

(setq dcl_id (load_dialog "test_dcl1.dcl"))

     (if (not (new_dialog "test_dcl1" dcl_id))
         (exit )
     );if

(action_tile "accept"
    "(done_dialog)"
);action_tile

(start_dialog)
(unload_dialog dcl_id)

(princ)

);defun
(princ)
;AUTOLISP CODING ENDS HERE
```

Now load and run your program.

A very simple dialog box containing a message should appear on your screen. It did? Good, well done!! (thunderous applause from the peanut gallery.)

Right, let's dissect the DCL coding.
(theme music from an old Dracula movie starts in the background.)

//DCL CODING STARTS HERE
Anything starting with // in DCL is regarded as a comment.

test_dcl1
The name of the dialog.

: dialog

The start of the dialog definition

{
The opening bracket for the dialog definition

label = "Test Dialog No 1";
The Label of the dialog definition.
This is what appears in the title bar of the dialog

: text
The start of a text tile definition

{
The opening bracket for the text tile definition

label = "This is a Test Message";
The label attribute of the text tile

alignment = centered;
The alignment attribute of the text tile

}
The closing bracket of the text tile

: button
The start of a button tile definition

{
The opening bracket for the button tile definition

key = "accept";
The key, or name of the button tile.
You will use this name to reference this button in your AutoLisp coding

label = "Close";
The label attribute. What appears on it.

is_default = true;
The default attribute. If this is true, this button will automatically be selected if the "Enter" button is pressed

fixed_width = true;
Forces the button to be just large enough for the label attribute

alignment = centered;
The alignment attribute

}
The closing bracket for the button tile

}
The closing bracket for the dialog definition

OK, that was easy hey? By the way, did you notice that each of the attribute lines finished with a semicolon? (;)
Another important thing that you must remember when dealing with attributes is that their values are case sensitive. (e.g.. "True" does not equal "true".)

Now let's have a wee look at the AutoLisp coding that puts this whole thing together. Again, we'll take it line by line :

(prompt "\nType TEST_DCL1 to run.....")
Inform the user how to start the program.
Just good manners.

(defun C:TEST_DCL1 (/ dcl_id)
Define the function and declare variables

(setq dcl_id (load_dialog "test_dcl1.dcl"))
Load the dialog file and set a reference to it.

(if (not (new_dialog "test_dcl1" dcl_id))
Load the dialog definition and check for it's existence.
Remember, a dialogue file can hold various dialogue definitions

(exit)
Exit the program if the dialog definition is not found.

);if
End if

(action_tile "accept"
If the user selects the tile who's name is "accept", then do the following :

"(done_dialog)"
Close the dialog

);action_tile
End of action_tile

(start_dialog)
Start the dialog

(unload_dialog dcl_id)
Unload the dialog from memory

(princ)
finish clean

);defun
End of function

(princ)
Load clean

Many people get confused in regards to the order of AutoLisp statements when dealing with DCL files. I don't blame 'em really. I mean look at the coding above!

- First we load the dialog file.
- Then we load the dialog definition.
- Then we run an action_tile statement.
- Then we start the dialog.
- And right after that, we unload the dialog.

Where's the logic in that?

Haha. But did you notice that the action_tile statement was quoted?
e.g.. *"(done_dialog)".*

In effect, what we are telling the tile is this:

"Remember this string, then pass it back to me when the user activates you."

"So, in other words, coding is pre-stored within a particular tile. When I select that tile the coding runs?"

Yep, that's it. But remember, no values will be returned until *(done_dialog)* is called, which is also quoted.

So, the sequence is like this :

- First we load the dialog file.
- Then we load the dialog definition.
- Then we run AND REMEMBER all the action_tile statements.
- Then we start the dialog.
- The dialog is displayed.
- Any action_tile statement is processed IF the relevant tile is selected.

When a tile that contains the *(done_dialog)* function is selected, we unload the dialog and return all tile values.

And that's it. Easy hey?
Next, we'll have a look at some predefined tiles, how to enter and retrieve values from a dialog, and how to validate these values.

OK, tea break over, back to work!!

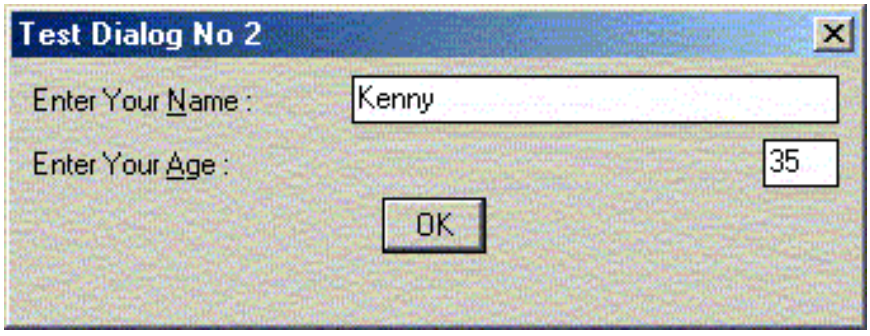
Copy and paste this into Notepad and save it as "TEST_DCL2.DCL".

```
//DCL CODING STARTS HERE
test_dcl2

: dialog

{

label = "Test Dialog No 2";

        : edit_box
        {
        label = "Enter Your Name :";
        mnemonic = "N";
        key = "name";
        alignment = centered;
        edit_limit = 30;
        edit_width = 30;
        }

        : edit_box
        {
        label = "Enter Your Age :";
        mnemonic = "A";
        key = "age";
        alignment = centered;
        edit_limit = 3;
        edit_width = 3;
        value = "";
        }

        : button
        {
        key = "accept";
        label = "OK";
        is_default = true;
        fixed_width = true;
        alignment = centered;
        }

        : errtile
        {
        width = 34;
        }
}
//DCL CODING ENDS HERE
```

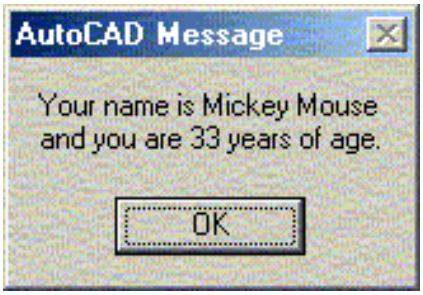
And now the AutoLisp coding. Copy and paste this and save it as "TEST_DCL2.LSP" and then load and run it :

```
;AUTOLISP CODING STARTS HERE
(prompt "\nType TEST_DCL2 to run.....")

(defun C:TEST_DCL2 ( / dcl_id)

(setq dcl_id (load_dialog "test_dcl2.dcl"))
     (if (not (new_dialog "test_dcl2" dcl_id))
         (exit )
     );if

(set_tile "name" "Enter Name Here")
(mode_tile "name" 2)
(action_tile "name" "(setq name $value)")
(action_tile "age" "(setq age $value)")
(action_tile "accept" "(val1)")

(start_dialog)
(unload_dialog dcl_id)

(alert (strcat "Your name is " name
               "\nand you are " age " years of age."))

(princ)

);defun

-----------------------

(defun val1 ()

(if (= (get_tile "name") "Enter Name Here")
        (progn
           (set_tile "error" "You must enter a name!")
           (mode_tile "name" 2)
        );progn
        (val2)
);if

);defun

-------------------

(defun val2 ()

(if (< (atoi (get_tile "age")) 1)
        (progn
```

```
          (set_tile "error" "Invalid Age - Please Try Again!!")
          (mode_tile "age" 2)
        );progn
        (done_dialog)
);if

);defun

(princ)
;AUTOLISP CODING ENDS HERE
```

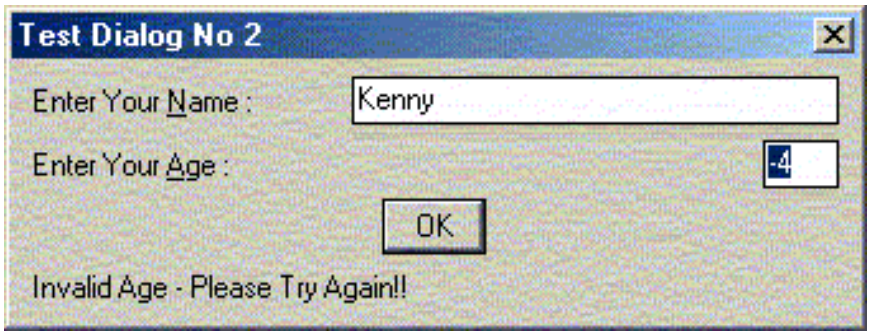


The dialog that is displayed contains two edit boxes. One to enter your name into, and one your age. After entering the relevant information select the "OK" button. An alert box will display showing your details.



Now run the program again, but do not enter your name. An error message will display informing you that you must enter your name.

Fill in your name but leave your age blank. Press enter again. Another error message will inform you that you have entered an invalid age.



Try it with a zero or a negative number. The same error message will display until you enter a number equal or greater than 1. Both of these examples are know as validation.

Let's have a look at the DCL coding.

:edit_box - This is a predefined tile that allows the user to enter or edit a single line of text.
There are a few new attributes defined in this tile :

mnemonic = "A"; **- This defines the mnemonic character for the tile.**
edit_limit = 3; **- This limits the size of the edit box to 3 characters.**
edit_width = 3; **- This limits the user to typing a maximum of 3 characters.**
value = ""; **- This sets the intial value of the edit box, in this case nothing.**

Now let's have a look at the AutoLisp coding. First the "action expressions".

(set_tile "name" "Enter Name Here") **- This sets the runtime (initial) value of a tile who's "key" attribute is "name".**

(mode_tile "name" 2) **- This sets the mode of the tile who's "key" attribute is "name". In this example, we have used a mode of "2" that allows for overwriting of what is already in the edit box.**

(action_tile "name" "(setq name $value)") **- Associates the specied tile with the action expression or call back function. In this case we are saying "If the tile who's "key" attribute is "name" is selected, store the value of this tile in the variable name."**

(get_tile "name") **- Retrieve the value of the tile who's "key" attribute is "name".**

Did you notice that we wrote :
(action_tile "accept" "(val1)")
instead of :
(action_tile "accept" "(done_dialog)")

Instead of closing the dialog when OK is selected, we initiate a subroutine that checks (validates) that the name information is exceptable. If it is not, the error tile is displayed.
(set_tile "error" "You must enter a name!")

Once the name has been validated, this routine then initiates a second subroutine that validates the age value.
If this not correct, the error tile is again displayed, this time with a different message.
(set_tile "error" "Invalid Age - Please Try Again!!")
If everything is correct, *(done_dialog)* is called, all values are returned and the alert box is displayed with the relevant information.

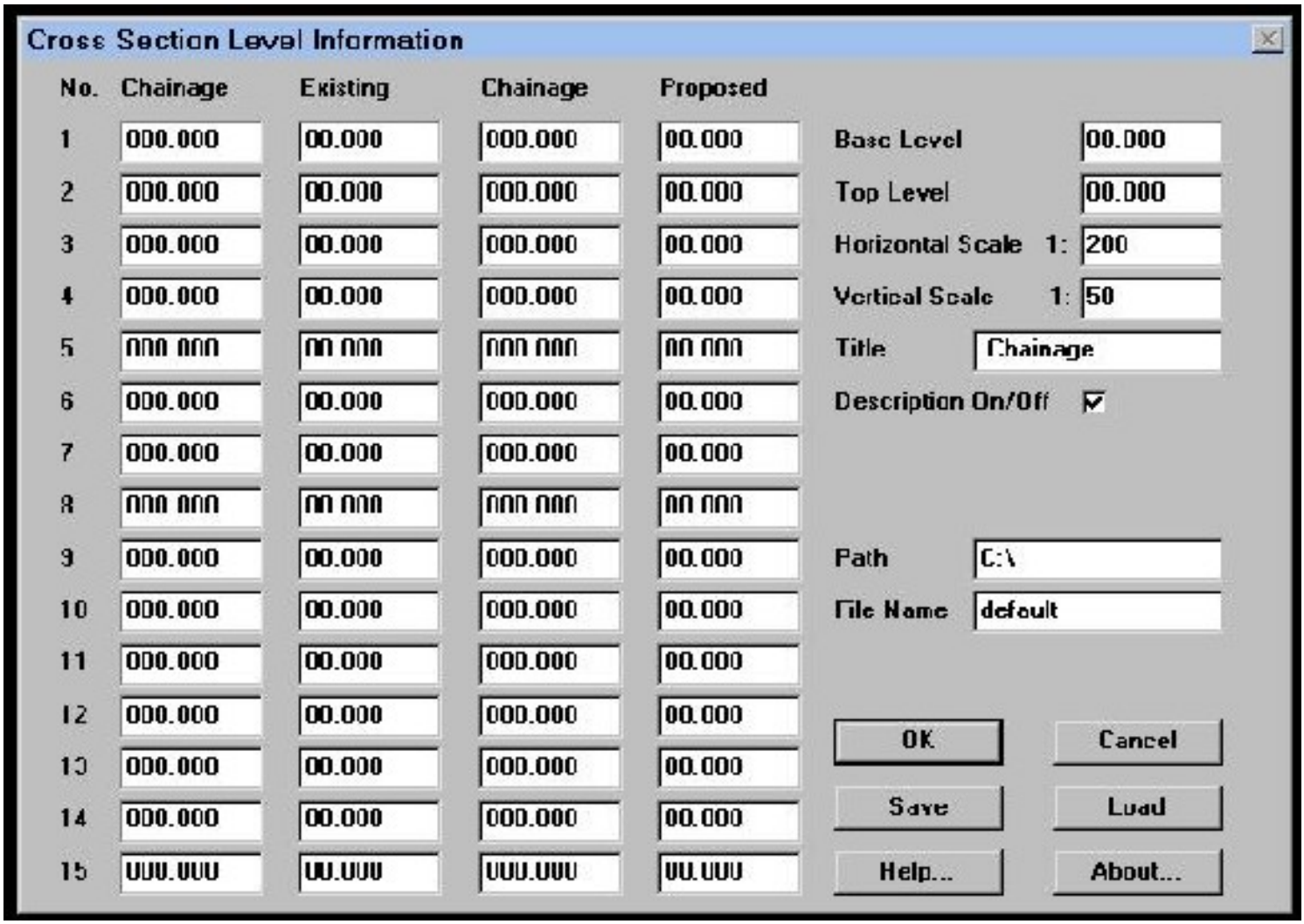
Dialog Box Layout

Laying out Dialog Boxes in DCL can be a bit tricky and entails quite a lot of "trial and error". But, if you follow a few simple guidelines, a complicated dialog box can be simplified quite considerably. To view the dialogs throughout this tutorial, we will be making use of the Visual Lisp Editor and the "Preview DCL in Editor" function. If you are unfamiliar with the usage of this very handy function, you'll find step by step instructions here.

If you would rather call the dialog boxes from AutoCAD, I have included AutoLisp coding that will fulfill this purpose in the download file that you can find at the end of this tutorial.

As well, explanations for all the tile attributes are covered in this section - "DCL Tile Attributes". I suggest you refer to this whilst going through this tutorial.

OK, let's get started. Consider this :



Rather a complicated dialog hey?

Laying out a dialog is similar in way to writing a program. You don't start off by writing the whole program in one go. You write the program in sections, testing each part as you go along. Only when each part has been thoroughly tested, do you start to put the whole program together.

We are going to do exactly the same, splitting our dialog into 3 separate DCL files before merging them all together into one.

Let's have a look at the button section first :

We'll start off by creating a dialog with just two buttons, namely the "OK" and "Cancel" buttons. Open a new file in the Visual Lisp editor and Copy and paste this.

```
afra : dialog {
          label = "A" ;

         : button {
          label = "OK";
          key = "accept";
          mnemonic = "O";
              alignment = centered;
              width = 12;
              is_default = true;
        }

        : button {
              label = "Cancel";
         key = "cancel";
         mnemonic = "C";
              alignment = centered;
              width = 12;
        }

        }
```

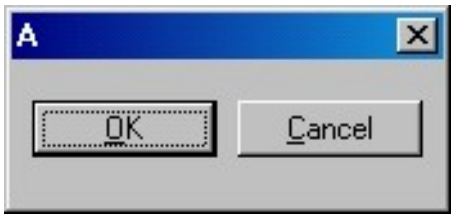
Save this as "Afra.dcl".

Using the "Preview DCL in Editor" function, your dialog should look like this :

Did you notice something? The button layout "defaulted" to a column layout. But we need these two buttons to be in a row! Ok, lets do that :

```
afra : dialog {
        label = "A" ;

        : row {

        : button {
        label = "OK";
        key = "accept";
        mnemonic = "O";
        alignment = centered;
        width = 12;
        is_default = true;
        }

        : button {
        label = "Cancel";
        key = "cancel";
        mnemonic = "C";
        alignment = centered;
        width = 12;
        }

        }

}
```

Your dialog should now look like this :



A word of advice. Do not rely on your tiles to default to columns. Rather explicitly define a column within your DCL Coding. If not? Well, confusion will reign.....

Now we'll add the other four buttons to create our button cluster :

```
afra : dialog {
        label = "A" ;

        : column {

        : row {

        : button {
        label = "OK";
        key = "accept";
```

```
mnemonic = "O";
alignment = centered;
width = 12;
is_default = true;
}

: button {
label = "Cancel";
key = "cancel";
mnemonic = "C";
alignment = centered;
width = 12;
}

}

: row {

: button {
label = "Save";
key = "save";
mnemonic = "S";
alignment = centered;
width = 12;
}

: button {
label = "Load";
key = "load";
mnemonic = "L";
alignment = centered;
width = 12;
}

}

: row {

: button {
label = "Help...";
key = "help";
mnemonic = "H";
alignment = centered;
width = 12;
}

: button {
label = "About...";
key = "About";
mnemonic = "H";
```

```
        alignment = centered;
        width = 12;
        }

        }

        }

}
```

And, our dialog should now look like this :

Did you notice how only one of the button tiles has the "is_default" attribute set to "true"? Good, you did! And why? Think about it!!!

For the sake of readability, and to save some space, you could also write the DCL coding like this :

```
afra : dialog {
        label = "A" ;

        : column {

        : row {

        : button {label = "OK"; key = "accept"; mnemonic = "O";
                  alignment = centered; width = 12;
                  is_default = true;}
        : button {label = "Cancel"; key = "cancel";mnemonic = "C";
                  alignment = centered; width = 12;}

        }

        : row {

        : button {label = "Save"; key = "save"; mnemonic = "S";
                  alignment = centered; width = 12;}
        : button {label = "Load";key = "load"; mnemonic = "L";
                  alignment = centered; width = 12;}

        }

        : row {
```

```
        : button {label = "Help..."; key = "help"; mnemonic = "H";
                          alignment = centered; width = 12;}
        : button {label = "About..."; key = "About"; mnemonic = "H";
                          alignment = centered; width = 12;}

        }

        }

}
```

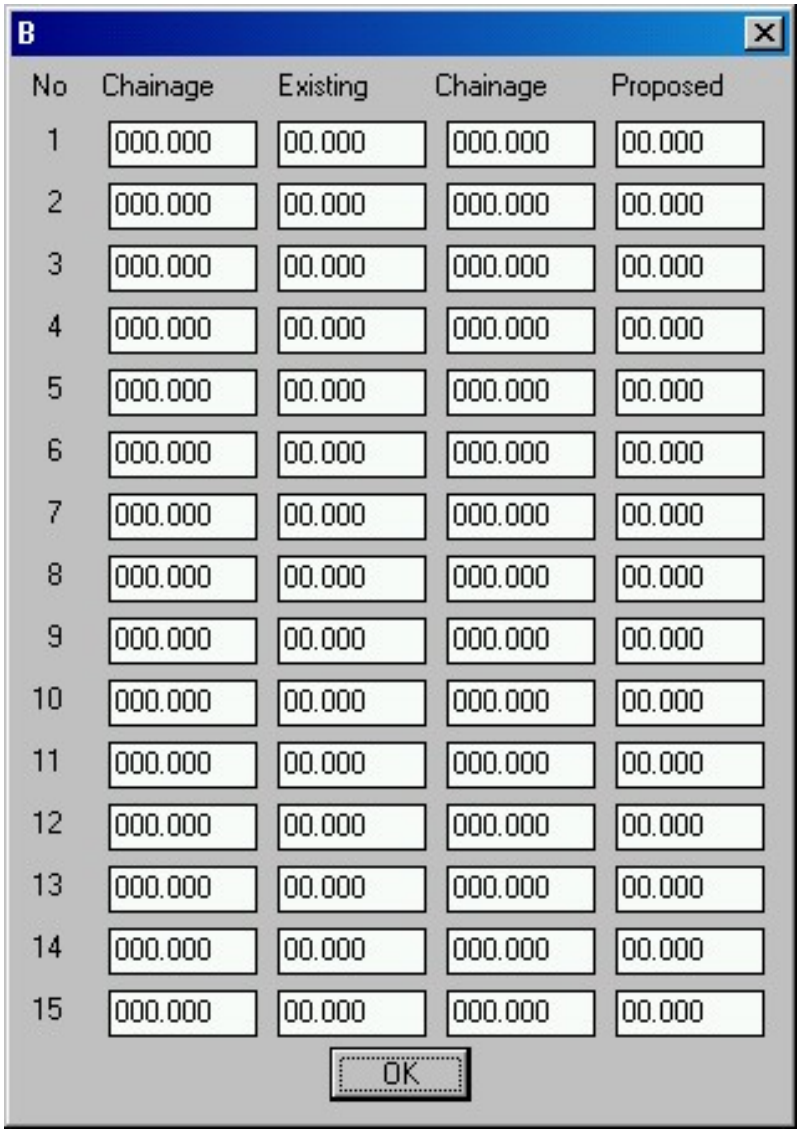
Right, now sit up straight and pay attention.
Open a new file and copy and paste this into it :

```
afra1 : dialog {
      label = "B" ;

      : column {

      : row {

            : text {label = "No";}
            : text {label = "Chainage";}
            : text {label = "Existing";}
            : text {label = "Chainage";}
            : text {label = "Proposed";}

      }

      : row {
            : text {label = "  1";}
            : edit_box {key = eb1; width = 7; value = "000.000";}
            : edit_box {key = eb1a; width = 7; value = "00.000";}
            : edit_box {key = eb1b; width = 7; value = "000.000";}
            : edit_box {key = eb1c; width = 7; value = "00.000";}
      }
      : row {
            : text {label = "  2";}
            : edit_box {key = eb2; width = 7; value = "000.000";}
            : edit_box {key = eb2a; width = 7; value = "00.000";}
            : edit_box {key = eb2b; width = 7; value = "000.000";}
            : edit_box {key = eb2c; width = 7; value = "00.000";}
      }
      : row {
            : text {label = "  3";}
            : edit_box {key = eb3; width = 7; value = "000.000";}
            : edit_box {key = eb3a; width = 7; value = "00.000";}
            : edit_box {key = eb3b; width = 7; value = "000.000";}
            : edit_box {key = eb3c; width = 7; value = "00.000";}
      }
      : row {
            : text {label = "  4";}
            : edit_box {key = eb4; width = 7; value = "000.000";}
            : edit_box {key = eb4a; width = 7; value = "00.000";}
            : edit_box {key = eb4b; width = 7; value = "000.000";}
            : edit_box {key = eb4c; width = 7; value = "00.000";}
      }
      : row {
            : text {label = "  5";}
            : edit_box {key = eb5; width = 7; value = "000.000";}
            : edit_box {key = eb5a; width = 7; value = "00.000";}
            : edit_box {key = eb5b; width = 7; value = "000.000";}
            : edit_box {key = eb5c; width = 7; value = "00.000";}
      }
      : row {
            : text {label = "  6";}
            : edit_box {key = eb6; width = 7; value = "000.000";}
```

```
        : edit_box {key = eb6a; width = 7; value = "00.000";}
        : edit_box {key = eb6b; width = 7; value = "000.000";}
        : edit_box {key = eb6c; width = 7; value = "00.000";}
}
: row {
        : text {label = "  7";}
        : edit_box {key = eb7; width = 7; value = "000.000";}
        : edit_box {key = eb7a; width = 7; value = "00.000";}
        : edit_box {key = eb7b; width = 7; value = "000.000";}
        : edit_box {key = eb7c; width = 7; value = "00.000";}
}
: row {
        : text {label = "  8";}
        : edit_box {key = eb8; width = 7; value = "000.000";}
        : edit_box {key = eb8a; width = 7; value = "00.000";}
        : edit_box {key = eb8b; width = 7; value = "000.000";}
        : edit_box {key = eb8c; width = 7; value = "00.000";}
}
: row {
        : text {label = "  9";}
        : edit_box {key = eb9; width = 7; value = "000.000";}
        : edit_box {key = eb9a; width = 7; value = "00.000";}
        : edit_box {key = eb9b; width = 7; value = "000.000";}
        : edit_box {key = eb9c; width = 7; value = "00.000";}
}
: row {
        : text {label = "10";}
        : edit_box {key = eb10; width = 7; value = "000.000";}
        : edit_box {key = eb10a; width = 7; value = "00.000";}
        : edit_box {key = eb10b; width = 7; value = "000.000";}
        : edit_box {key = eb10c; width = 7; value = "00.000";}
}
: row {
: text {label = "11";}
        : edit_box {key = eb11; width = 7; value = "000.000";}
        : edit_box {key = eb11a; width = 7; value = "00.000";}
        : edit_box {key = eb11b; width = 7; value = "000.000";}
        : edit_box {key = eb11c; width = 7; value = "00.000";}
}
: row {
        : text {label = "12";}
        : edit_box {key = eb12; width = 7; value = "000.000";}
        : edit_box {key = eb12a; width = 7; value = "00.000";}
        : edit_box {key = eb12b; width = 7; value = "000.000";}
        : edit_box {key = eb12c; width = 7; value = "00.000";}
}
: row {
        : text {label = "13";}
        : edit_box {key = eb13; width = 7; value = "000.000";}
        : edit_box {key = eb13a; width = 7; value = "00.000";}
        : edit_box {key = eb13b; width = 7; value = "000.000";}
        : edit_box {key = eb13c; width = 7; value = "00.000";}
}
: row {
        : text {label = "14";}
        : edit_box {key = eb14; width = 7; value = "000.000";}
        : edit_box {key = eb14a; width = 7; value = "00.000";}
        : edit_box {key = eb14b; width = 7; value = "000.000";}
```

```
        : edit_box {key = eb14c; width = 7; value = "00.000";}
    }
    : row {
        : text {label = "15";}
        : edit_box {key = eb15; width = 7; value = "000.000";}
        : edit_box {key = eb15a; width = 7; value = "00.000";}
        : edit_box {key = eb15b; width = 7; value = "000.000";}
        : edit_box {key = eb15c; width = 7; value = "00.000";}
    }

    }

    ok_only;

}
```

Save this as "afra1.dcl". Now view the dialog :

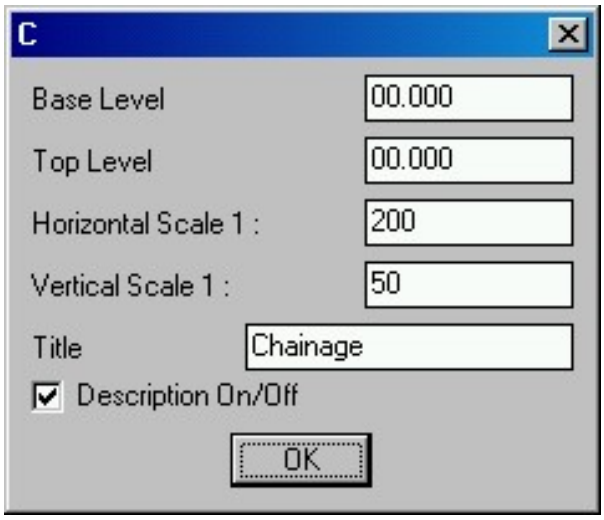


Did you notice how we where forced to include an "OK" button within the dialog?
Don't worry though, we'll remove that later.

Ok, on to the next section. Open a new file and add this coding :

```
        afra2 : dialog {
    label = "C" ;

    : column {

        : edit_box {key = "eb16"; label = "Base Level";
                edit_width = 12;value = "00.000";}
        : edit_box {key = "eb17"; label = "Top Level";
                edit_width = 12; value = "00.000";}
        : edit_box {key = "eb18"; label = "Horizontal Scale 1 :";
                edit_width = 12; value = "200";}
        : edit_box {key = "eb19"; label = "Vertical Scale 1 :";
                edit_width = 12; value = "50";}
        : edit_box {key = "eb20"; label = "Title";
                edit_width = 20;value = "Chainage";}
        : toggle {key = "tg1"; value = 1;
                label = "Description On/Off";}
    }

    ok_only;
}
```

Save this as "afra2.dcl" and then display it. It will look like this :

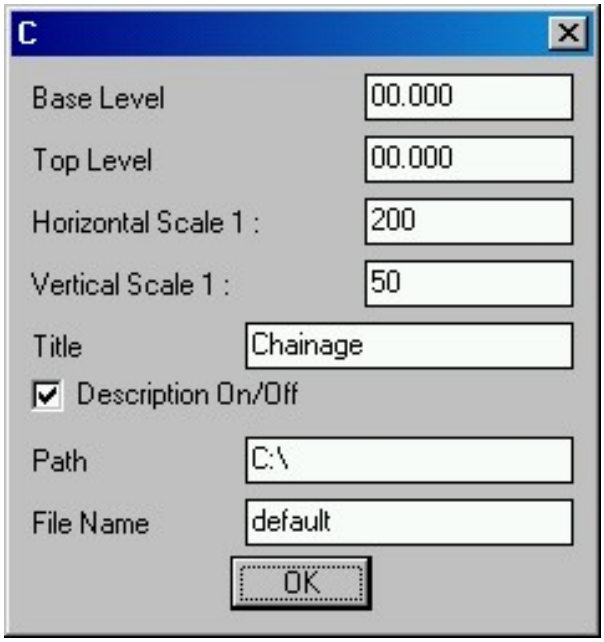


Now let's add the the other two text boxes :

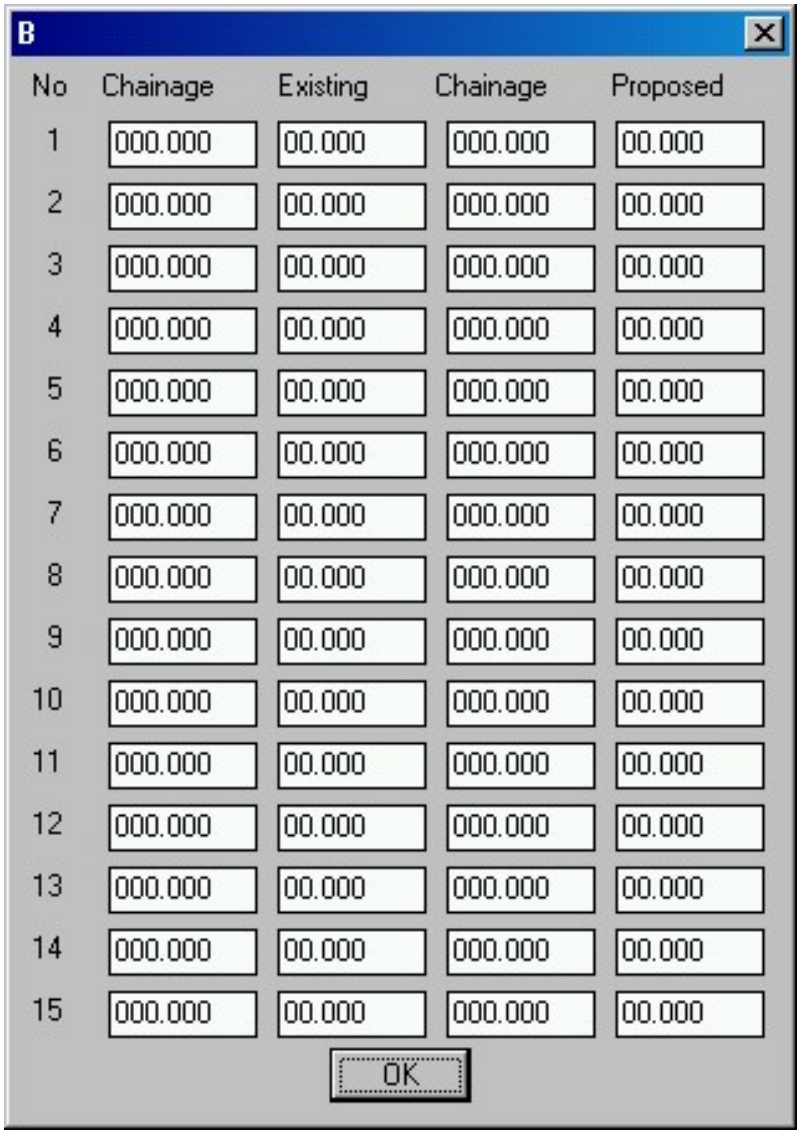
```
afra2 : dialog {
        label = "C" ;

        : column {

                : edit_box {key = "eb16"; label = "Base Level";
                        edit_width = 12;value = "00.000";}
                : edit_box {key = "eb17"; label = "Top Level";
                        edit_width = 12; value = "00.000";}
                : edit_box {key = "eb18"; label = "Horizontal Scale 1 :";
                        edit_width = 12; value = "200";}
                : edit_box {key = "eb19"; label = "Vertical Scale 1 :";
```

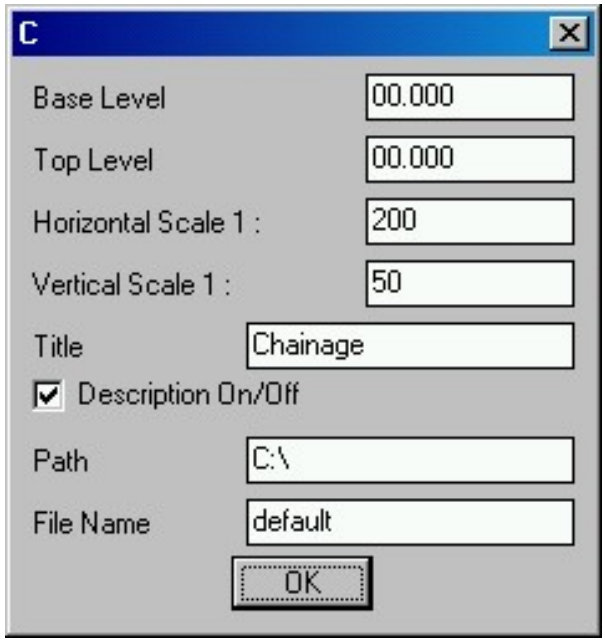
```
                                edit_width = 12; value = "50";}
                : edit_box {key = "eb20"; label = "Title";
                                edit_width = 20; value = "Chainage";}
                : toggle {key = "tg1"; value = 1;
                                label = "Description On/Off";}

                : edit_box {key = "eb21"; label = "Path";
                                edit_width = 20; value = "C:\\";}
                : edit_box {key = "eb22"; label = "File Name";
                                edit_width = 20; value = "default";}
        }

        ok_only;
}
```

This will give us this :



Ok, time now to put this all together. This is what we need to do :





The following coding is a merged version of the three dialog boxes that you have just created:

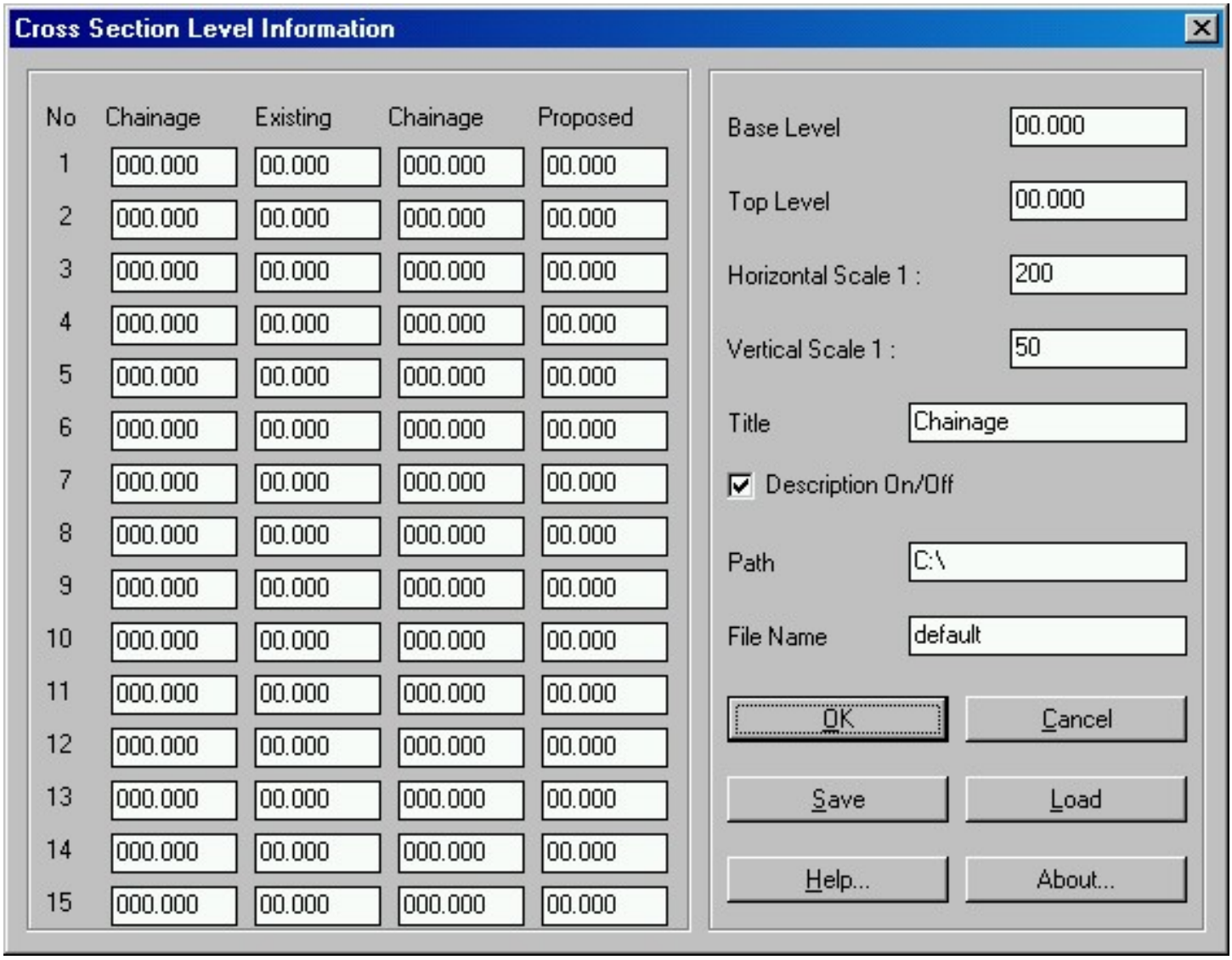
```
afralisp : dialog {
        label = "Cross Section Level Information" ;

    : row {

    : boxed_column {

    : row {

        : text {label = "No";}
        : text {label = "Chainage";}
        : text {label = "Existing";}
        : text {label = "Chainage";}
        : text {label = "Proposed";}

    }

    : row {
        : text {label = "  1";}
        : edit_box {key = eb1; width = 7; value = "000.000";}
        : edit_box {key = eb1a; width = 7; value = "00.000";}
        : edit_box {key = eb1b; width = 7; value = "000.000";}
        : edit_box {key = eb1c; width = 7; value = "00.000";}
    }
```

```
: row {
      : text {label = "  2";}
      : edit_box {key = eb2; width = 7; value = "000.000";}
      : edit_box {key = eb2a; width = 7; value = "00.000";}
      : edit_box {key = eb2b; width = 7; value = "000.000";}
      : edit_box {key = eb2c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  3";}
      : edit_box {key = eb3; width = 7; value = "000.000";}
      : edit_box {key = eb3a; width = 7; value = "00.000";}
      : edit_box {key = eb3b; width = 7; value = "000.000";}
      : edit_box {key = eb3c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  4";}
      : edit_box {key = eb4; width = 7; value = "000.000";}
      : edit_box {key = eb4a; width = 7; value = "00.000";}
      : edit_box {key = eb4b; width = 7; value = "000.000";}
      : edit_box {key = eb4c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  5";}
      : edit_box {key = eb5; width = 7; value = "000.000";}
      : edit_box {key = eb5a; width = 7; value = "00.000";}
      : edit_box {key = eb5b; width = 7; value = "000.000";}
      : edit_box {key = eb5c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  6";}
      : edit_box {key = eb6; width = 7; value = "000.000";}
      : edit_box {key = eb6a; width = 7; value = "00.000";}
      : edit_box {key = eb6b; width = 7; value = "000.000";}
      : edit_box {key = eb6c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  7";}
      : edit_box {key = eb7; width = 7; value = "000.000";}
      : edit_box {key = eb7a; width = 7; value = "00.000";}
      : edit_box {key = eb7b; width = 7; value = "000.000";}
      : edit_box {key = eb7c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  8";}
      : edit_box {key = eb8; width = 7; value = "000.000";}
      : edit_box {key = eb8a; width = 7; value = "00.000";}
      : edit_box {key = eb8b; width = 7; value = "000.000";}
      : edit_box {key = eb8c; width = 7; value = "00.000";}
}
: row {
      : text {label = "  9";}
      : edit_box {key = eb9; width = 7; value = "000.000";}
      : edit_box {key = eb9a; width = 7; value = "00.000";}
      : edit_box {key = eb9b; width = 7; value = "000.000";}
      : edit_box {key = eb9c; width = 7; value = "00.000";}
}
: row {
      : text {label = "10";}
```

```
        : edit_box {key = eb10; width = 7; value = "000.000";}
        : edit_box {key = eb10a; width = 7; value = "00.000";}
        : edit_box {key = eb10b; width = 7; value = "000.000";}
        : edit_box {key = eb10c; width = 7; value = "00.000";}
    }
    : row {
    : text {label = "11";}
        : edit_box {key = eb11; width = 7; value = "000.000";}
        : edit_box {key = eb11a; width = 7; value = "00.000";}
        : edit_box {key = eb11b; width = 7; value = "000.000";}
        : edit_box {key = eb11c; width = 7; value = "00.000";}
    }
    : row {
        : text {label = "12";}
        : edit_box {key = eb12; width = 7; value = "000.000";}
        : edit_box {key = eb12a; width = 7; value = "00.000";}
        : edit_box {key = eb12b; width = 7; value = "000.000";}
        : edit_box {key = eb12c; width = 7; value = "00.000";}
    }
    : row {
        : text {label = "13";}
        : edit_box {key = eb13; width = 7; value = "000.000";}
        : edit_box {key = eb13a; width = 7; value = "00.000";}
        : edit_box {key = eb13b; width = 7; value = "000.000";}
        : edit_box {key = eb13c; width = 7; value = "00.000";}
    }
    : row {
        : text {label = "14";}
        : edit_box {key = eb14; width = 7; value = "000.000";}
        : edit_box {key = eb14a; width = 7; value = "00.000";}
        : edit_box {key = eb14b; width = 7; value = "000.000";}
        : edit_box {key = eb14c; width = 7; value = "00.000";}
    }
    : row {
        : text {label = "15";}
        : edit_box {key = eb15; width = 7; value = "000.000";}
        : edit_box {key = eb15a; width = 7; value = "00.000";}
        : edit_box {key = eb15b; width = 7; value = "000.000";}
        : edit_box {key = eb15c; width = 7; value = "00.000";}
    }

    }


    : boxed_column {

        : edit_box {key = "eb16"; label = "Base Level";
                    edit_width = 12;value = "00.000";}
        : edit_box {key = "eb17"; label = "Top Level";
                    edit_width = 12; value = "00.000";}
        : edit_box {key = "eb18"; label = "Horizontal Scale 1 :";
                    edit_width = 12; value = "200";}
        : edit_box {key = "eb19"; label = "Vertical Scale 1 :";
                    edit_width = 12; value = "50";}
        : edit_box {key = "eb20"; label = "Title";
                    edit_width = 20; value = "Chainage";}
        : toggle {key = "tg1"; value = 1;
                    label = "Description On/Off";}
```

```
        : edit_box {key = "eb21"; label = "Path";
                    edit_width = 20; value = "C:\\";}
        : edit_box {key = "eb22"; label = "File Name";
                    edit_width = 20; value = "default";}

    : row {

    : button {label = "OK"; key = "accept"; mnemonic = "O";
              alignment = centered; width = 12; is_default = true;}
    : button {label = "Cancel"; key = "cancel";mnemonic = "C";
              alignment = centered; width = 12;}

    }

    : row {

    : button {label = "Save"; key = "save";
              mnemonic = "S"; alignment = centered; width = 12;}
    : button {label = "Load";key = "load";
              mnemonic = "L"; alignment = centered; width = 12;}

    }

    : row {

    : button {label = "Help..."; key = "help";
              mnemonic = "H"; alignment = centered; width = 12;}
    : button {label = "About..."; key = "About";
              mnemonic = "H"; alignment = centered; width = 12;}

    }

    }

    }

}
```

Save this as "AfraLisp.dcl".
Your merged dialog will now look like this :



Ok, I admit that it's not exactly the same but it's as close as dammit.
You could, if you are that way inclined, mess around with boxed columns, boxed rows, and spacers, to get a closer match, and if you feel the need, then go for it!!!
Me, I'm out of here to fetch myself a beer.

Previewing DCL Files

Firstly, open AutoCAD.

Now open the Visual Lisp Editor :

Tools -> AutoLisp -> Visual Lisp Editor

Open the DCL file that you wish to view :

File -> Open

Select "*DCL Source Files*" from the "*Files of Type*" and then select your DCL file.

Once your DCL file has opened, select :

Tools -> Interface Tools -> Preview DCL in Editor

Enter your dialog name if it is not already displayed.

Select "*OK*"

Your dialog should be displayed.

Dialog Boxes - Step by Step

This tutorial will take you through all the steps of designing an AutoLISP routine with, a Dialog Box Interface.
It will guide you through the coding of the DCL file, to the retrieval of the data from the Dialog Box.

The dialog box we will be designing will consist of the following :

- A Radio Column consisting of 6 Radio Buttons to allow the user to choose the type of bolt.
- A Drop Down List Box for the size of the bolt.
- An Edit Box for the length of the slot.
- A Slider, also for the length of the slot.
- A Boxed Row containing 2 Toggle Buttons.
- An Edit Box to enter Notes.
- An OK/Cancel pre-defined set of tiles.
- An Image Tile.
- A Paragraph tile containing Text.

Let's start with a very simple Dialog Box containing simply, an OK button and a cancel button.
The dialog box will look like this :

Let's look at the DCL coding for this dialog :

```
samp1 : dialog {                                    //dialog name
      label = "Structural Holes" ;                  //give it a label

     ok_cancel ;                                    //predifined OK/Cancel

     }                                              //end dialog
```

Note the use of the predefined OK/Cancel tile. This is a standard tile set as defined in the AutoCAD Base. DCL File. (You can refer to the AutoCAD Customization Manual for more examples.)

Now for the AutoLISP coding that calls the dialog and handles each tile :

```
(defun C:samp1 ()                                   ;define function

  (setq dcl_id (load_dialog "samp1.dcl"))           ;load dialog

  (if (not (new_dialog "samp1" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

     (action_tile
    "cancel"                                        ;if cancel button pressed
```

```
    "(done_dialog) (setq userclick nil)"                ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                            ;if O.K. pressed
    " (done_dialog)(setq userclick T))"                 ;close dialog, set flag
  );action tile

  (start_dialog)                                        ;start dialog

  (unload_dialog dcl_id)                                ;unload

(princ)

);defun C:samp

(princ)
```

Let's add a paragraph of text to the dialog. Only the DCL file changes here, the AutoLISP coding remains the same except for the function name.
NOTE **: //* preceding the comments denotes new coding :**

The revised DCL coding looks like this :

```
samp2 : dialog {                                    //dialog name
      label = "Structural Holes" ;                  //give it a label


     ok_cancel ;                                    //predifined OK/Cancel

     : paragraph {                                  //*define paragraph

     : text_part {                                  //*define text
     label = "Designed and Created";                //*give it some text
     }                                              //*end text

     : text_part {                                  //*define more text
     label = "by Kenny Ramage";                     //*some more text
     }                                              //*end text

     }                                              //*end paragraph

     }                                              //end dialog
```

And the coding remains the same except for name change :

```
(defun C:samp2 ()                                   ;define function
```

```
  (setq dcl_id (load_dialog "samp2.dcl"))              ;load dialog

  (if (not (new_dialog "samp2" dcl_id)                 ;test for dialog

      );not

    (exit)                                             ;exit if no dialog

  );if

     (action_tile
    "cancel"                                           ;if cancel button pressed
    "(done_dialog) (setq userclick nil)"               ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                           ;if O.K. pressed
    " (done_dialog)(setq userclick T))"                ;close dialog, set flag
  );action tile

  (start_dialog)                                       ;start dialog

  (unload_dialog dcl_id)                               ;unload

(princ)

);defun C:samp

(princ)
```

Now we will add an image tile to make it look nice :

The DCL coding :

```
samp3 : dialog {                                   //dialog name
      label = "Structural Holes" ;                 //give it a label

     ok_cancel ;                                   //predifined OK/Cancel

     : row {                                       //define row

     : image {                                     //*define image tile
     key = "im" ;                                  //*give it a name
     height = 1.0 ;                                //*and a height
     width = 1.0 ;                                 //*and now a width
     }                                             //*end image

     : paragraph {                                 //define paragraph
```

```
    : text_part {                          //define text
    label = "Designed and Created";        //give it some text
    }                                      //end text

    : text_part {                          //define more text
    label = "by Kenny Ramage";             //some more text
    }                                      //end text

    }                                      //end paragraph

    }                                      //end row

    }                                      //end dialog
```

Because we haven't given the image tile a fixed height or a fixed width, it will expand or contract to suit the available space. Now the AutoLISP coding :

```
(defun C:samp3 ()                                   ;define function

  (setq dcl_id (load_dialog "samp3.dcl"))           ;load dialog

  (if (not (new_dialog "samp3" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

  (setq w (dimx_tile "im")                          ;*get image tile width
        h (dimy_tile "im")                          ;*get image tile height

);setq

  (start_image "im")                                ;*start the image
  (fill_image 0 0 w h 5)                            ;*fill it with blue
  (end_image)                                       ;*end image

    (action_tile
    "cancel"                                        ;if cancel button pressed
    "(done_dialog) (setq userclick nil)"            ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                        ;if O.K. pressed
    " (done_dialog)(setq userclick T))"             ;close dialog, set flag
  );action tile

  (start_dialog)                                    ;start dialog

  (unload_dialog dcl_id)                            ;unload

(princ)

);defun C:samp

(princ)
```

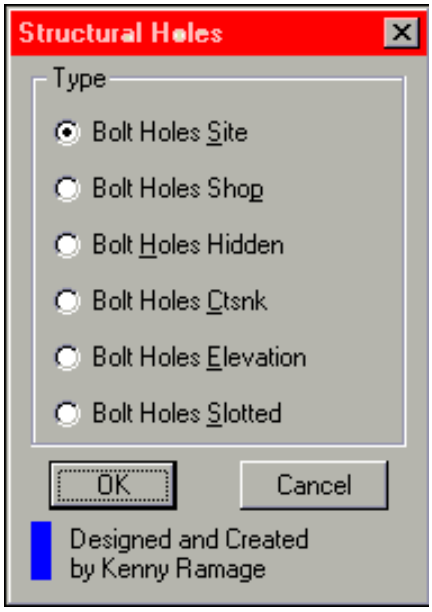
Note the following coding :

```
(setq w (dimx_tile "im")                    ;*get image tile width
      h (dimy_tile "im")                    ;*get image tile height
```

This retrieves the width and height of the image tile and allocates them to variables w and h respectively.
The next code fragment uses these variables to "draw" the image tile.
The number "5" denotes the AutoCAD colour "Blue" :

```
(start_image "im")                          ;*start the image
(fill_image 0 0 w h 5)                      ;*fill it with blue
(end_image)                                 ;*end image
```

We are now going to add a set of radio buttons enclosed in a radio column.
This is to allow the user to select the type of bolt.



The DCL coding :

```
samp4 : dialog {                                    //dialog name
      label = "Structural Holes" ;                  //give it a label

       :boxed_radio_column {                        //*define radio column
       label = "Type" ;                             //*give it a label

        : radio_button {                            //*define radion button
          key = "rb1" ;                             //*give it a name
          label = "Bolt Holes &Site" ;              //*give it a label
          value = "1" ;                             //*switch it on
        }                                           //*end definition

        : radio_button {                            //*define radio button
          key = "rb2" ;                             //*give it a name
          label = "Bolt Holes Sho&p" ;              //*give it a label
        }                                           //*end definition

        : radio_button {                            //*define radio button
          key = "rb3" ;                             //*give it a name
          label = "Bolt Holes &Hidden" ;            //*give it a label
          }                                         //*end definition

        : radio_button {                            //*define radio button
          key = "rb4" ;                             //*give it a name
          label = "Bolt Holes &Ctsnk" ;             //*give it a label
        }                                           //*end definition

        : radio_button {                            //*define radio button
          key = "rb5" ;                             //*give it a name
          label = "Bolt Holes &Elevation" ;         //*give it a label
          }                                         //*end definition

        : radio_button {                            //*define radion button
          key = "rb6" ;                             //*give it a name
```

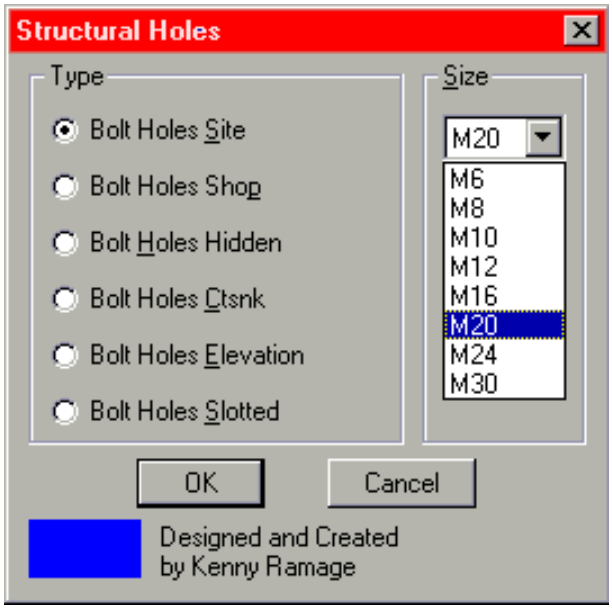
```
        label = "Bolt Holes &Slotted" ;        //*give it a label
      }                                        //*end definition

      }                                        //*end radio column

    ok_cancel ;                                //predifined OK/Cancel

    : row {                                    //define row

    : image {                                  //define image tile
    key = "im" ;                               //give it a name
    height = 1.0 ;                             //and a height
    width = 1.0 ;                              //and now a width
    }                                          //end image

    : paragraph {                              //define paragraph

    : text_part {                              //define text
    label = "Designed and Created";            //give it some text
    }                                          //end text

    : text_part {                              //define more text
    label = "by Kenny Ramage";                 //some more text
    }                                          //end text

    }                                          //end paragraph

    }                                          //end row

    }                                          //end dialog
```

And the AutoLISP coding :

```
(defun C:samp4 ()                                   ;define function

  (setq dcl_id (load_dialog "samp4.dcl"))           ;load dialog

  (if (not (new_dialog "samp4" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

  (setq w (dimx_tile "im")                          ;get image tile width
        h (dimy_tile "im")                          ;get image tile height

);setq

  (start_image "im")                                ;start the image
  (fill_image 0 0 w h 5)                            ;fill it with blue
  (end_image)                                       ;end image

  (action_tile "rb1" "(setq hole \"site\")")        ;*store hole type
  (action_tile "rb2" "(setq hole \"shop\")")        ;*store hole type
  (action_tile "rb3" "(setq hole \"hid\")")         ;*store hole type
```

```
  (action_tile "rb4" "(setq hole \"ctsk\")")          ;*store hole type
  (action_tile "rb5" "(setq hole \"elev\")")          ;*store hole type
  (action_tile "rb6" "(setq hole \"slot\")")          ;*store hole type


    (action_tile
    "cancel"                                          ;if cancel button pressed
    "(done_dialog) (setq userclick nil)"              ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                          ;if O.K. pressed
    " (done_dialog)(setq userclick T))"               ;close dialog, set flag
  );action tile

  (start_dialog)                                      ;start dialog

  (unload_dialog dcl_id)                              ;unload

(princ)

);defun C:samp

(princ)
```

Now we'll add a drop down list so that we can select the bolt size.



The DCL coding :

```
samp5 : dialog {                                  //dialog name
      label = "Structural Holes" ;                //give it a label

       : row {                                    //*define row

       :boxed_radio_column {                      //define radio column
       label = "Type" ;                           //give it a label

        : radio_button {                          //define radion button
```

```
      key = "rb1" ;                          //give it a name
      label = "Bolt Holes &Site" ;           //give it a label
      value = "1" ;                          //switch it on
    }                                        //end definition

    : radio_button {                         //define radio button
      key = "rb2" ;                          //give it a name
      label = "Bolt Holes Sho&p" ;           //give it a label
    }                                        //end definition

    : radio_button {                         //define radio button
      key = "rb3" ;                          //give it a name
      label = "Bolt Holes &Hidden" ;         //give it a label
      }                                      //end definition

    : radio_button {                         //define radio button
      key = "rb4" ;                          //give it a name
      label = "Bolt Holes &Ctsnk" ;          //give it a label
    }                                        //end definition

    : radio_button {                         //define radio button
      key = "rb5" ;                          //give it a name
      label = "Bolt Holes &Elevation" ;      //give it a label
      }                                      //end definition

    : radio_button {                         //define radion button
      key = "rb6" ;                          //give it a name
      label = "Bolt Holes &Slotted" ;        //give it a label
    }                                        //end definition

    }                                        //end radio column

    : boxed_column {                         //*define boxed column
    label = "&Size";                         //*give it a label

    : popup_list {                           //*define popup list
    key = "selections";                      //*give it a name
    value = "5" ;                            //*initial value
    }                                        //*end list

  }                                          //*end boxed column

  }                                          //*end row

ok_cancel ;                                  //predifined OK/Cancel

: row {                                      //define row

: image {                                    //define image tile
key = "im" ;                                 //give it a name
height = 1.0 ;                               //and a height
width = 1.0 ;                                //and now a width
}                                            //end image

: paragraph {                                //define paragraph

: text_part {                                //define text
label = "Designed and Created";              //give it some text
```

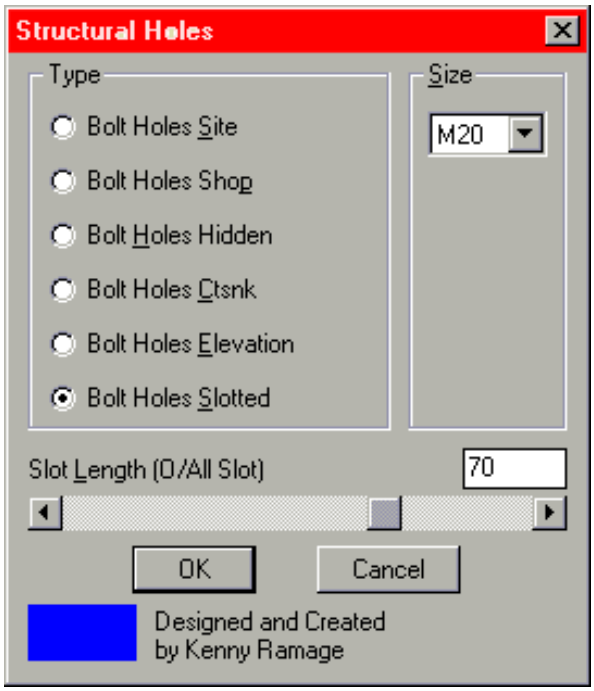
```
    }                                           //end text

    : text_part {                               //define more text
    label = "by Kenny Ramage";                  //some more text
    }                                           //end text

    }                                           //end paragraph

    }                                           //end row

    }                                           //end dialog
```

Note how we have put the Radio Column and the Drop Down List box into a Row. Looks good hey....Now the AutoLISP coding :

```
(defun C:samp5 ()                                   ;define function

  (setq siz "M20")                                  ;*preset hole size

  (setq NAMES '("M6" "M8" "M10" "M12"
                "M16" "M20" "M24" "M30")            ;*define list
  );setq

  (setq dcl_id (load_dialog "samp5.dcl"))           ;load dialog

  (if (not (new_dialog "samp5" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

  (setq w (dimx_tile "im")                          ;get image tile width
        h (dimy_tile "im")                          ;get image tile height

);setq

  (start_image "im")                                ;start the image
  (fill_image 0 0 w h 5)                            ;fill it with blue
  (end_image)                                       ;end image

  (start_list "selections")                         ;*start the list box
  (mapcar 'add_list NAMES)                          ;*fill the list box
  (end_list)                                        ;*end list

  (action_tile "rb1" "(setq hole \"site\")")        ;store hole type
  (action_tile "rb2" "(setq hole \"shop\")")        ;store hole type
  (action_tile "rb3" "(setq hole \"hid\")")         ;store hole type
  (action_tile "rb4" "(setq hole \"ctsk\")")        ;store hole type
  (action_tile "rb5" "(setq hole \"elev\")")        ;store hole type
  (action_tile "rb6" "(setq hole \"slot\")")        ;store hole type


    (action_tile
    "cancel"                                        ;if cancel button pressed
```

```
    "(done_dialog) (setq userclick nil)"                ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                            ;if O.K. pressed
    (strcat                                             ;string 'em together
      "(progn
       (setq SIZ (atof (get_tile \"selections\")))"     ;*get list selection
      " (done_dialog)(setq userclick T))"               ;close dialog, set flag
    );strcat
  );action tile

  (start_dialog)                                        ;start dialog

  (unload_dialog dcl_id)                                ;unload

   (if userclick                                        ;*check O.K. was selected
    (progn

      (setq SIZ (fix SIZ))                              ;*convert to integer
      (setq SIZ (nth SIZ NAMES))                        ;*get the size

    );progn

  );if userclick

(princ)

);defun C:samp

(princ)
```

Let us now add an Edit Box and a slider so that the user can enter the size of the slot, if a slotted hole is chosen.
Note that the Edit Box and Slider only become active if the slotted bolt radio button is selected.



The DCL coding :

```
samp6 : dialog {                                  //dialog name
      label = "Structural Holes" ;                //give it a label

       : row {                                    //define row

       :boxed_radio_column {                      //define radio column
       label = "Type" ;                           //give it a label

        : radio_button {                          //define radion button
          key = "rb1" ;                           //give it a name
          label = "Bolt Holes &Site" ;            //give it a label
          value = "1" ;                           //switch it on
        }                                         //end definition

        : radio_button {                          //define radio button
          key = "rb2" ;                           //give it a name
          label = "Bolt Holes Sho&p" ;            //give it a label
        }                                         //end definition

        : radio_button {                          //define radio button
          key = "rb3" ;                           //give it a name
          label = "Bolt Holes &Hidden" ;          //give it a label
          }                                       //end definition

        : radio_button {                          //define radio button
          key = "rb4" ;                           //give it a name
          label = "Bolt Holes &Ctsnk" ;           //give it a label
        }                                         //end definition

        : radio_button {                          //define radio button
          key = "rb5" ;                           //give it a name
          label = "Bolt Holes &Elevation" ;       //give it a label
```

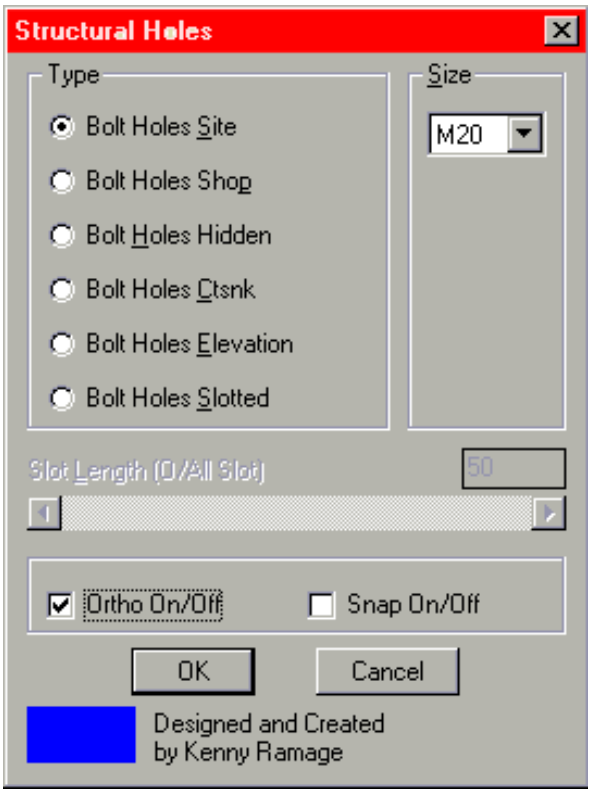
```
    }                                          //end definition

   : radio_button {                            //define radion button
     key = "rb6" ;                             //give it a name
     label = "Bolt Holes &Slotted" ;           //give it a label
   }                                           //end definition

   }                                           //end radio column

   : boxed_column {                            //define boxed column
   label = "&Size";                            //give it a label

   : popup_list {                              //define popup list
   key = "selections";                         //give it a name
   value = "5" ;                               //initial value
   }                                           //end list

  }                                            //end boxed column

  }                                            //end row

  : edit_box {                                 //*define edit box
    key = "eb1" ;                              //*give it a name
    label = "Slot &Length (O/All Slot)" ;      //*give it a label
    edit_width = 6 ;                           //*6 characters only
   }                                           //*end edit box

   : slider {                                  //*defin slider
   key = "myslider" ;                          //*give it a name
   max_value = 100;                            //*upper value
   min_value = 0;                              //*lower value
   value = "50";                               //*initial value
   }                                           //*end slider

ok_cancel ;                                    //predifined OK/Cancel

: row {                                        //define row

: image {                                      //define image tile
key = "im" ;                                   //give it a name
height = 1.0 ;                                 //and a height
width = 1.0 ;                                  //and now a width
}                                              //end image

: paragraph {                                  //define paragraph

: text_part {                                  //define text
label = "Designed and Created";                //give it some text
}                                              //end text

: text_part {                                  //define more text
label = "by Kenny Ramage";                     //some more text
}                                              //end text

}                                              //end paragraph

}                                              //end row

}                                              //end dialog
```

And now the AutoLISP coding :

```
(defun C:samp6 ()                                   ;define function

  (setq lngth 50.0)                                 ;*preset slot length

  (setq hole "site")                                ;preset hole type

  (setq siz "M20")                                  ;preset hole size

  (setq NAMES '("M6" "M8" "M10" "M12"
                "M16" "M20" "M24" "M30")            ;define list
  );setq

  (setq dcl_id (load_dialog "samp6.dcl"))           ;load dialog

  (if (not (new_dialog "samp6" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

  (setq w (dimx_tile "im")                          ;get image tile width
        h (dimy_tile "im")                          ;get image tile height

);setq

  (start_image "im")                                ;start the image
  (fill_image 0 0 w h 5)                            ;fill it with blue
  (end_image)                                       ;end image

  (start_list "selections")                         ;start the list box
  (mapcar 'add_list NAMES)                          ;fill the list box
  (end_list)                                        ;end list

  (set_tile "eb1" "50")                             ;*put data into edit box
  (mode_tile "eb1" 1)                               ;*disable edit box
  (mode_tile "myslider" 1)                          ;*disable slider

  (action_tile "myslider"                           ;*if user moves slider
         "(slider_action $value $reason)")          ;*pass arguments to
slider_action

  (action_tile "eb1"                                ;*is user enters slot length
         "(ebox_action $value $reason)")            ;*pass arguments to
ebox_action

  (defun slider_action (val why)                    ;*define function
      (if (or (= why 2) (= why 1))                  ;*check values
      (set_tile "eb1" val)))                        ;*update edit box

  (defun ebox_action (val why)                      ;*define function
      (if (or (= why 2) (= why 1))                  ;*check values
      (set_tile "myslider" val)))                   ;*update slider

  (action_tile "rb1" "(setq hole \"site\")")        ;store hole type
  (action_tile "rb2" "(setq hole \"shop\")")        ;store hole type
  (action_tile "rb3" "(setq hole \"hid\")")         ;store hole type
```

```
  (action_tile "rb4" "(setq hole \"ctsk\")")           ;store hole type
  (action_tile "rb5" "(setq hole \"elev\")")           ;store hole type
  (action_tile "rb6" "(setq hole \"slot\")             ;store hole type
                      (mode_tile \"eb1\" 0)            ;*enable edit box
                      (mode_tile \"myslider\" 0)       ;*enable slider
                      (mode_tile \"eb1\" 2)")          ;*switch focus to edit box

    (action_tile
    "cancel"                                           ;if cancel button pressed
    "(done_dialog) (setq userclick nil)"               ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                           ;if O.K. pressed
    (strcat                                            ;string 'em together
      "(progn
       (setq SIZ (atof (get_tile \"selections\")))"    ;get list selection
      "(setq lngth (atof (get_tile \"eb1\")))"         ;*get slot length
      " (done_dialog)(setq userclick T))"              ;close dialog, set flag
    );strcat
  );action tile

  (start_dialog)                                       ;start dialog

  (unload_dialog dcl_id)                               ;unload

   (if userclick                                       ;check O.K. was selected
    (progn

      (setq SIZ (fix SIZ))                             ;convert to integer
      (setq SIZ (nth SIZ NAMES))                       ;get the size

    );progn

  );if userclick

(princ)

);defun C:samp

(princ)
```

Now, for the fun of it, let's add some toggles :



Here is the DCL coding :

```
samp7 : dialog {                                    //dialog name
      label = "Structural Holes" ;                  //give it a label

       : row {                                      //define row

       :boxed_radio_column {                        //define radio column
       label = "Type" ;                             //give it a label

        : radio_button {                            //define radion button
          key = "rb1" ;                             //give it a name
          label = "Bolt Holes &Site" ;              //give it a label
          value = "1" ;                             //switch it on
        }                                           //end definition

        : radio_button {                            //define radio button
          key = "rb2" ;                             //give it a name
          label = "Bolt Holes Sho&p" ;              //give it a label
        }                                           //end definition

        : radio_button {                            //define radio button
          key = "rb3" ;                             //give it a name
          label = "Bolt Holes &Hidden" ;            //give it a label
          }                                         //end definition

        : radio_button {                            //define radio button
          key = "rb4" ;                             //give it a name
          label = "Bolt Holes &Ctsnk" ;             //give it a label
        }                                           //end definition

        : radio_button {                            //define radio button
          key = "rb5" ;                             //give it a name
          label = "Bolt Holes &Elevation" ;         //give it a label
          }                                         //end definition
```

```
   : radio_button {                          //define radion button
     key = "rb6" ;                           //give it a name
     label = "Bolt Holes &Slotted" ;         //give it a label
   }                                         //end definition

   }                                         //end radio column

   : boxed_column {                          //define boxed column
   label = "&Size";                          //give it a label

   : popup_list {                            //define popup list
   key = "selections";                       //give it a name
   value = "5" ;                             //initial value
   }                                         //end list

  }                                          //end boxed column

  }                                          //end row

  : edit_box {                               //define edit box
    key = "eb1" ;                            //give it a name
    label = "Slot &Length (O/All Slot)" ;    //give it a label
    edit_width = 6 ;                         //6 characters only
   }                                         //end edit box

   : slider {                                //defin slider
   key = "myslider" ;                        //give it a name
   max_value = 100;                          //upper value
   min_value = 0;                            //lower value
   value = "50";                             //initial value
   }                                         //end slider

   :boxed_row {                              //*define boxed row

   :toggle {                                 //*define toggle
   key = "tog1";                             //*give it a name
   label = "Ortho On/Off";                   //*give it a label
   }                                         //*end toggle

   :toggle {                                 //*define toggle
   key = "tog2";                             //*give it a name
   label = "Snap On/Off";                    //*give it a label
   }                                         //*end definition

   }                                         //*end boxed row

ok_cancel ;                                  //predifined OK/Cancel

: row {                                      //define row

: image {                                    //define image tile
key = "im" ;                                 //give it a name
height = 1.0 ;                               //and a height
width = 1.0 ;                                //and now a width
}                                            //end image

: paragraph {                                //define paragraph

: text_part {                                //define text
label = "Designed and Created";              //give it some text
```

```
    }                                       //end text

    : text_part {                           //define more text
    label = "by Kenny Ramage";              //some more text
    }                                       //end text

    }                                       //end paragraph

    }                                       //end row

    }                                       //end dialog
```

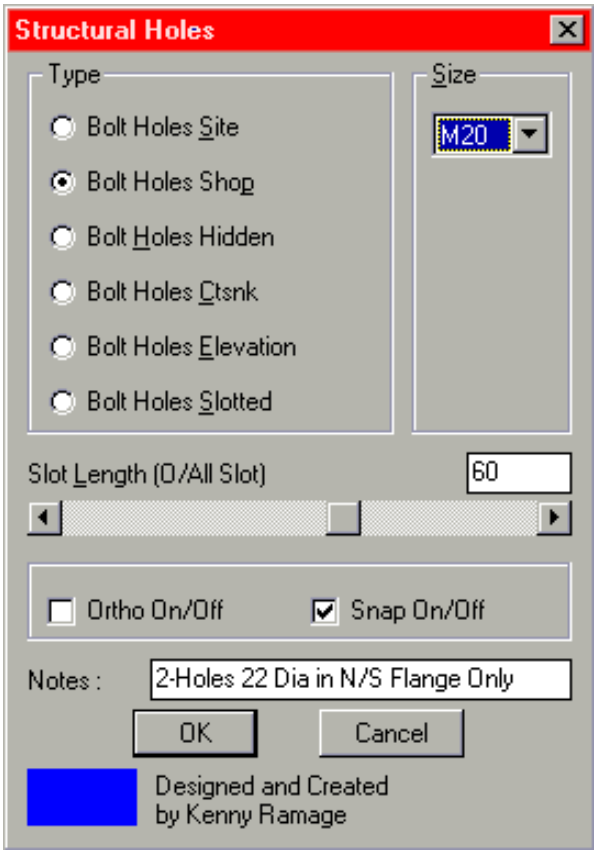
And now the AutoLISP coding :

```
(defun C:samp7 ()                                   ;define function

  (setq lngth 50.0)                                 ;preset slot length

  (setq hole "site")                                ;preset hole type

  (setq siz "M20")                                  ;preset hole size

  (setq NAMES '("M6" "M8" "M10" "M12"
                "M16" "M20" "M24" "M30")            ;define list
  );setq

  (setq dcl_id (load_dialog "samp7.dcl"))           ;load dialog

  (if (not (new_dialog "samp7" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

  (setq w (dimx_tile "im")                          ;get image tile width
        h (dimy_tile "im")                          ;get image tile height

);setq

  (start_image "im")                                ;start the image
  (fill_image 0 0 w h 5)                            ;fill it with blue
  (end_image)                                       ;end image

  (start_list "selections")                         ;start the list box
  (mapcar 'add_list NAMES)                          ;fill the list box
  (end_list)                                        ;end list

  (set_tile "eb1" "50")                             ;put dat into edit box
  (mode_tile "eb1" 1)                               ;disable edit box
  (mode_tile "myslider" 1)                          ;disable slider

  (setq orth (itoa (getvar "orthomode")))           ;*get orthomode value
  (set_tile "tog1" orth)                            ;*switch toggle on or off

  (setq sna (itoa (getvar "snapmode")))             ;*get snap value
  (set_tile "tog2" sna)                             ;*switch toggle on or off
```

```
  (action_tile "myslider"                                ;if user moves slider
        "(slider_action $value $reason)")                ;pass arguments to
slider_action

  (action_tile "eb1"                                     ;is user enters slot length
        "(ebox_action $value $reason)")                  ;pass arguments to
ebox_action

  (defun slider_action (val why)                         ;define function
     (if (or (= why 2) (= why 1))                        ;check values
     (set_tile "eb1" val)))                              ;update edit box

  (defun ebox_action (val why)                           ;define function
     (if (or (= why 2) (= why 1))                        ;check values
     (set_tile "myslider" val)))                         ;update slider

  (action_tile "tog1" "(setq orth $value)")              ;*get ortho toggle value
  (action_tile "tog2" "(setq sna $value)")               ;*get snap toggle value

  (action_tile "rb1" "(setq hole \"site\")")             ;store hole type
  (action_tile "rb2" "(setq hole \"shop\")")             ;store hole type
  (action_tile "rb3" "(setq hole \"hid\")")              ;store hole type
  (action_tile "rb4" "(setq hole \"ctsk\")")             ;store hole type
  (action_tile "rb5" "(setq hole \"elev\")")             ;store hole type
  (action_tile "rb6" "(setq hole \"slot\")               ;store hole type
                      (mode_tile \"eb1\" 0)              ;enable edit box
                      (mode_tile \"myslider\" 0)         ;enable slider
                      (mode_tile \"eb1\" 2)")            ;switch focus to edit box

    (action_tile
    "cancel"                                             ;if cancel button pressed
    "(done_dialog) (setq userclick nil)"                 ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                             ;if O.K. pressed
    (strcat                                              ;string 'em together
      "(progn
       (setq SIZ (atof (get_tile \"selections\")))"      ;get list selection
      "(setq lngth (atof (get_tile \"eb1\")))"           ;get slot length
      "(setvar \"orthomode\" (atoi orth))"               ;*ortho on/off
      "(setvar \"snapmode\" (atoi sna))"                 ;*snap on/off
      " (done_dialog)(setq userclick T))"                ;close dialog, set flag
    );strcat
  );action tile

  (start_dialog)                                         ;start dialog

  (unload_dialog dcl_id)                                 ;unload

   (if userclick                                         ;check O.K. was selected
    (progn

      (setq SIZ (fix SIZ))                               ;convert to integer
      (setq SIZ (nth SIZ NAMES))                         ;get the size

    );progn

  );if userclick
```

```
(princ)
);defun C:samp
(princ)
```

To finish off the box, we'll add an Edit Box so that the user can include some notes, if he so wishes.



The Complete DCL coding :

```
samp8 : dialog {                                    //dialog name
      label = "Structural Holes" ;                  //give it a label

       : row {                                      //define row

       :boxed_radio_column {                        //define radio column
       label = "Type" ;                             //give it a label

        : radio_button {                            //define radion button
          key = "rb1" ;                             //give it a name
          label = "Bolt Holes &Site" ;              //give it a label
          value = "1" ;                             //switch it on
        }                                           //end definition

        : radio_button {                            //define radio button
          key = "rb2" ;                             //give it a name
          label = "Bolt Holes Sho&p" ;              //give it a label
        }                                           //end definition

        : radio_button {                            //define radio button
          key = "rb3" ;                             //give it a name
          label = "Bolt Holes &Hidden" ;            //give it a label
          }                                         //end definition

        : radio_button {                            //define radio button
          key = "rb4" ;                             //give it a name
          label = "Bolt Holes &Ctsnk" ;             //give it a label
        }                                           //end definition
```

```
   : radio_button {                          //define radio button
     key = "rb5" ;                           //give it a name
     label = "Bolt Holes &Elevation" ;       //give it a label
     }                                       //end definition

   : radio_button {                          //define radion button
     key = "rb6" ;                           //give it a name
     label = "Bolt Holes &Slotted" ;         //give it a label
   }                                         //end definition

   }                                         //end radio column

   : boxed_column {                          //define boxed column
   label = "&Size";                          //give it a label

   : popup_list {                            //define popup list
   key = "selections";                       //give it a name
   value = "5" ;                             //initial value
   }                                         //end list

  }                                          //end boxed column

  }                                          //end row

  : edit_box {                               //define edit box
    key = "eb1" ;                            //give it a name
    label = "Slot &Length (O/All Slot)" ;    //give it a label
    edit_width = 6 ;                         //6 characters only
   }                                         //end edit box

   : slider {                                //defin slider
   key = "myslider" ;                        //give it a name
   max_value = 100;                          //upper value
   min_value = 0;                            //lower value
   value = "50";                             //initial value
   }                                         //end slider

   :boxed_row {                              //define boxed row

   :toggle {                                 //define toggle
   key = "tog1";                             //give it a name
   label = "Ortho On/Off";                   //give it a label
   }                                         //end toggle

   :toggle {                                 //define toggle
   key = "tog2";                             //give it a name
   label = "Snap On/Off";                    //give it a label
   }                                         //end definition

   }                                         //end boxed row

   : edit_box {                              //*define edit box
    key = "eb2" ;                            //*give it a name
    label = "Notes :" ;                      //*give it a label
    edit_width = 30 ;                        //*30 characters
   }                                         //*end edit box

ok_cancel ;                                  //predifined OK/Cancel
```

```
    : row {                                  //define row

    : image {                                //define image tile
    key = "im" ;                             //give it a name
    height = 1.0 ;                           //and a height
    width = 1.0 ;                            //and now a width
    }                                        //end image

    : paragraph {                            //define paragraph

    : text_part {                            //define text
    label = "Designed and Created";          //give it some text
    }                                        //end text

    : text_part {                            //define more text
    label = "by Kenny Ramage";               //some more text
    }                                        //end text

    }                                        //end paragraph

    }                                        //end row

    }                                        //end dialog
```

And Now the Complete AutoLISP coding :

```
(defun C:samp8 ()                                   ;define function

  (setq lngth 50.0)                                 ;preset slot length

  (setq hole "site")                                ;preset hole type

  (setq siz "M20")                                  ;preset hole size

  (setq NAMES '("M6" "M8" "M10" "M12"
                "M16" "M20" "M24" "M30")            ;define list
  );setq

  (setq dcl_id (load_dialog "samp8.dcl"))           ;load dialog

  (if (not (new_dialog "samp8" dcl_id)              ;test for dialog

      );not

    (exit)                                          ;exit if no dialog

  );if

  (setq w (dimx_tile "im")                          ;get image tile width
        h (dimy_tile "im")                          ;get image tile height

);setq

  (start_image "im")                                ;start the image
  (fill_image 0 0 w h 5)                            ;fill it with blue
  (end_image)                                       ;end image

  (start_list "selections")                         ;start the list box
```

```
  (mapcar 'add_list NAMES)                          ;fill the list box
  (end_list)                                        ;end list

  (set_tile "eb1" "50")                             ;put dat into edit box
  (mode_tile "eb1" 1)                               ;disable edit box
  (mode_tile "myslider" 1)                          ;disable slider

  (setq orth (itoa (getvar "orthomode")))           ;get orthomode value
  (set_tile "tog1" orth)                            ;switch toggle on or off

  (setq sna (itoa (getvar "snapmode")))             ;get snap value
  (set_tile "tog2" sna)                             ;switch toggle on or off

  (action_tile "myslider"                           ;if user moves slider
         "(slider_action $value $reason)")          ;pass arguments to
slider_action

  (action_tile "eb1"                                ;is user enters slot length
         "(ebox_action $value $reason)")            ;pass arguments to
ebox_action

  (defun slider_action (val why)                    ;define function
      (if (or (= why 2) (= why 1))                  ;check values
      (set_tile "eb1" val)))                        ;update edit box

  (defun ebox_action (val why)                      ;define function
      (if (or (= why 2) (= why 1))                  ;check values
      (set_tile "myslider" val)))                   ;update slider

  (action_tile "tog1" "(setq orth $value)")         ;get ortho toggle value
  (action_tile "tog2" "(setq sna $value)")          ;get snap toggle value

  (action_tile "rb1" "(setq hole \"site\")")        ;store hole type
  (action_tile "rb2" "(setq hole \"shop\")")        ;store hole type
  (action_tile "rb3" "(setq hole \"hid\")")         ;store hole type
  (action_tile "rb4" "(setq hole \"ctsk\")")        ;store hole type
  (action_tile "rb5" "(setq hole \"elev\")")        ;store hole type
  (action_tile "rb6" "(setq hole \"slot\")          ;store hole type
                      (mode_tile \"eb1\" 0)         ;enable edit box
                      (mode_tile \"myslider\" 0)    ;enable slider
                      (mode_tile \"eb1\" 2)")       ;switch focus to edit box

    (action_tile
    "cancel"                                        ;if cancel button pressed
    "(done_dialog) (setq userclick nil)"            ;close dialog, set flag
    );action_tile

  (action_tile
    "accept"                                        ;if O.K. pressed
    (strcat                                         ;string 'em together
      "(progn
       (setq SIZ (atof (get_tile \"selections\")))"  ;get list selection
      "(setq lngth (atof (get_tile \"eb1\")))"      ;get slot length
      "(setq notes (get_tile \"eb2\"))"             ;*get notes
      "(setvar \"orthomode\" (atoi orth))"          ;ortho on/off
      "(setvar \"snapmode\" (atoi sna))"            ;snap on/off
      " (done_dialog)(setq userclick T))"           ;close dialog, set flag
    );strcat
  );action tile
```

```
  (start_dialog)                                    ;start dialog

  (unload_dialog dcl_id)                            ;unload

   (if userclick                                    ;check O.K. was selected
    (progn

      (setq SIZ (fix SIZ))                          ;convert to integer
      (setq SIZ (nth SIZ NAMES))                    ;get the size

    );progn

  );if userclick

(princ)

);defun C:samp

(princ)
```

I have deliberately left all variables local, so that you can check their values at the command line.

That's it Folks....Enjoy your Dialog Boxes.....

Dialog Boxes in Action

So here we are. Dialog boxes, the bane of all AutoLispers!
I must say they are quite a pain in the backside. This is when Visual Basic comes into its own.
Creating Dialog Boxes using VB is a pure pleasure. (It's the rest that's a pain).
Anyway, I digress, so stuff the torpedoes and full steam ahead.

In this tutorial we are going to try to write a program that draws structural steel beams. The user will select the size of beam from a list in a dialog box. The data for the beam will then be retrieved from an external data file. We will then draw the beam without touching our mouse or keyboard. (Pure Magic!!)
This is a complete working example of parametric programming.

First, lets start by writing the data file.
Create a file named BEAM.DAT containing the following :

```
**UNIVERSAL BEAMS DATA
**----------------------
**H X B X T1 X T2 X R1
**----------------------
*100x55x8
100.0,55.0,4.1,5.7,7.0
*120x64x10
120.0,64.0,4.4,6.3,7.0
*140x73x13
140.0,73.0,4.7,6.9,7.0
*160x82x16
160.0,82.0,5.0,7.4,9.0
*180x91x19
180.0,91.0,5.3,8.0,9.0
*200x100x22
200.0,100.0,5.6,8.5,12.0
*203x133x25
203.2,133.4,5.8,7.8,7.6
*203x133x30
206.8,133.8,6.3,9.6,7.6
*254x146x31
251.5,146.1,6.1,8.6,7.6
*254x146x37
256.0,146.4,6.4,10.9,7.6
*254x146x43
259.6,147.3,7.3,12.7,7.6
*305x102x25
**------------------------
```

The first line (*100x55x8) is the name of the beam. (100 deep x 50 wide x 8 kg/m beam.)
The second line contains the physical dimensions that we need to draw the beam.
(100.0,55.0,4.1,5.7,7.0) (Height, width, web thickness, flange thickness and root radius.)

Next we need to write our DCL (Dialog Control Language) file.
Name this file BEAM.DCL. The coding is below :

```
beam : dialog {                          //dialog name
       label = "Beams.";                 //dialog label
     : list_box {                        //list box
        label = "&Choose Section :";     //list box label
        key = "selections";              //key to list
        height = 12;                     //height
        allow_accept = true ;            //allow double clicking
     }
     ok_cancel ;                         //OK and Cancel Buttons
     :text_part {                        //Text Label
      label = "Designed and Created";
      }
     :text_part {                        //Text Label
      label = "by Kenny Ramage";
      }
     }
```

As you can see, the first line is simply the name of the dialog box.
Remember, this is the name that AutoLisp knows the box by.
The second line is the label attribute. This is the wording that you see at the top of the box when it is displayed.

The third line displays a list box. This has it's own attributes :

A label attribute - Wording that is displayed above the list box.

A key attribute - This is the key that Autolisp uses to refer to the list.
Height - Simply the height of the box. If your list exceeds the height of the list box vertical scroll bars are added.

Allow_accept - Instead of selecting an item from the list and then the OK button, this allows you to select by double-clicking the list item.

The next section displays pre-constructed OK and Cancel buttons.
(Refer to the AutoCAD Customisation Manual for more examples of these.)
The last section is were you put your name, E-Mail address, Tel. Number, Thank You notes to me, etc.

Take note of were the { and } occur. DCL is very similar to AutoLISP in that what you open, you must close.

O.K Time for a rest and a cup of tea. (brantea!!)

What! Back already?
O.K. Now let us try and tie this lot together.
Time for the AutoLisp Coding. Ready, here we go :

```
(defun DTR (a)                                  ;convert to radians
  (* pi (/ a 180.0))
);defun

;;;*============================================================
```

```
(defun C:BEAM (/)                                   ;function defined

;;;*Initialise and Setup Dialog Box

  (setq OLDECHO (getvar "CMDECHO")                  ;store system variables
        OLDBLIP (getvar "BLIPMODE")
        OLDSNAP (getvar "OSMODE")
  );setq
  (setvar "CMDECHO" 0)
  (setvar "BLIPMODE" 0)
  (setq NAMES '("100x55x8" "120x64x10" "140x73x13" "160x82x16"
                "180x91x19" "200x100x22" "203x133x25" "203x133x30"
                "254x146x31" "254x146x37" "254x146x43");list of names
  );setq
  (setq dialogshow T)                               ;set flag
  (setq dcl_id (load_dialog "beam.dcl"))            ;initialise dialog box
  (if (not                                          ;check dcl exists
        (new_dialog "beam" dcl_id)                  ;load into memory
      );not
    (setq dialogshow nil)                           ;if not exit
  );if
(if dialogshow                                      ;if dialog
(progn                                              ;continue with programme
(start_list "selections")                           ;start list box
  (mapcar 'add_list NAMES)                          ;add names from list
  (end_list)                                        ;end list box
  (action_tile
    "cancel"                                        ;if cancel selected
    "(done_dialog) (setq userclick1 nil)"           ;close dialog, set flag
  );action tile
  (action_tile
    "accept"                                        ;if OK or double click
    (strcat
      "(progn (setq SIZA (atof (get_tile \"selections\")))" ;get size of beam
      "(done_dialog) (setq userclick1 T))"          ;close dialog and set flag
    );strcat
  );action tile
  (start_dialog)                                    ;display dialog
  (unload_dialog dcl_id)                            ;unload dialog
  (if userclick1                                    ;if OK or double click
    (progn                                          ;continue with programme
      (setq SIZA (fix SIZA))                        ;convert index to integer
      (setq SIZA (nth SIZA NAMES))                  ;retrieve name from list

;;;*Retrieve Data from External Data File

;;;*=================================================
;;;*This section is covered in the External Data Tutor
;;;*=================================================

      (setq dlist nil
            size  (strcat "*" SIZA)
            file  (findfile "beam.dat")
```

```
            fp    (open file "r")
            item  (read-line fp)
  );setq
  (while item
    (if (= item size)
      (setq data (read-line fp)
            item nil
      );setq
      (setq item (read-line fp))
    );if
  );while

;;;*Format List

  (if data
    (progn
      (setq maxs  (strlen data)
            count 1
            chrct 1
      );setq
      (while (< count maxs)
        (if (/= "," (substr data count 1))
          (setq chrct (1+ chrct))
          (setq numb  (atof (substr data (1+ (- count chrct)) chrct))
                dlist (append dlist (list numb))
                chrct 1
          );setq
        );if
        (setq count (1+ count))
      );while
      (setq numb  (atof (substr data (1+ (- count chrct))))
            dlist (append dlist (list numb))
      );setq
    );progn
  );if data
  (close fp)

;;;*This routine draws the beam.

;;;*=======================================================
;;;*This section is covered in the Calculating Points Tutor
;;;*=======================================================

     (mapcar 'set '(H B T1 T2 R1) dlist)
      (setq OLDSNAP (getvar "OSMODE"))
      (while
      (setq IP (getpoint "\nInsertion Point: "))
      (setvar "OSMODE" 0)
      (setq P1  (polar IP 0 (/ T1 2))
            P2  (polar P1 (DTR 90.0) (/ (- H (+ T2 T2 R1 R1)) 2))
            P3  (polar P2 (DTR 90.0) R1)
            P4  (polar P3 0 R1)
            P5  (polar P4 0 (/ (- B (+ T1 R1 R1)) 2))
```

```
            P6  (polar P5 (DTR 90.0) T2)
            P7  (polar P6 (DTR 180.0) B)
            P8  (polar P7 (DTR 270.0) T2)
            P9  (polar P8 0 (/ (- B (+ T1 R1 R1)) 2))
            P10 (polar P9 0 R1)
            P11 (polar P10 (DTR 270.0) R1)
            P12 (polar P11 (DTR 270.0) (- H (+ T2 T2 R1 R1)))
            P13 (polar P12 (DTR 270.0) R1)
            P14 (polar P13 (DTR 180.0) R1)
            P15 (polar P14 (DTR 180.0) (/ (- B (+ T1 R1 R1)) 2))
            P16 (polar P15 (DTR 270.0) T2)
            P17 (polar P16 0 B)
            P18 (polar P17 (DTR 90.0) T2)
            P19 (polar P18 (DTR 180.0) (/ (- B (+ T1 R1 R1)) 2))
            P20 (polar P19 (DTR 180.0) R1)
            P21 (polar P20 (DTR 90.0) R1)
      );setq
      (command "PLINE" P1 "W" "0.0" "0.0" P2 "ARC"
               P4 "LINE" P5 P6 P7 P8 P9 "ARC"
               P11 "LINE" P12 "ARC" P14 "LINE" P15 P16
               P17 P18 P19 "ARC" P21 "LINE" P1 ""
      );command
      (prompt "\nRotation Angle: ")
      (command "ROTATE" "LAST" "" IP pause
      );command
      (setvar "OSMODE" OLDSNAP)
     );while
    );progn
  );if userclick1
);progn
);if dialogshow
(setvar "OSMODE" OLDSNAP)
(setvar "BLIPMODE" OLDBLIP)
(setvar "CMDECHO" OLDECHO)
  (princ)
);defun BEAM
```

Phew, quite a mouthfull hey. Let's walk through it...

First, we define our degrees to radians function.
We then start defining our main function and declaring our variables.
(I have deliberatively left all variables global so that you can check their values once the programme has run.)

After storing then resetting some of our system variables we create a list of names that will appear in our list box.

Following this we load our dialog box, setting up error checking to make sure that the dialog file is loaded before continuing.
If everything is OK we call start_list, add_list and end_list to create our list box. Here the MAPCAR function is useful for turning a *raw* AutoLisp list into a listbox display :

(start_list "selections") - Specify name of the list box. (KEY)
(mapcar 'add_list NAMES) - Specify the list. (NAMES)
(end_list)

The value returned by a list box tile is the INDEX of the selected item.
(i.e. first item = 0 ; second item = 1 ; third item = 2 ; etc.)

The action_tile function is one of the most critical functions to employ when giving your dialog box some functionality.
On it's own a DCL definition does nothing more than define a *lifeless* dialog box. Take a look at the following code :

```
(action_tile
    "cancel"                                    ;if cancel selected
    "(done_dialog) (setq userclick1 nil)"       ;close dialog, set flag
  );action tile
```

Did you notice all the quotes around the lisp code? When you write an action_tile function for a DCL tile, your code is esentially telling the tile :

"Remember this string, then pass it back to me when the user activates you."

The string (i.e Anything within double-quotation marks) is dormant until the user picks the tile that is *remembering* your string. At that time the tile passes the string back to AutoLisp. AutoLisp then converts the string to functioning code and the code is executed.
Lets look at another code fragment.
The following is the action_tile expression assigned to the OK button :

```
  (action_tile
    "accept"                                    ;if OK or double click
    (strcat
      "(progn (setq SIZA (atof (get_tile \"selections\")))" ;get size of beam
      "(done_dialog) (setq userclick1 T))"      ;close dialog and set flag
    );strcat
  );action tile
```

When the user selects the OK button, the lengthy string assigned to the button is passed-back and turned into AutoLisp code that looks like this :

```
    (progn
       (setq SIZA (atof (get_tile "selections")))
         (done_dialog)
         (setq userclick1 T)
    );
```

This code does several things :
It retrieves the value of the INDEX from the list box.
It terminates the dialog box.
It assigns a value of T to the variable userclick1.

Now that everything is set and ready to go, we invoke the dialog box.

(start_dialog)

Once it is on the screen, it controls the programme flow, until the user hits OK, Cancel or double-clicks on an item in the list box.

(unload_dialog dcl_id)

The user has made his selection so now we can unload the dialog box.

For the rest of the program we are back to conventional AutoLisp code.
(This is basically self explanatory, especially if you have studied my earlier tutorials.)

The program retrieves the name of the item selected from the list box by utilising the index (SIZA) of the selected item and the nth function.

(setq SIZA (nth SIZA NAMES)).

It then searches the data file for the corresponding name, retrieves the list of dimensions, formats the list then stores the dimensions into individual variables.

(mapcar 'set '(H B T1 T2 R1) dlist)

The points required to draw the beam are then calculated using the POLAR function and the beam is drawn and rotated if required.
Notice how this section of the coding is contained in a while loop to allow the user to draw more than one beam if he wishes. This saves him from having to re-run the entire routine again and again if he wants to draw multiple, identical beams.

Nesting Dialog Boxes.

**This is another area that seems to cause a lot of confusion for some people.
Let's take a slow walk through the process of nesting and hiding dialog boxes and see if we can't explain it more clearly.**

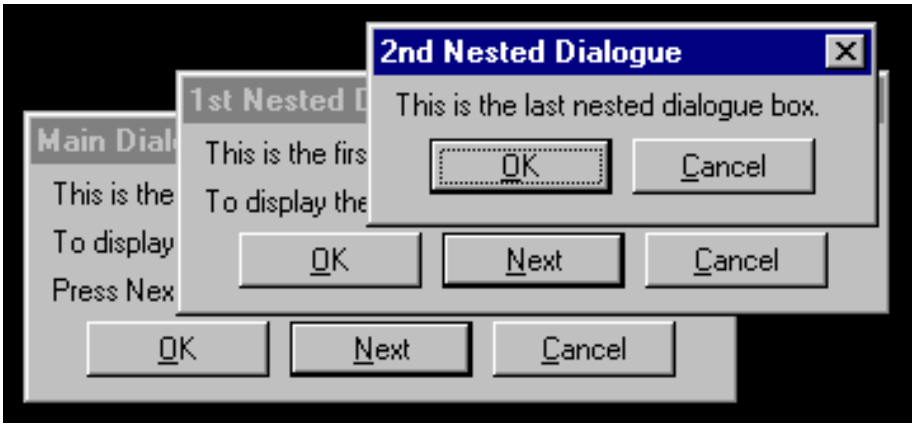
**To nest dialog boxes is pretty straightforward. You simply call the new, nested dialog box from within an action expression or callback function.
For example :**

```
(action_tile "next" "(nextfunction)")
```

This simply says that when the tile with the key of 'next' is selected, then run the function (nextfunction). (nextfunction), of course, would have it's own dialog statements.

**Just a couple of comments on nesting dialog boxes.
Firstly, you cannot use the previous dialog box whilst a nested dialog is active. You must close the nested dialog first.
And secondly, AutoCAD states that there is a limit of no more than eight nested dialog boxes allowed, but recommends a maximum of four.
(Try it, I haven't!!!)
Thirdly, try and keep each nested dialog smaller than the preceding dialog.**

Here's an example of a function with a main dialog box and two nested dialog boxes.



First the DCL Coding:

```
main : dialog {
        label = "Main Dialog";

        : column {

        : text {
        key = "txt1";
        value = "This is the main dialog box.";
               }
```

```
        : text {
        key = "txt2";
        value = "To display the next, nested dialog,";
               }

        : text {
        key = "txt3";
        value = "Press Next....";
               }
        }
        : row {

        : spacer { width = 1; }

        : button {
        label = "OK";
        key = "accept";
        width = 12;
        fixed_width = true;
        mnemonic = "O";
        is_default = true;
        }

        : button {
        label = "Next";
        key = "next";
        width = 12;
        fixed_width = true;
        mnemonic = "N";
        }

        : button {
        label = "Cancel";
        key = "cancel";
        width = 12;
        fixed_width = true;
        mnemonic = "C";
        is_cancel = true;
        }

        : spacer { width = 1;}

        }

                 }

////////////////////////////////////////////////

nest1 : dialog {
```

```
label = "1st Nested Dialog";

: column {

: text {
key = "txt1";
value = "This is the first nested dialog box.";
       }

: text {
key = "txt2";
value = "To display the next, nested dialog, press Next";
       }
        }
: row {

: spacer { width = 1; }

: button {
label = "OK";
key = "accept";
width = 12;
fixed_width = true;
mnemonic = "O";
is_default = true;
}

: button {
label = "Next";
key = "next";
width = 12;
fixed_width = true;
mnemonic = "N";
}

: button {
label = "Cancel";
key = "cancel";
width = 12;
fixed_width = true;
mnemonic = "C";
is_cancel = true;
}

: spacer { width = 1;}

}

         }
```

//

```
nest2 : dialog {
        label = "2nd Nested Dialog";

        : column {

        : text {
        key = "txt1";
        value = "This is the last nested dialog box.";
               }
        }

        : row {

        : spacer { width = 1; }

        : button {
        label = "OK";
        key = "accept";
        width = 12;
        fixed_width = true;
        mnemonic = "O";
        is_default = true;
        }

        : button {
        label = "Cancel";
        key = "cancel";
        width = 12;
        fixed_width = true;
        mnemonic = "C";
        is_cancel = true;
        }

        : spacer { width = 1;}

        }

                }

////////////////////////////////////////////
```

And now the Autolisp Coding:

```
(defun c:nest ()
;define the function

  (setq dcl_id (load_dialog "nest.dcl"))
  ;load the DCL file
```

```
  (if (not (new_dialog "main" dcl_id))
  ;load the dialog box

    (exit)
    ;if not loaded exit

  )

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result nil)"
  )
  ;do this if Cancel button selected

  (action_tile
    "accept"
    "(done_dialog)
     (setq result T)"
  )
  ;do this if OK button selected

  (action_tile
    "next"
    "(nest1)"
  )
  ;if Next button is selected, call
  ;the nested dialog function

  (start_dialog)
  ;start the dialog

  (unload_dialog dcl_id)
  ;unload the dialog

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun nest1 ()
;define the function

  (setq dcl_id1 (load_dialog "nest.dcl"))
  ;load the DCL file

  (if (not (new_dialog "nest1" dcl_id1))
  ;load the nested dialog box
```

```
    (exit)
    ;if not loaded exit

  )

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result1 nil)"
  )
  ;if cancel selected do this

  (action_tile
    "accept"
    "(done_dialog)
     (setq result1 T)"
  )
  ;if OK selected do this

  (action_tile
    "next"
    "(nest2)"
  )
  ;if Next selected call the
  ;second nested dialog function


  (start_dialog)
  ;start the nested dialog box

  (unload_dialog dcl_id1)
  ;unload the nested dialog box

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun nest2 ()
;define the function

  (setq dcl_id2 (load_dialog "nest.dcl"))
  ;load the DCL file

  (if (not (new_dialog "nest2" dcl_id2))
  ;load the second nested dialog box

    (exit)
    ;if not found exit
```

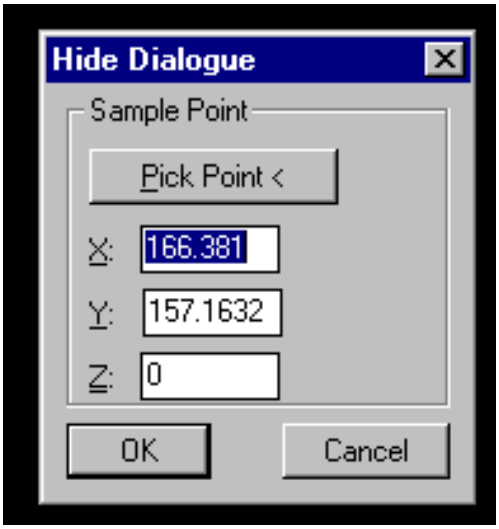
```
  )

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result2 nil)"
  )
  ;do this if Cancel selected

  (action_tile
    "accept"
    "(done_dialog)
     (setq result2 T)"
  )
  ;do this if OK selected

  (start_dialog)
  ;start the second nested dialog box

  (unload_dialog dcl_id2)
  ;unload it

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ);load clean
```

See, pretty simple, hey....
Now, what you have all been waiting for....Hiding Dialog Boxes!!!
(Sorry, but you will have to go to the next page.)
Hurry up, I'm waiting.....

Hiding Dialog Boxes.

Often you need to make a selection on the screen whilst a dialog box is active. But, to do this, you need to be able to hide the dialog box to allow the user to make a selection from the screen. You must then restore the dialog box, along with the values that the user has selected.
When you end a dialog box you use the (done_dialog) function. But did you know that when you call the (done_dialog) function a status argument is returned?
Well it's true!!! Now think about this. When (done_dialog) is called from a tile who's key is 'cancel' (The Cancel button), it returns a status of 0. If a tile who's key is 'accept' is choosen (The OK button), it returns a status of 1. So, if you use the (done_dialog) function alongwith a status argument of say 4, you know that the dialog box has been hidden and not ended or cancelled.
But how do I retrieve that status argument?
Easy, when you call (start_dialog), it returns the (done_dialog) status value. eg. (setq flag (start_dialog).
By testing this value in a loop we can determine whether the dialog was simply hidden or ended or cancelled. Below is a sample routine that should, hopefully, I hope and pray, explain it to you a lot better.

This routine simply displays a dialog box that asks the user for a point.
If the user choosers the pick point button the dialog is hidden to allow the user to pick a point on the screen. The dialog is then re-displayed and the x, y and z edit boxes are updated to the new point value. It also allows the user to
enter the point values directly into the edit boxes if he so wishes.



First the DCL Coding:

```
hidebox : dialog {

        label = "Hide Dialog";

        : boxed_column {
          label = "Sample Point";
```

```
        : button {
        label = "Pick Point <";
        key = "hide";
        width = 12;
        fixed_width = true;
        mnemonic = "P";
        }

        : edit_box {
        key = "eb1";
        label = "&X:";
        width = 8;
        fixed_width = true;
                    }

        : edit_box {
        key = "eb2";
        label = "&Y:";
        width = 8;
        fixed_width = true;
                    }

        : edit_box {
        key = "eb3";
        label = "&Z:";
        width = 8;
        fixed_width = true;
                    }

        }

        ok_cancel;

                  }
```

And now the AutoCAD Coding:

```
(defun c:hidebox ()
;define the function

  (setq ptx "1.000"
        pty "0.000"
        ptz "0.000"
        flag 4
  );setq
  ;set default x,y, and z values
  ;and set flag to 4
```

```
(setq dcl_id (load_dialog "hidebox.dcl"))
;load the DCL file

(while (> flag 2)
;check the flag status and carry on looping
;if it is greater than 2

(if (not (new_dialog "hidebox" dcl_id))
;load the dialog box

  (exit)
  ;if not loaded exit

)

(set_tile "eb1" ptx)
;display x value

(set_tile "eb2" pty)
;display y value

(set_tile "eb3" ptz)
;display z value

(mode_tile "eb1" 2)
;set focus to x edit box

(mode_tile "eb1" 3)
;select contents

(action_tile
  "cancel"
  "(done_dialog)
   (setq result nil)"
)
;if Cancel button selected, close
;the dialog. This action sets the
;flag to 0.

(action_tile
  "accept"
  "(setq ptx (get_tile \"eb1\"))
   (setq pty (get_tile \"eb2\"))
   (setq ptz (get_tile \"eb3\"))
   (done_dialog)
   (setq result T)"
)
;if OK button was selected, get the edit box
;point values, close the dialog. This action
;sets the flag to 1.
```

```
  (action_tile
    "hide"
    "(done_dialog 4)"
  )
  ;if pick button selected, hide the dialog
  ;and set the flag to 4

  (setq flag (start_dialog))
  ;start the dialog and set flag
  ;to value of start dialog

  (if (= flag 4)
  ;if the pick button was selected

    (progn
    ;do the following

      (setq selpoint (getpoint "\nInsertion Point: "))
      ;get the insertion point

      (setq ptx (rtos (car selpoint) 2 4))
      ;get the x value

      (setq pty (rtos (cadr selpoint) 2 4))
      ;get the y value

      (setq ptz (rtos (caddr selpoint) 2 4))
      ;get the z value


    );progn

  );if

  );while

  (unload_dialog dcl_id)
  ;unload the dialog

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ);load clean
```

To hide AND nest dialog boxes is just a case of combining the two.
(I'll leave that up to you to figure out....)

Hiding Dialog Boxes in DCL

Ladies and Gentlemen *"WASSUP"*
Hee, hee - I've always wanted to do that.

Probably the most vexing (on my part) query I get, is in regards to hiding dialog boxes.
It seems to take ages and numerous emails before the poor soul trying to grasp the concept finally understands the principle, or (and this is more likely) gives up and joins a monastery/nunnery.
So, without further due, and absolutely no regard for my own safety, I'm going to have another shot at explaining how to go about hiding such a wee delicate thing as a dialog box.
Okay, are you ready? Are you sat nice and comfy on your cuddly, wuddly little bottom?
The let's do it!!!
("*Randall, could you please keep the noise down 'cos I'm trying to explain something here. What? You're eating!! Try and use a knife and and fork - I know it's chicken and you're allowed to use your fingers, but you're supposed to cook the thing and remove the feathers first.")*

On to the more serious bit.....

Often you need to make a selection on the screen whilst a dialog box is active. But to do this, you need to be able to hide the dialog box to allow the user to make a selection from the screen. You must then restore the dialog box, along with the values that the user has selected. To accomplish this lets first have a look at two of the most critical DCL functions :

(done_dialog [status])

- **status : A positive integer that *(start_dialog)* will return instead of returning 1 for OK or 0 for Cancel. The meaning of any status value greater than 1 is determined by your application.**
- **Usage Notes : An explicit AutoLisp action for the "accept" button must specify a status of 1 (or an application-defined value); otherwise, *(start_dialog)* returns the default value, 0, which makes it appear as if the dialog box was canceled.**

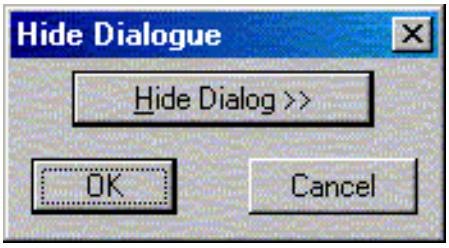
(start_dialog)

- **Return Values : The *(start_dialog)* function returns the optional status passed to *(done_dialog).* The default value is 1 if the user presses OK, 0 if the user presses Cancel, or -1 if all dialog boxes are terminated with *(term_dialog).* If *(done_dialog)* is passed an integer status greater than 1, *(start_dialog)* returns this value, whose meaning is determined by the application.**

What the heck does all this mean????
When you want to close a dialog, you call the *(done_dialog)* function.
But did you know that when you call the *(done_dialog)* function a status argument is returned? Well it's true!!!
Now think about this. When *(done_dialog)* is called from a tile who's key is 'cancel' (the Cancel button), it returns a status of 0.
If a tile who's key is 'accept' is chosen (the OK button), it will return a status value of 1.
Note carefully, YOU must define a status of 1.
Now here's the important bit, if you use the *(done_dialog)* function along with a status value of say 4, you know that the dialog box has been hidden and not ended or cancelled.
But how do I retrieve that status value?
Easy, when you call *(start_dialog)*, it returns the *(done_dialog)* status value.

(setq flag (start_dialogue))

By testing this value in a *(while)* loop we can determine whether the dialog was simply hidden or accepted or cancelled. Below is a sample routine that should, hopefully, I hope and pray, explain it to you a lot better.
This routine simply displays a dialog box with three buttons
If you select "OK" or "Cancel", an alert box is displayed and the dialog is simply closed.
But, if you click the "Select Object" button, the dialog is hidden and an alert box is displayed informing you of the fact. After selecting OK, the dialog is re-displayed or, to use the words of a great magician that I once knew, un-hidden.



First the DCL Coding :

```
hidedialog1 : dialog {
        label = "Hide Dialogue";

        : button {
        label = "Hide Dialog >>";
        key = "hide";
        width = 8;
        fixed_width = true;
        mnemonic = "H";
                alignment = centered;
                is_default = true;
        }

        : spacer { width = 1;}

        ok_cancel;

         }
```

Save this as "HideDialog1.DCL
And now the AutoLisp coding. Save this as "HideDialog1.LSP :

```
(defun c:hidedialog1 ( / dcl_id flag)

  ;set flag to 4
  (setq flag 4)

  ;load the DCL file
  (setq dcl_id (load_dialog "hidedialog1.dcl"))

  ;check the flag status and carry on looping
  ;if it is greater than 2
  (while (> flag 2)

  ;load the dialog box
  (if (not (new_dialog "hidedialog1" dcl_id))

    ;if not loaded exit
    (exit))

  ;Note: I have explicitly defined the status
  ;return value for the "cancel" "accept" and "hide"
  ;tiles. I recommend you do the same.

  ;if Cancel button selected, close
  ;the dialog. This action sets the
  ;flag to 0.
  (action_tile "cancel" "(done_dialog 0)")

  ;if OK button was selected, close
  ; the dialog. This action
  ;sets the flag to 1.
  (action_tile "accept" "(done_dialog 1)")

  ;if pick button selected, hide the dialog
  ;and set the flag to 4
  (action_tile "hide" "(done_dialog 4)")

  ;start the dialog and set flag
  ;to value of start dialog
  (setq flag (start_dialog))

  ;if the OK button was selected
  (if (= flag 1) (alert "You chose the OK button"))

  ;if the Cancel button was selected
  (if (= flag 0) (alert "You chose the Cancel button"))

  ;if the pick button was selected
  (if (= flag 4) (alert "You chose the Hide Dialog button"))

  );while
```

```
  ;unload the dialog
  (unload_dialog dcl_id)

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ);load clean
```

Let's now get really brave and have a look at a wee practical example.....

We will now attempt to design a dialog box that will displays certain properties of a selected entity. We would like the dialog to display first, and then hide itself whilst we go about making up our mind which object we want to select. Once selected, the dialog will be updated with the objects property values. We also want to be able to continue selecting objects which means that the dialog must update after each selection.
Okay, let's give it a go.

First the DCL Coding :

```
hidedialog : dialog {
        label = "Hide Dialogue";

        : list_box {
                label = "&Properties :";
                key = "selections";
                height = 7;
        width = 25;
        }

        : button {
        label = "Select Object >>";
        key = "hide";
        width = 8;
        fixed_width = true;
        mnemonic = "S";
                alignment = centered;
                is_default = true;
        }

        : spacer { width = 1;}

        ok_cancel;

         }
```

Save this as "HideDialog.DCL
And now the AutoLisp coding. Save this as "HideDialog.LSP :

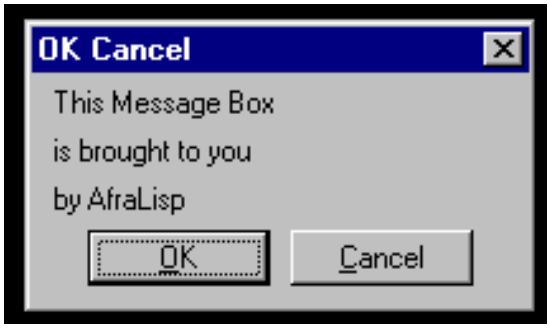
```
(defun c:hidedialog ( / dcl_id thelist lay col ltp lwt flag)

  (vl-load-com)

  ;set up default list box values
  (setq thelist '("Layer = NULL" "Color = NULL"
                  "Linetype = NULL" "Lineweight = NULL"))

  ;set flag to 4
  (setq flag 4)

  ;load the DCL file
  (setq dcl_id (load_dialog "hidedialog.dcl"))

  ;check the flag status and carry on looping
  ;if it is greater than 2
  (while (> flag 2)

  ;load the dialog box
  (if (not (new_dialog "hidedialog" dcl_id))

    ;if not loaded exit
    (exit))

  ;populate the list box with the values
  ;in thelist
  (start_list "selections")
  (mapcar 'add_list thelist)
  (end_list)

  ;if Cancel button selected, close
  ;the dialog. This action sets the
  ;flag to 0.
  (action_tile "cancel" "(done_dialog 0)")

  ;if OK button was selected, close
  ; the dialog. This action
  ;sets the flag to 1.
  (action_tile "accept" "(done_dialog 1)")

  ;if pick button selected, hide the dialog
  ;and set the flag to 4
  (action_tile "hide" "(done_dialog 4)")
```

```
;start the dialog and set flag
;to value of start dialog
(setq flag (start_dialog))

;if the OK button was selected
(if (= flag 1) (alert "You chose the OK button"))

;if the Cancel button was selected
(if (= flag 0) (alert "You chose the Cancel button"))

;if the pick button was selected
(if (= flag 4)

  ;do the following
  (progn

      ;select the object
      (setq ent (entsel))

      ;convert to vl object
      (setq ent (vlax-Ename->Vla-Object (car ent)))

      ;get the properties and place them in a list
      ;the layer first
      (setq lay (strcat "Layer = "
                    (vla-get-layer ent))

      ;then the color
            col (strcat "Color = "
                      (itoa (vla-get-color ent)))

      ;next the linetype
            ltp (strcat "Linetype = "
                      (vla-get-linetype ent))

      ;finally the lineweight
           lwt (strcat "Lineweight = "
                      (itoa (vla-get-lineweight ent)))

      ;create the list
           thelist  (list lay col ltp lwt)

      );setq

  );progn

);if

);while
```

```
  ;unload the dialog
  (unload_dialog dcl_id)

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ);load clean
```

Hey just think! You can now design your own properties dialog.
Well that's about it regarding hiding dialog boxes.
I hope you had a good time, enjoyed the snacks (especially the chicken) and didn't drink too much beer.
Remember, don't drink and drive because you might spill some.

AutoLisp Message Box.



This library of AutoLisp Functions closely emulates the Visual Basic Message Box Function. It will display 5 different types of message boxes and allows you to enter 3 text message lines and customise the Message Box title.
The message boxes available are :

- Ok Only
- Ok Cancel
- Rentry Cancel
- Yes No
- Yes No Cancel

The syntax for usage is as follows:

```
(function name "message1" "message2" "message3" "title")
```

For example:

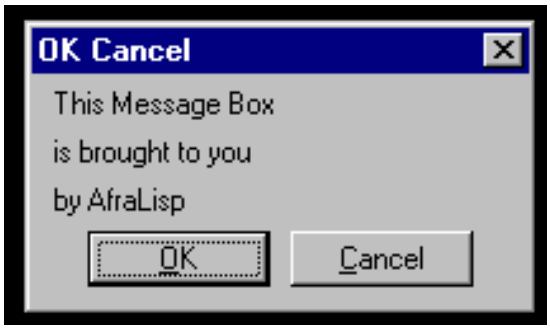
```
(lspokcancel "This is the first line" "This is the second line"
             "This is the third line" "This is the title")
```

Here is a detailed explanation of each message box:

Ok Only.

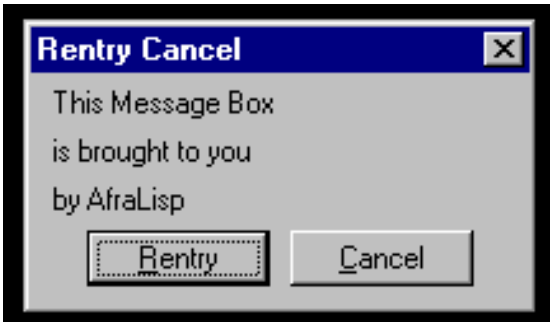
Syntax : (lspOkOnly "message1" "message2" "message3" "title")
This message box returns nothing as it is used to convey a message only

Ok Cancel.



Syntax : (lspOkCancel "message1" "message2" "message3" "title")
This message box returns True if OK selected and nil if Cancel selected

Rentry Cancel.

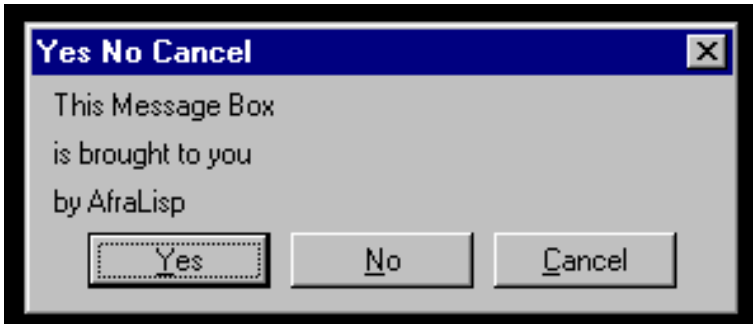


Syntax : (lspRentryCancel "message1" "message2" "message3" "title")
This message box returns True if Rentry selected and nil if Cancel selected

Yes No.

Syntax : (lspYesNo "message1" "message2" "message3" "title")
This message box returns True if Yes selected and "F" if No selected

Yes No Cancel.



Syntax : (lspYesNoCancel "message1" "message2" "message3" "title")
This message box returns True if Yes selected, "F" if No selected and nil if Cancel selected.

Source Coding.

Here is the DCL Source Coding:

```
lspOkCancel : dialog {

        key = "main";

        : column {

        : text {
        key = "message1";
               }

        : text {
        key = "message2";
               }

        : text {
        key = "message3";
               }
        }
        : row {

        : spacer { width = 1; }

        : button {
        label = "OK";
        key = "accept";
        width = 12;
        fixed_width = true;
```

```
        mnemonic = "O";
        is_default = true;
        }

        : button {
        label = "Cancel";
        key = "cancel";
        width = 12;
        fixed_width = true;
        mnemonic = "C";
        is_cancel = true;
        }

        : spacer { width = 1;}

        }

                }

////////////////////////////////////////////////

lspYesNo : dialog {

        key = "main";

        : column {

        : text {
        key = "message1";
               }

        : text {
        key = "message2";
               }

        : text {
        key = "message3";
               }
        }
        : row {

        : spacer { width = 1; }

        : button {
        label = "Yes";
        key = "yes";
        width = 12;
        fixed_width = true;
        mnemonic = "Y";
        is_default = true;
        }
```

```
        : button {
        label = "No";
        key = "no";
        width = 12;
        fixed_width = true;
        mnemonic = "N";
        is_cancel = true;
        }

        : spacer { width = 1;}

        }

                }

////////////////////////////////////////////

lspOkOnly : dialog {

        key = "main";

        : column {

        : text {
        key = "message1";
               }

        : text {
        key = "message2";
               }

        : text {
        key = "message3";
               }
        }
        : row {

        : spacer { width = 1; }

        : button {
        label = "OK";
        key = "accept";
        width = 12;
        fixed_width = true;
        mnemonic = "O";
        is_default = true;
        alignment = centered;
        }

        : spacer { width = 1;}
```

```
        }

                }

////////////////////////////////////////////////

lspYesNoCancel : dialog {

        key = "main";

        : column {

        : text {
        key = "message1";
               }

        : text {
        key = "message2";
               }

        : text {
        key = "message3";
               }
        }
        : row {

        : spacer { width = 1; }

        : button {
        label = "Yes";
        key = "yes";
        width = 12;
        fixed_width = true;
        mnemonic = "Y";
        is_default = true;
        }

        : button {
        label = "No";
        key = "no";
        width = 12;
        fixed_width = true;
        mnemonic = "N";
        }

        : button {
        label = "Cancel";
        key = "cancel";
        width = 12;
        fixed_width = true;
```

```
        mnemonic = "C";
        is_cancel = true;
        }

        : spacer { width = 1;}

        }

                }

////////////////////////////////////////////

lspRentryCancel : dialog {

        key = "main";

        : column {

        : text {
        key = "message1";
               }

        : text {
        key = "message2";
               }

        : text {
        key = "message3";
               }
        }
        : row {

        : spacer { width = 1; }

        : button {
        label = "Rentry";
        key = "rentry";
        width = 12;
        fixed_width = true;
        mnemonic = "R";
        is_default = true;
        }

        : button {
        label = "Cancel";
        key = "Cancel";
        width = 12;
        fixed_width = true;
        mnemonic = "C";
        is_cancel = true;
        }
```

```
        : spacer { width = 1;}
        }
                 }
////////////////////////////////////////////
```

And the Autolisp Code:

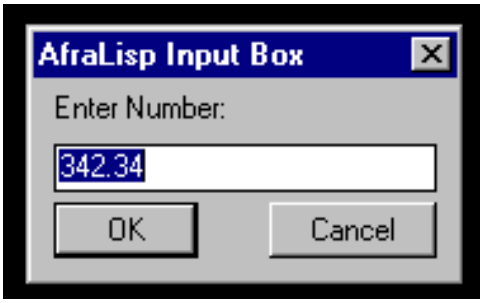
```
(defun lspOkCancel (message1 message2 message3 main)

  (setq dcl_id (load_dialog "msgbox.dcl"))
  (if (not (new_dialog "lspOkCancel" dcl_id))
    (exit)
  )

  (set_tile "message1" message1)
  (set_tile "message2" message2)
  (set_tile "message3" message3)
  (set_tile "main" main)

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result nil)"
  )
  (action_tile
    "accept"
    "(done_dialog)
     (setq result T)"
  )
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun lspYesNo (message1 message2 message3 main)

  (setq dcl_id (load_dialog "msgbox.dcl"))
  (if (not (new_dialog "lspYesNo" dcl_id))
    (exit)
  )

  (set_tile "message1" message1)
  (set_tile "message2" message2)
```

```
  (set_tile "message3" message3)
  (set_tile "main" main)

  (action_tile
    "no"
    "(done_dialog)
     (setq result \"F\")"
  )
  (action_tile
    "yes"
    "(done_dialog)
     (setq result T)"
  )
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun lspOkOnly (message1 message2 message3 main)

  (setq dcl_id (load_dialog "msgbox.dcl"))
  (if (not (new_dialog "lspOkOnly" dcl_id))
    (exit)
  )

  (set_tile "message1" message1)
  (set_tile "message2" message2)
  (set_tile "message3" message3)
  (set_tile "main" main)


  (action_tile
    "yes"
    "(done_dialog)"
  )
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun lspYesNoCancel (message1 message2 message3 main)

  (setq dcl_id (load_dialog "msgbox.dcl"))
  (if (not (new_dialog "lspYesNoCancel" dcl_id))
    (exit)
  )
```

```
  (set_tile "message1" message1)
  (set_tile "message2" message2)
  (set_tile "message3" message3)
  (set_tile "main" main)

  (action_tile
    "no"
    "(done_dialog)
     (setq result \"F\")"
  )
  (action_tile
    "yes"
    "(done_dialog)
     (setq result T)"
  )
  (action_tile
    "cancel"
    "(done_dialog)
     (setq result nil)"
  )
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun lspRentryCancel (message1 message2 message3 main)

  (setq dcl_id (load_dialog "msgbox.dcl"))
  (if (not (new_dialog "lspRentryCancel" dcl_id))
    (exit)
  )

  (set_tile "message1" message1)
  (set_tile "message2" message2)
  (set_tile "message3" message3)
  (set_tile "main" main)

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result nil)"
  )
  (action_tile
    "rentry"
    "(done_dialog)
     (setq result T)"
  )
```

```
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ)
```

AutoLisp Input Box.



This routine is very similar to the Visual Basic Input Box Function.
To use this routine you feed the function 3 arguments, the Input Box Prompt, the Title of the Input Box, and the Default Value of the Edit Box.

The syntax is straightforward:

```
(inputbox "prompt" "title" "default")
```

For example, the Input Box above was called by using the following command:

```
(inputbox "Enter Number" "AfraLisp Inputbox" "342.34")
```

The Input Box returns a variable "inputvalue" containing the value of the Edit Box.

Source Coding.

First the DCL Coding:

```
inputbox : dialog {

        key = "title";

        : text {
        key = "prompt";
               }
        : edit_box {
        key = "eb1";
                   }
        ok_cancel;

                  }
```

And next the AutoLisp Coding:

```
(defun inputbox (prompt title default)

  (setq dcl_id (load_dialog "inputbox.dcl"))
  (if (not (new_dialog "inputbox" dcl_id))
    (exit)
  )

  (set_tile "prompt" prompt)
  (set_tile "title" title)
  (set_tile "eb1" default)
  (mode_tile "eb1" 2)

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result nil)"
  )
  (action_tile
    "accept"
    "(setq inputvalue (get_tile \"eb1\"))
     (done_dialog)
     (setq result T)"
  )
  (start_dialog)
  (unload_dialog dcl_id)
  (princ)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ)
```

Referencing DCL Files.

If you look in your AutoCAD Support directory, you will find a file entitled Base.Dcl. This file contains the DCL definitions for the basic, predefined tiles and tile types. It also contains definitions for commonly used prototypes.
You will also find Acad.Dcl in this directory. This file contains the standard definitions of all dialog boxes used by AutoCAD.
But, did you know that a DCL file can also use tiles defined in another DCL file by naming the other file in what is called an *include directory.*
Let me try and show you how this works.

First create a new DCL file called Include1.Dcl:

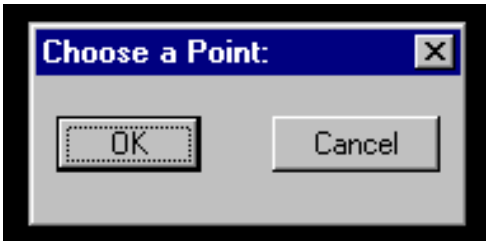
```
include1 : dialog {

        label = "Choose a Point:";

        ok_cancel;

                }
```

Now, create a new AutoLisp file entitled Include1.Lsp with the following coding:

```
(defun c:include1 ()
;define the function

  (setq dcl_id (load_dialog "include1.dcl"))
  ;load the DCL file

  (if (not (new_dialog "include1" dcl_id))
  ;load the dialog box

    (exit)
    ;if not loaded exit

  )

  (action_tile
    "cancel"
    "(done_dialog)
     (setq result nil)"
  )
  ;if Cancel button selected, close
  ;the dialog.
```

```
  (action_tile
    "accept"

    "(done_dialog)
     (setq result T)"
  )
  ;if OK button was selected

  (start_dialog)
  ;start the dialog

  (unload_dialog dcl_id)
  ;unload the dialog

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ);load clean
```

Open AutoCAD and load and run the routine.
You should have a very simple dialog box looking like this :



Now, let's say we had a company logo, or something like that, that we wanted to display on all our custom dialog boxes. We could put the DCL coding for the logo in all our DCL files. But what happens, if say our telephone number changes. We would then have to edit all of our dialog boxes containing the DCL coding for our logo. Here's a better way. We will create another DCL file containing the DCL coding for our logo, and *include* this file in our original DCL file. By doing it this way, we only have to change the master DCL file if we want to make revisions to our logo.

Let's create our logo DCL file first.
Create a new DCL file entitled Atitle.Dcl:

```
atitle : boxed_column {

        : paragraph {
```

```
                : text_part {
                  label = "Brought To";
                }

                : text_part {
                  label = " You By";
                }

                : text_part {

                  label = "AfraLisp";
                }

                : text_part {

                  label = "063-235837";
                }
        }

        }
```

Now revise the coding of our original include1.dcl to look like this:

```
@include "atitle.dcl"

include1 : dialog {

        label = "Choose a Point:";

        ok_cancel;

        atitle;

                 }
```

Re-Load and run Include1.Lsp. It should look like this:

O.K I agree, it's not the best looking logo I've ever seen, but I'm sure you get the idea.

You can even include two or more references to DCL Files if you wish.
Let's have a look at another example. First create a new DCL file called gpoint.dcl. Add this coding to it:

```
gpoint  : boxed_column {
          label = "Sample Point";

        : button {
        label = "Pick Point <";
        key = "hide";
        width = 12;
        fixed_width = true;
        mnemonic = "P";
        }

        : edit_box {
        key = "eb1";
        label = "&X:";
        width = 8;
        fixed_width = true;
                   }

        : edit_box {
        key = "eb2";
        label = "&Y:";
        width = 8;
        fixed_width = true;
                   }

        : edit_box {
        key = "eb3";
        label = "&Z:";
        width = 8;
        fixed_width = true;
                   }

        }
```

Now create a new DCL file entitle Include.Dcl and add this coding:

```
@include "gpoint.dcl"
@include "atitle.dcl"


include : dialog {
```

```
        label = "Choose a Point:";

        gpoint;

        ok_cancel;

        atitle;

                }
```

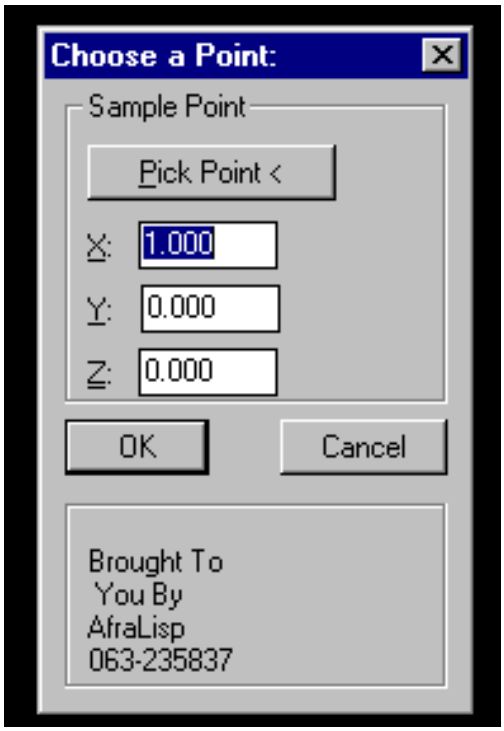
The coding for your lisp file, called Include.Lsp will look like this:

```
(defun c:include ()
;define the function

  (setq ptx "1.000"
        pty "0.000"
        ptz "0.000"
        flag 4
  );setq
  ;set default x,y, and z values
  ;and set flag to 4

  (setq dcl_id (load_dialog "include.dcl"))
  ;load the DCL file

  (while (> flag 2)
  ;check the flag status and carry on looping
  ;if it is greater than 2

  (if (not (new_dialog "include" dcl_id))
  ;load the dialog box

    (exit)
    ;if not loaded exit

  )

  (set_tile "eb1" ptx)
  ;display x value

  (set_tile "eb2" pty)
  ;display y value

  (set_tile "eb3" ptz)
  ;display z value

  (mode_tile "eb1" 2)
```

```
;set focus to x edit box

(mode_tile "eb1" 3)
;select contents

(action_tile
  "cancel"
  "(done_dialog)
   (setq result nil)"
)
;if Cancel button selected, close
;the dialog. This action sets the
;flag to 0.

(action_tile
  "accept"
  "(setq ptx (get_tile \"eb1\"))
   (setq pty (get_tile \"eb2\"))
   (setq ptz (get_tile \"eb3\"))
   (done_dialog)
   (setq result T)"
)
;if OK button was selected, get the edit box
;point values, close the dialog. This action
;sets the flag to 1.

(action_tile
  "hide"
  "(done_dialog 4)"
)
;if pick button selected, hide the dialog
;and set the flag to 4

(setq flag (start_dialog))
;start the dialog and set flag
;to value of start dialog

(if (= flag 4)
;if the pick button was selected

  (progn
  ;do the following

    (setq selpoint (getpoint "\nInsertion Point: "))
    ;get the insertion point

    (setq ptx (rtos (car selpoint) 2 4))
    ;get the x value

    (setq pty (rtos (cadr selpoint) 2 4))
```

```
      ;get the y value

      (setq ptz (rtos (caddr selpoint) 2 4))
      ;get the z value


    );progn

  );if

  );while

  (unload_dialog dcl_id)
  ;unload the dialog

  (princ)

);defun

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(princ);load clean
```

Your dialog box should look like this:



If you wanted, you could create a whole library of DCL files containing customised dialog definitions which you could include/reference in all of your DCL files.

Functional Synopsis of DCL Tiles.

Many people have requested a listing of all of the available DCL tiles along with a working AutoLisp example of each tile.
In this tutorial we will have a look at each tile, the DCL coding for it, as well as the AutoLisp coding required to make the dialogue box tile functional.

AutoCAD and DCL

The AutoCAD PDB facility has a large set of built-in, or predefined, tiles that can be used by themselves or as the basis for more complex tiles.
These tiles can be sub-divided into 5 main categories:

Exit Buttons and Error Tiles.

- ok_only
- ok_cancel
- ok_cancel_help
- ok_cancel_help_errtile
- ok_cancel_help_info
- errtile

Predefined Active Tiles.

- button
- edit_box
- list_box
- popup_list
- radio_button
- toggle
- slider
- image_button

Decorative and Informative Tiles.

- image
- text
- spacer
- spacer_0
- spacer_1

Text Clusters.

- concatenation
- paragraph
- text_part

Tile Clusters.

- boxed_column
- boxed_radio_column
- boxed_radio_row
- boxed_row
- column
- dialog
- radio_column
- radio_row
- row

---

ok_only.

Syntax:

```
ok_only;
```

The ok_only tile is defined in the Base.Dcl file.
The key of the OK button is 'accept'.

DCL Coding:

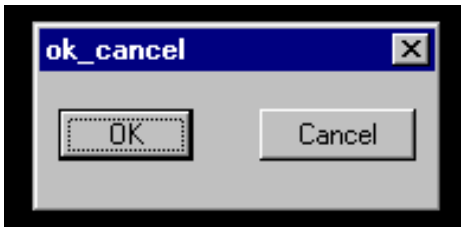
```
lisp48a : dialog {                          //dialog name
          label = "ok_only" ;               //give it a label

        ok_only ;                           //predefined OK button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48a ()
;define function

  (setq dcl_id (load_dialog "lisp48a.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48a" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
```

```
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

    );action tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

    (alert "You Choose O.K.")
    ;inform the user

  );if

 (princ)

);defun

(princ)
```

ok_cancel.



Syntax:

```
ok_cancel;
```

The ok_cancel tile is defined in the Base.Dcl file.
The key of the OK button is 'accept'.
The key of the Cancel button is 'cancel'.

DCL Coding:

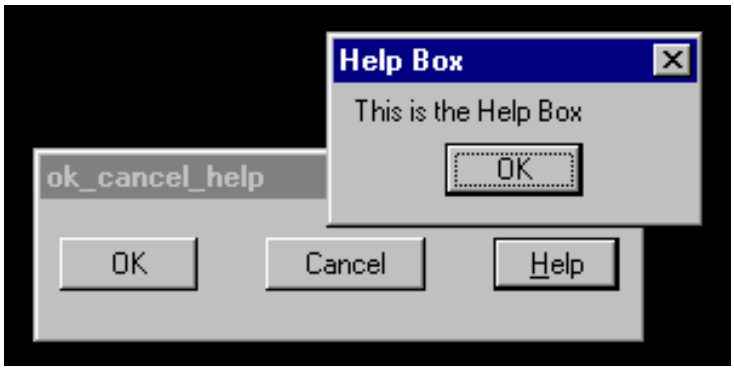
```
lisp48b : dialog {                          //dialog name
          label = "ok_cancel" ;             //give it a label

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48b ()
;define function

  (setq dcl_id (load_dialog "lisp48b.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48b" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed
```

```
      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

ok_cancel_help.



Syntax:

```
ok_cancel_help;
```

The ok_cancel_help tile is defined in the Base.Dcl file.
The key of the OK button is 'accept'.
The key of the Cancel button is 'cancel'.
The key of the Help button is 'help'.

DCL Coding:

```
lisp48c : dialog {                          //dialog name
          label = "ok_cancel_help" ;        //give it a label

        ok_cancel_help;                     //predefined OK/Cancel/Help
                                            //button

        }                                   //end dialog

gotohelp : dialog {                         //dialog name
           label = "Help Box" ;             //give it a label

         : text {                           //define text
           label = "This is the Help Box" ; //give it a label
         }                                  //end text

        ok_only ;                           //predefined OK Button

        }                                   //end dialog
```

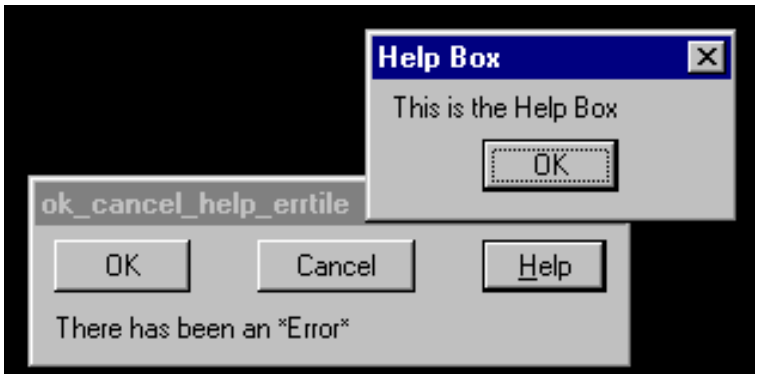
AutoLisp Coding:

```
(defun C:lisp48c ()
;define function
```

```
(setq dcl_id (load_dialog "lisp48c.dcl"))
;load dialog

(if (not (new_dialog "lisp48c" dcl_id)
;test for dialog

    );not

  (exit)
  ;exit if no dialog

);if

(action_tile
    "accept"
    ;if O.K. pressed

    "(done_dialog) (setq userclick T)"
    ;close dialog, set flag

 );action tile

  (action_tile
  "cancel"
  ;if cancel button pressed

  "(done_dialog) (setq userclick nil)"
  ;close dialog, lower flag

  );action_tile

  (action_tile
     "help"
  ;if help button pressed

     "(gotohelp)"
     ;call help function

  );action_tile


(start_dialog)
;start dialog

(unload_dialog dcl_id)
;unload

(if userclick
;if OK selected

   (alert "You selected OK")
   ;inform the user

   (alert "You selected Cancel")
```

```
    ;inform the user

  );if

 (princ)

);defun


(defun gotohelp ()
;define function

  (setq dcl_id (load_dialog "lisp48c.dcl"))
  ;load dialog

  (if (not (new_dialog "gotohelp" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog)"
      ;close dialog

   );action tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

 (princ)

);defun


(princ)
```

ok_cancel_help_errtile.



Syntax:

```
ok_cancel_help_errtile;
```

The ok_cancel_help_errtile tile is defined in the Base.Dcl file.
The key of the OK button is 'accept'.
The key of the Cancel button is 'cancel'.
The key of the Help button is 'help'.
The key of the Error tile is 'error'.

DCL Coding:

```
lisp48d : dialog {                              //dialog name
          label = "ok_cancel_help_errtile" ;    //give it a label

        ok_cancel_help_errtile;                 //predefined OK/Cancel/Help/
                                                //errtile button

        }                                       //end dialog

gotohelp : dialog {                             //dialog name
           label = "Help Box" ;                 //give it a label

         : text {                               //defin text
           label = "This is the Help Box" ;     //give it a label
         }                                      //end text

        ok_only ;                               //predefined OK Button

        }                                       //end dialog
```

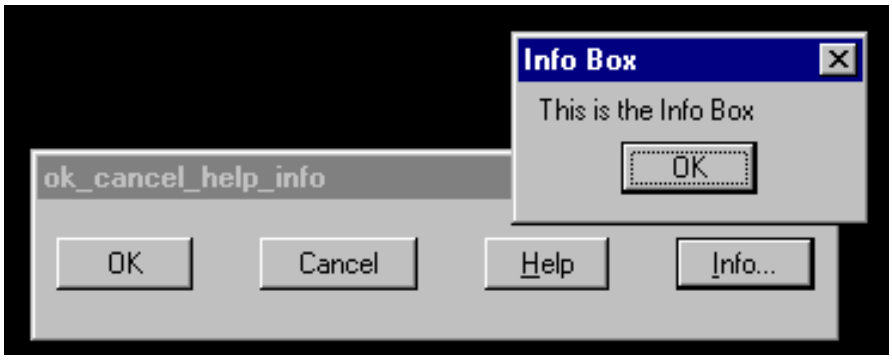
AutoLisp Coding:

```
(defun C:lisp48d ()
```

```
;define function

  (setq dcl_id (load_dialog "lisp48d.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48d" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

    (action_tile
       "help"
    ;if help button pressed

       "(gotohelp)"
       ;call help function

    );action_tile

    (set_tile
     "error"
    ;if there is an error

     "There has been an *Error*"
     ;display this message

    );set_tile


  (start_dialog)
  ;start dialog
```

```
  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun


(defun gotohelp ()
;define function

  (setq dcl_id (load_dialog "lisp48d.dcl"))
  ;load dialog

  (if (not (new_dialog "gotohelp" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog)"
      ;close dialog

   );action tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

 (princ)

);defun


(princ)
```

ok_cancel_help_info.



Syntax:

```
ok_cancel_help_info;
```

The ok_cancel_help_info tile is defined in the Base.Dcl file.
The key of the OK button is 'accept'.
The key of the Cancel button is 'cancel'.
The key of the Help button is 'help'.
The key of the Info button is 'info'.

DCL Coding:

```
lisp48e : dialog {                              //dialog name
          label = "ok_cancel_help_info" ;       //give it a label

        ok_cancel_help_info;                    //predefined OK/Cancel/Help/
                                                //Info button

        }                                       //end dialog

gotohelp : dialog {                             //dialog name
           label = "Help Box" ;                 //give it a label

         : text {                               //defin text
           label = "This is the Help Box" ;     //give it a label
         }                                      //end text

        ok_only ;                               //predefined OK Button

        }                                       //end dialog

gotoinfo : dialog {                             //dialog name
           label = "Info Box" ;                 //give it a label

         : text {                               //define text
           label = "This is the Info Box" ;     //give it a label
         }                                      //end text
```

```
      ok_only ;                                  //predefined OK Button

      }                                          //end dialog
```

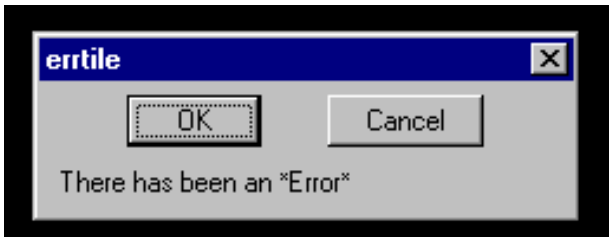
AutoLisp Coding:

```
(defun C:lisp48e ()
;define function

  (setq dcl_id (load_dialog "lisp48e.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48e" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

    (action_tile
       "help"
    ;if help button pressed

       "(gotohelp)"
       ;call help function

    );action_tile

    (action_tile
       "info"
    ;if Info button pressed
```

```
      "(gotoinfo)"
      ;call info function

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun


(defun gotohelp ()
;define function

  (setq dcl_id (load_dialog "lisp48e.dcl"))
  ;load dialog

  (if (not (new_dialog "gotohelp" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog)"
      ;close dialog

   );action tile

  (start_dialog)
  ;start dialog
```

```
  (unload_dialog dcl_id)
  ;unload

 (princ)

);defun

(defun gotoinfo ()
;define function

  (setq dcl_id (load_dialog "lisp48e.dcl"))
  ;load dialog

  (if (not (new_dialog "gotoinfo" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog)"
      ;close dialog

   );action tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

 (princ)

);defun

(princ)
```

errtile.



Syntax:

```
errtile;
```

The errtile tile is defined in the Base.Dcl file.
The key of the errtile is 'error'.

DCL Coding:

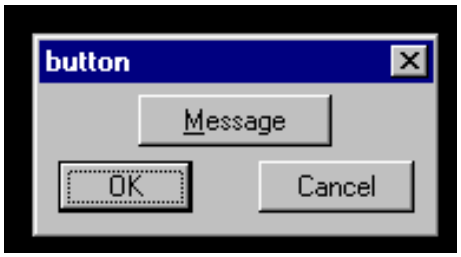
```
lisp48f : dialog {                          //dialog name
          label = "errtile" ;               //give it a label

        ok_cancel ;                         //predefined OK/Cancel button

        errtile;                            //predefined Error tile

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48f ()
;define function

  (setq dcl_id (load_dialog "lisp48f.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48f" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed
```

```
      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile


    ;the following would be called
    ;from your error handler
    (set_tile
     "error"
    ;if there is an error

     "There has been an *Error*"
     ;display this message

    );set_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

button.



Syntax:

```
: button {
  action alignment fixed_height fixed_width
  height is_cancel is_default is_enabled
  is_tab_stop key label mnemonic width
}
```

DCL Coding:

```
lisp48g : dialog {                          //dialog name
          label = "button" ;                //give it a label

        : button {                          //add a button
          label = "Message";                //give it a label
          key = "btn1";                     //give it a name
          mnemonic = "M";                   //give it a mnemonic
          fixed_width = true;               //fix the width
          alignment = centered;             //center the button
        }                                   //end button

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48g ()
;define function

  (setq dcl_id (load_dialog "lisp48g.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48g" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog
```

```
  );if

  (action_tile
      "btn1"
      ;if the message button is pressed

      "(setq col (acad_colordlg 2))"
      ;display the AutoCAD colour dialogue

  );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

   (progn
   ;do the following

     (setq col (itoa col))
     ;convert to string

     (alert (strcat "You selected Colour No: " col))
     ;inform the user

   );progn

  );if

 (princ)

);defun
```

(princ)

edit_box.

Syntax:

```
: edit_box {
  action alignment allow_accept edit_limit
  edit_width fixed_height fixed_width
  height is_enabled is_tab_stop key label mnemonic
  value width password_char
}
```

DCL Coding:

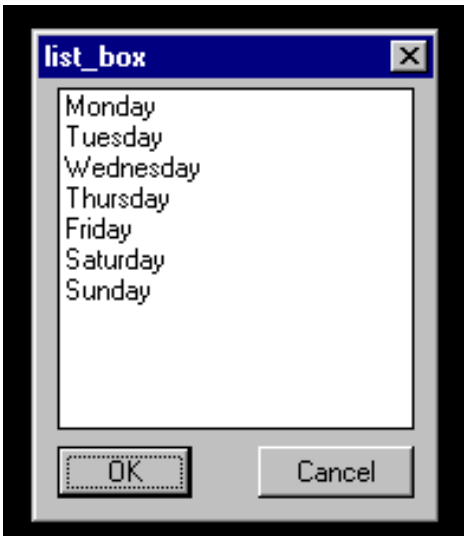
```
lisp48h : dialog {                          //dialog name
          label = "edit_box" ;              //give it a label

        : edit_box {                        //define edit box
          key = "eb1";                      //give it a name
          edit_limit = 8;                   //restrict to 8 letters
          label = "Enter Password";         //give it a label
          mnemonic = "P";                   //define mnemonic
          password_char = "*";              //define password character
        }                                   //end edit box

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48h ()
;define function

  (setq dcl_id (load_dialog "lisp48h.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48h" dcl_id)
  ;test for dialog

      );not

    (exit)
```

```
    ;exit if no dialog

  );if

  (action_tile
      "eb1"
      ;if a password is entered

      "(setq pass $value)"
      ;store the password in the variable 'pass'

  );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

  (alert (strcat "Password is: " pass))
  ;display the password

  );if

 (princ)

);defun

(princ)
```

list_box.



Syntax:

```
: list_box {
  action alignment allow_accept fixed_height
  fixed_width height is_enabled is_tab_stop
  key label list mnemonic multiple_select
  tabs value width
}
```

DCL Coding:

```
lisp48i : dialog {                          //dialog name
          label = "list_box" ;              //give it a label

        : list_box {                        //define list box
        key = "selections";                 //give it a name
        }                                   //end list

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

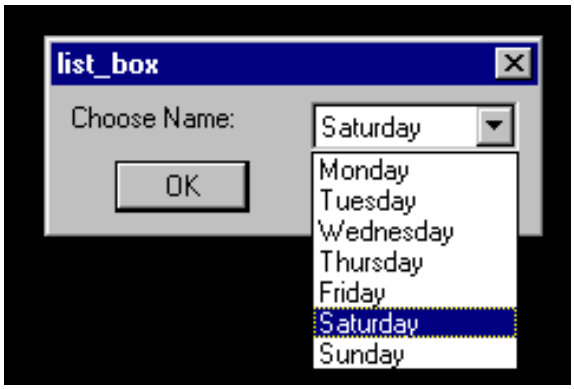
AutoLisp Coding:

```
(defun C:lisp48i ()
;define function

  (setq NAMES '("Monday" "Tuesday" "Wednesday" "Thursday"
                "Friday" "Saturday" "Sunday")
  );setq
  ;define list
```

```
(setq dcl_id (load_dialog "lisp48i.dcl"))
;load dialog

(if (not (new_dialog "lisp48i" dcl_id)
;test for dialog

    );not

  (exit)
  ;exit if no dialog

);if

(start_list "selections")
;start the list box

(mapcar 'add_list NAMES)
;fill the list box

(end_list)
;end list

(action_tile
  "accept"
  ;if O.K. pressed

  (strcat
  ;string 'em together
    "(progn

    (setq SIZ (atof (get_tile \"selections\")))"
    ;get list selection

    "(done_dialog) (setq userclick T))"
    ;close dialog

  );strcat

);action tile

  (action_tile
  "cancel"
  ;if cancel button pressed

  "(done_dialog) (setq userclick nil)"
  ;close dialog

  );action_tile

(start_dialog)
;start dialog

(unload_dialog dcl_id)
;unload

 (if userclick
```

```
   ;check O.K. was selected

    (progn
    ;if it was do the following

      (setq SIZ (fix SIZ))
      ;convert to integer

      (setq SIZ (nth SIZ NAMES))
      ;get the Day

      (alert (strcat "You Selected: " SIZ))
      ;display the Day

    );progn

  );if userclick


 (princ)

);defun

(princ)
```

popup_list.



Syntax:

```
: popup_list {
  action alignment edit_width fixed_height
  fixed_width height is_enabled is_tab_stop
  key label list mnemonic tabs value width
}
```

DCL Coding:

```
lisp48j : dialog {                          //dialog name
          label = "popup_list" ;            //give it a label

        : popup_list {                      //define popup list
        label = "Choose Name:";             //give it a label
        key = "selections";                 //give it a name
        value = "5" ;                       //initial value
        edit_width = 12;                    //fix the width
        }                                   //end list

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48j ()
;define function

  (setq NAMES '("Monday" "Tuesday" "Wednesday" "Thursday"
                "Friday" "Saturday" "Sunday")
  );setq
  ;define list

  (setq dcl_id (load_dialog "lisp48j.dcl"))
  ;load dialog
```

```
(if (not (new_dialog "lisp48j" dcl_id)
;test for dialog

    );not

  (exit)
  ;exit if no dialog

);if

(start_list "selections")
;start the list box

(mapcar 'add_list NAMES)
;fill the list box

(end_list)
;end list

(action_tile
  "accept"
  ;if O.K. pressed

  (strcat
  ;string 'em together
    "(progn

    (setq SIZ (atof (get_tile \"selections\")))"
    ;get list selection

    "(done_dialog) (setq userclick T))"
    ;close dialog

  );strcat

);action tile

  (action_tile
  "cancel"
  ;if cancel button pressed

  "(done_dialog) (setq userclick nil)"
  ;close dialog

  );action_tile

(start_dialog)
;start dialog

(unload_dialog dcl_id)
;unload

 (if userclick
 ;check O.K. was selected
```

```
    (progn
    ;if it was do the following

      (setq SIZ (fix SIZ))
      ;convert to integer

      (setq SIZ (nth SIZ NAMES))
      ;get the Day

      (alert (strcat "You Selected: " SIZ))
      ;display the Day

    );progn

  );if userclick


 (princ)

);defun

(princ)
```

radio_button.

Syntax:

```
: radio_button {
  action alignment fixed_height fixed_width
  height is_enabled is_tab_stop key label
  mnemonic value width
}
```

DCL Coding:

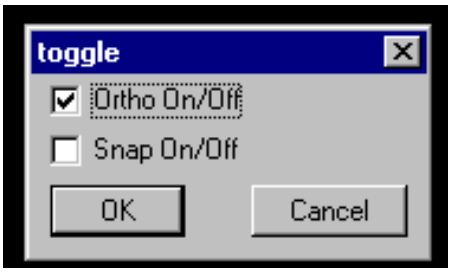
```
lisp48k : dialog {                          //dialog name
          label = "radio_button" ;          //give it a label

        : radio_button {                    //*define radio button
          key = "rb1" ;                     //*give it a name
          label = "Bolt Holes &Site" ;      //*give it a label
          value = "1" ;                     //*switch it on
        }                                   //*end definition

        : radio_button {                    //*define radio button
          key = "rb2" ;                     //*give it a name
          label = "Bolt Holes Sho&p" ;      //*give it a label
        }                                   //*end definition

        : radio_button {                    //*define radio button
          key = "rb3" ;                     //*give it a name
          label = "Bolt Holes &Hidden" ;    //*give it a label
          }                                 //*end definition

        : radio_button {                    //*define radio button
          key = "rb4" ;                     //*give it a name
          label = "Bolt Holes &Ctsnk" ;     //*give it a label
        }                                   //*end definition

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48k ()
;define function

  (setq hole "Site")
  ;store default hole type

  (setq dcl_id (load_dialog "lisp48k.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48k" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile "rb1" "(setq hole \"Site\")")
  ;*store hole type

  (action_tile "rb2" "(setq hole \"Shop\")")
  ;*store hole type

  (action_tile "rb3" "(setq hole \"Hidden\")")
  ;*store hole type

  (action_tile "rb4" "(setq hole \"Countersunk\")")
  ;*store hole type

  (action_tile
    "accept"
    ;if O.K. pressed

    "(done_dialog) (setq userclick T)"
    ;close dialog

  );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog

    );action_tile

  (start_dialog)
  ;start dialog
```

```
  (unload_dialog dcl_id)
  ;unload

   (if userclick
   ;check O.K. was selected

      (alert (strcat "You Selected: " hole))
      ;display the Hole Type

  );if userclick

 (princ)

);defun

(princ)
```

toggle.



Syntax:

```
: toggle {
  action alignment fixed_height fixed_width
  height is_enabled is_tab_stop key label
  mnemonic value width
}
```

DCL Coding:

```
lisp48l : dialog {                          //dialog name
          label = "toggle" ;                //give it a label

        :toggle {                           //define toggle
        key = "tog1";                       //give it a name
        label = "Ortho On/Off";             //give it a label
        }                                   //end toggle

        :toggle {                           //define toggle
        key = "tog2";                       //give it a name
        label = "Snap On/Off";              //give it a label
        }                                   //end definition

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

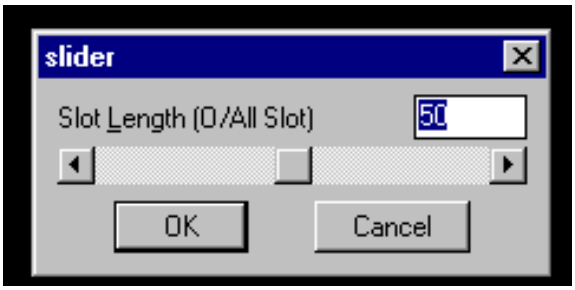
AutoLisp Coding:

```
(defun C:lisp48l ()
;define function

  (setq dcl_id (load_dialog "lisp48l.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48l" dcl_id)
  ;test for dialog

      );not
```

```
  (exit)
  ;exit if no dialog

);if

(setq orth (itoa (getvar "orthomode")))
;get orthomode value

(set_tile "tog1" orth)
;switch toggle on or off

(setq sna (itoa (getvar "snapmode")))
;get snap value

(set_tile "tog2" sna)
;switch toggle on or off

(action_tile "tog1" "(setq orth $value)")
;get ortho toggle value

(action_tile "tog2" "(setq sna $value)")
;get snap toggle value

(action_tile
  "accept"
  ;if O.K. pressed

  (strcat
  ;string 'em together

    "(progn

     (setvar \"orthomode\" (atoi orth))"
    ;ortho on/off

    "(setvar \"snapmode\" (atoi sna))"
    ;snap on/off

    "(done_dialog)(setq userclick T))"
    ;close dialog, set flag

  );strcat

);action tile

  (action_tile
  "cancel"
  ;if cancel button pressed

  "(done_dialog) (setq userclick nil)"
  ;close dialog

  );action_tile

(start_dialog)
```

```
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

   (progn
   ;do the following

    (if (= orth 1)

      (alert "Ortho is ON")
      (alert "Ortho is OFF")
    );if

    (if (= sna 1)

      (alert "Snap is ON")
      (alert "Snap is OFF")
    );if

   );progn

  );if userclick

 (princ)

);defun

(princ)
```

slider.



Syntax:

```
: slider {
  action alignment big_increment fixed_height
  fixed_width height key label layout max_value
  min_value mnemonic small_increment value width
}
```

Note : As a slider should be used in conjunction with an edit box, the DCL and AutoLisp coding for the edit box has been included here.

DCL Coding:

```
lisp48m : dialog {                              //dialog name
          label = "slider" ;                    //give it a label

        : edit_box {                            //define edit box
         key = "eb1" ;                          //give it a name
         label = "Slot &Length (0/All Slot)" ;  //give it a label
         edit_width = 6 ;                       //6 characters only
        }                                       //end edit box

        : slider {                              //define slider
        key = "myslider" ;                      //give it a name
        max_value = 100;                        //upper value
        min_value = 0;                          //lower value
        value = "50";                           //initial value
        }                                       //end slider

        ok_cancel ;                             //predefined OK/Cancel button

        }                                       //end dialog
```

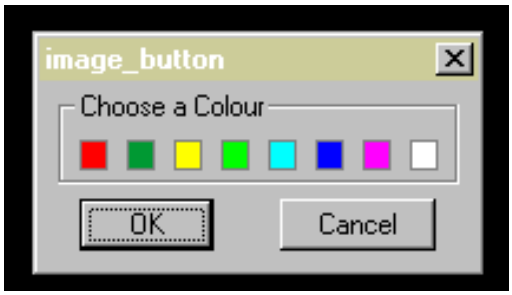
AutoLisp Coding:

```
(defun C:lisp48m ()
;define function
```

```
(setq lngth 50.0)
;preset slot length

(setq dcl_id (load_dialog "lisp48m.dcl"))
;load dialog

(if (not (new_dialog "lisp48m" dcl_id)
;test for dialog

    );not

  (exit)
  ;exit if no dialog

);if

(set_tile "eb1" "50")
;put data into edit box

(mode_tile "eb1" 2)
;switch focus to edit box

(action_tile "myslider"
;if user moves slider

       "(slider_action $value $reason)")
       ;pass arguments to slider_action

(action_tile "eb1"
;if user enters slot length

       "(ebox_action $value $reason)")
       ;pass arguments to ebox_action

(defun slider_action (val why)
;define function

    (if (or (= why 2) (= why 1))
    ;check values

    (set_tile "eb1" val)))
    ;update edit box

(defun ebox_action (val why)
;define function

    (if (or (= why 2) (= why 1))
    ;check values

    (set_tile "myslider" val)))
    ;update slider

(action_tile
  "accept"
  ;if O.K. pressed
```

```
    (strcat
    ;string 'em together

      "(progn

      (setq lngth (get_tile \"eb1\"))"
      ;get slot length

      "(done_dialog)(setq userclick T))"
      ;close dialog, set flag

    );strcat

  );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

   (if userclick
   ;check O.K. was selected

      (alert (strcat "You Selected: " lngth))
      ;display the selected length.

  );if userclick

 (princ)

);defun

(princ)
```

image_button.



Syntax:

```
: image_button {
  action alignment allow_accept aspect_ratio
  color fixed_height fixed_width height
  is_enabled is_tab_stop  key label mnemonic
  width
}
```

DCL Coding:

```
lisp48n : dialog {                              //dialog name
          label = "image_button" ;              //give it a label

                : boxed_row {                   //define a boxed row
                  label = "Choose a Colour";    //gie it a label

                : image_button {                //define image button
                key = "im0" ;                   //give it a name
                height = 1.0 ;                  //define height
                width = 2.0 ;                   //define width
                fixed_width = true;             //fix the width
                allow_accept = true ;           //allow double-click
                }

                : image_button {
                key = "im1" ;
                height = 1.0 ;
                width = 2.0 ;
                fixed_width = true;
                allow_accept = true ;
                }

                : image_button {
                key = "im2" ;
                height = 1.0 ;
                width = 2.0 ;
                fixed_width = true;
                allow_accept = true ;
                }
```

```
            : image_button {
            key = "im3" ;
            height = 1.0 ;
            width = 2.0 ;
            fixed_width = true;
            allow_accept = true ;
            }

            : image_button {
            key = "im4" ;
            height = 1.0 ;
            width = 2.0 ;
            fixed_width = true;
            allow_accept = true ;
            }

            : image_button {
            key = "im5" ;
            height = 1.0 ;
            width = 2.0 ;
            fixed_width = true;
            allow_accept = true ;
            }

            : image_button {
            key = "im6" ;
            height = 1.0 ;
            width = 2.0 ;
            fixed_width = true;
            allow_accept = true ;
            }

            : image_button {
            key = "im7" ;
            height = 1.0 ;
            width = 2.0 ;
            fixed_width = true;
            allow_accept = true ;
            }

            }                                   //end boxed row

      ok_cancel ;                               //predefined OK/Cancel button

      }                                         //end dialog
```

AutoLisp Coding:

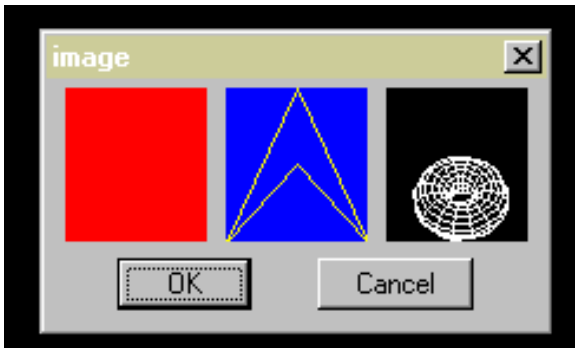
```
(defun C:lisp48n ()
;define function
```

```
(setq dcl_id (load_dialog "lisp48n.dcl"))
;load dialog

(if (not (new_dialog "lisp48n" dcl_id)
;test for dialog

    );not

  (exit)
  ;exit if no dialog

);if

(setq width  (dimx_tile "im0")
;get the image width

      height (dimy_tile "im0")
      ;get the image height

) ;_ end of setq
(start_image "im0")
;start the image

(fill_image 0 0 width height 1)
;fill it with the relevant colour

(end_image)
;end the image

(setq width  (dimx_tile "im1")
      height (dimy_tile "im1")
) ;_ end of setq
(start_image "im1")
(fill_image 0 0 width height 30)
(end_image)

(setq width  (dimx_tile "im2")
      height (dimy_tile "im2")
) ;_ end of setq
(start_image "im2")
(fill_image 0 0 width height 2)
(end_image)

(setq width  (dimx_tile "im3")
      height (dimy_tile "im3")
) ;_ end of setq
(start_image "im3")
(fill_image 0 0 width height 3)
(end_image)

(setq width  (dimx_tile "im4")
      height (dimy_tile "im4")
) ;_ end of setq
```

```
(start_image "im4")
(fill_image 0 0 width height 4)
(end_image)

(setq width  (dimx_tile "im5")
      height (dimy_tile "im5")
) ;_ end of setq
(start_image "im5")
(fill_image 0 0 width height 5)
(end_image)

(setq width  (dimx_tile "im6")
      height (dimy_tile "im6")
) ;_ end of setq
(start_image "im6")
(fill_image 0 0 width height 6)
(end_image)

(setq width  (dimx_tile "im7")
      height (dimy_tile "im7")
) ;_ end of setq
(start_image "im7")
(fill_image 0 0 width height 7)
(end_image)

(action_tile "im0" "(setq la \"0\")")
'get the name of the colour selected

(action_tile "im1" "(setq la \"1\")")
(action_tile "im2" "(setq la \"2\")")
(action_tile "im3" "(setq la \"3\")")
(action_tile "im4" "(setq la \"4\")")
(action_tile "im5" "(setq la \"5\")")
(action_tile "im6" "(setq la \"6\")")
(action_tile "im7" "(setq la \"7\")")

(action_tile
  "accept"
  ;if O.K. pressed

   "(done_dialog)(setq userclick T)"
    ;close dialog, set flag

 );action tile

  (action_tile
  "cancel"
  ;if cancel button pressed

  "(done_dialog) (setq userclick nil)"
  ;close dialog

  );action_tile

(start_dialog)
```

```
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

   (if userclick
   ;check O.K. was selected

      (alert (strcat "You Selected Colour: " la))
      ;display the colour selected.

  );if userclick

 (princ)

);defun

(princ)
```

image.



Syntax:

```
: image {
  action alignment aspect_ratio color
  fixed_height fixed_width height
  is_enabled is_tab_stop  key label mnemonic
  value width
}
```

DCL Coding:

```
lisp48o : dialog {                          //dialog name
          label = "image" ;                 //give it a label

        : row {                             //begin row

        : image {                           //define image tile
          key = "im1" ;                     //give it a name
          height = 5.0 ;                    //and a height
          width = 10.0 ;                    //and now a width
          fixed_width = true;               //fix the width
          fixed_height = true;              //fix the height
        }                                   //end image

        : image {                           //define image tile
          key = "im2" ;                     //give it a name
          height = 5.0 ;                    //add a height
          width = 10.0 ;                    //and now a width
          fixed_width = true;               //fix the width
          fixed_height = true;              //fix the height
        }                                   //end image

        : image {                           //define image tile
          key = "im3" ;                     //give it a name
          height = 5.0 ;                    //add a height
          width = 10.0 ;                    //and now a width
          fixed_width = true;               //fix the width
          fixed_height = true;              //fix the height
          color = graphics_background;      //set background color
```

```
      }                                          //end image

      }                                          //end row

      ok_cancel ;                                //predefined OK/Cancel button

      }                                          //end dialog
```

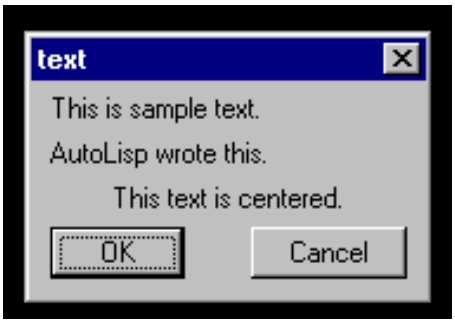
AutoLisp Coding:

```
(defun C:lisp48o ()
;define function

  (setq dcl_id (load_dialog "lisp48o.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48o" dcl_id)
  ;test for dialog

     );not

    (exit)
    ;exit if no dialog

  );if

  (setq w (dimx_tile "im1")
  ;get image tile width

        h (dimy_tile "im1")
        ;get image tile height

  );setq

  (start_image "im1")
  ;start the image

  (fill_image 0 0 w h 1)
  ;fill it with red

  (end_image)
  ;end image

  (setq w (dimx_tile "im2")
  ;get image tile width

        h (dimy_tile "im2")
        ;get image tile height

  );setq

  (start_image "im2")
  ;start the image
```

```
(fill_image 0 0 w h 5)
;fill the image with blue

(setq w1 (/ w 2))
;calculate vector

(setq h1 (/ h 2))
;calculate vector

(vector_image 0 h w1 0 2)
;draw vector

(vector_image w1 0 w h 2)
;draw vector

(vector_image 0 h w1 h1 2)
;draw vector

(vector_image w1 h1 w h 2)
;draw vector

(end_image)
;end image

(setq w (dimx_tile "im3")
;get image tile width

      h (dimy_tile "im3")
      ;get image tile height

);setq

(start_image "im3")
;start the image

(slide_image 0 0 w h "acad(torus)")
;display a slide

(end_image)
;end image


(action_tile
    "accept"
    ;if O.K. pressed

    "(done_dialog) (setq userclick T)"
    ;close dialog, set flag

 );action tile

  (action_tile
  "cancel"
  ;if cancel button pressed
```

```
    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

text.



Syntax:

```
: text {
  alignment fixed_height fixed_width height
  is_bold key label value width
}
```

DCL Coding:

```
lisp48p : dialog {                          //dialog name
          label = "text" ;                  //give it a label

        : text {                            //define text
          key = "txt1";                     //give it a name
          value = "This is sample text.";   //add some text
        }                                   //end text

        : text {                            //define text
          key = "txt2";                     //give it a name
        }                                   //end text

        : text {                            //define text
          key = "txt3";                     //give it a name
          value = "This text is centered."; //center the text
          alignment = centered;             //center the text
        }                                   //end text

        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

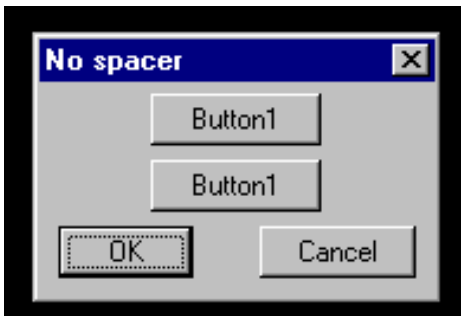
AutoLisp Coding:

```
(defun C:lisp48p ()
;define function

  (setq dcl_id (load_dialog "lisp48p.dcl"))
  ;load dialog
```

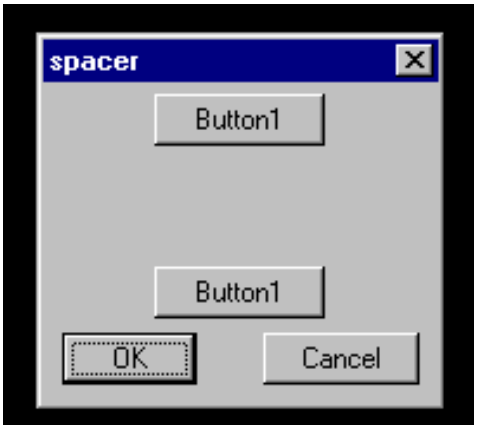
```
  (if (not (new_dialog "lisp48p" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (set_tile "txt2" "AutoLisp wrote this.")

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

spacer.

Dialogue Without Spacer



Dialogue With Spacer



Syntax:

```
: spacer {
  alignment fixed_height fixed_width
  height width
}
```

DCL Coding:

```
lisp48q : dialog {                          //dialog name
          label = "spacer" ;                //give it a label

        : button {                          //define button
          key = "btn1";                     //give it a name
          label = "Button1";                //give it a label
          fixed_width = true;               //fix it's width
          alignment = centered;             //center it
        }                                   //end button

        : spacer {                          //define space
          height = 3;                       //define height
        }                                   //end spacer

        : button {                          //define button
```

```
        key = "btn2";                          //give it a name
        label = "Button1";                     //give it a label
        fixed_width = true;                    //fix it's width
        alignment = centered;                  //center it
      }                                        //end button

      ok_cancel ;                              //predefined OK/Cancel button

      }                                        //end dialog
```

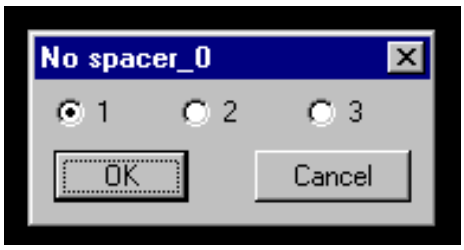
AutoLisp Coding:

```
(defun C:lisp48q ()
;define function

  (setq dcl_id (load_dialog "lisp48q.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48q" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
```

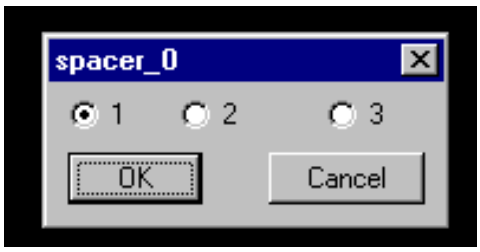
```
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

spacer_0.

Dialogue Without Spacer_0



Dialogue With Spacer_0



Syntax:

```
spacer_0;
```

The spacer_0 tile is defined in the Base.Dcl file.

DCL Coding:

```
lisp48r : dialog {                          //dialog name
            label = "spacer_0" ;            //give it a label

          : row {                           //define row

          : radio_button {                  //define radio button
            label = "1";                    //give it a label
            key = "rb1";                    //give it a name
            value = "1";                    //switch it on
          }                                 //end radio button

          : radio_button {                  //define radio button
            label = "2";                    //give it a label
            key = "rb2";                    //give it a name
          }                                 //end radio button

          spacer_0;                         //add spacer_0

          : radio_button {                  //define radio button
            label = "3";                    //give it a label
            key = "rb3";                    //give it a name
          }                                 //end radio button
```

```
      }                                          //end row

      ok_cancel ;                                //predefined OK/Cancel button

      }                                          //end dialog
```

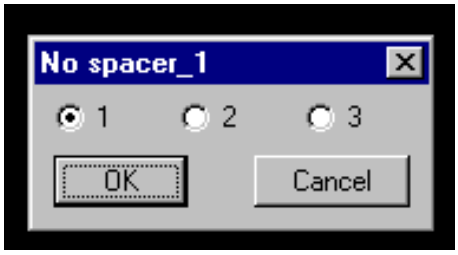
AutoLisp Coding:

```
(defun C:lisp48r ()
;define function

  (setq dcl_id (load_dialog "lisp48r.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48r" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user
```

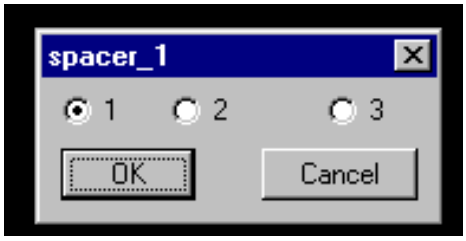
```
    (alert "You selected Cancel")
    ;inform the user

  );if

 (princ)

);defun

(princ)
```

spacer_1.

Dialogue Without Spacer_1



Dialogue With Spacer_1



Syntax:

```
spacer_1;
```

The spacer_1 tile is defined in the Base.Dcl file.

DCL Coding:

```
lisp48s : dialog {                          //dialog name
          label = "spacer_1" ;              //give it a label

        : row {                             //define row

        : radio_button {                    //define radio button
          label = "1";                      //give it a label
          key = "rb1";                      //give it a name
          value = "1";                      //switch it on
        }                                   //end radio button

        : radio_button {                    //define radio button
          label = "2";                      //give it a label
          key = "rb2";                      //give it a name
        }                                   //end radio button

        spacer_1;                           //add spacer_1

        : radio_button {                    //define radio button
          label = "3";                      //give it a label
          key = "rb3";                      //give it a name
        }                                   //end radio button
```

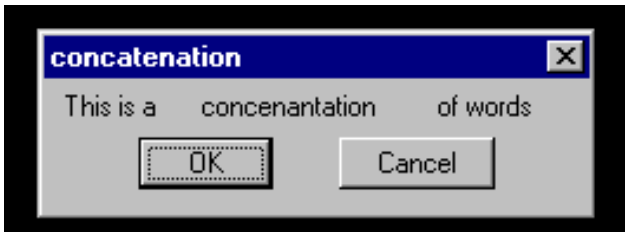
```
      }                                          //end row

      ok_cancel ;                                //predefined OK/Cancel button

      }                                          //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48s ()
;define function

  (setq dcl_id (load_dialog "lisp48s.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48s" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user
```

```
    (alert "You selected Cancel")
    ;inform the user

  );if

 (princ)

);defun

(princ)
```

concatenation.



Syntax:

```
: concatenation {
}
```

The concatenation tile is defined in the Base.Dcl file.

DCL Coding:

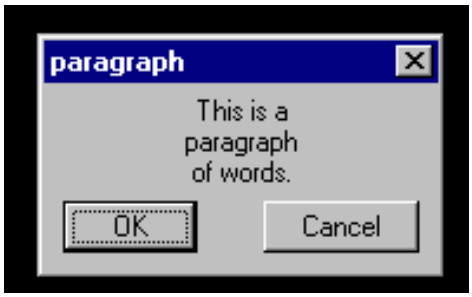
```
lisp48t : dialog {                          //dialog name
          label = "concatenation" ;         //give it a label

        : concatenation {                   //define concatenation

        : text_part {                       //text part
          label = "This is a";              //add text
        }                                   //end text

        : text_part {                       //text part
          label = "concenantation";         //add text
        }                                   //end text

        : text_part {                       //text part
          label = "of words";               //add text
        }                                   //end text

        }                                   //end concatenation


        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48t ()
;define function

  (setq dcl_id (load_dialog "lisp48t.dcl"))
  ;load dialog
```

```
  (if (not (new_dialog "lisp48t" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

paragraph.



Syntax:

```
: paragraph {
}
```

The paragraph tile is defined in the Base.Dcl file.

DCL Coding:

```
lisp48u : dialog {                          //dialog name
          label = "paragraph" ;             //give it a label

        : paragraph {                       //define paragraph

        : text_part {                       //text
          label = "This is a";              //add text
          alignment = centered;             //center the text
        }                                   //end text

        : text_part {                       //text
          label = "paragraph";              //add text
          alignment = centered;             //center the text
        }                                   //end text

        : text_part {                       //text
          label = "of words.";              //add text
          alignment = centered;             //center the text
        }                                   //end text

        }                                   //end paragraph


        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48u ()
```

```
;define function

  (setq dcl_id (load_dialog "lisp48u.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48u" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

text_part.

Syntax:

```
: text_part {
  key label
}
```

The text_part tile is defined in the Base.Dcl file.

DCL Coding:

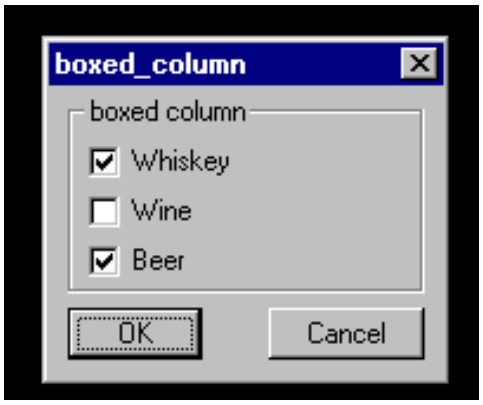
```
lisp48v : dialog {                          //dialog name
          label = "text_part" ;             //give it a label

        : paragraph {                       //define paragraph

        : text_part {                       //text
          label = "You are on";             //add text
          alignment = centered;             //center the text
        }                                   //end text

        : text_part {                       //text
          label = "Layer";                  //add text
          alignment = centered;             //center the text
        }                                   //end text

        : text_part {                       //text
          key = "txt1";                     //give it a name
          alignment = centered;             //center the text
        }                                   //end text

        }                                   //end paragraph


        ok_cancel ;                         //predefined OK/Cancel button

        }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48v ()
;define function

  (setq la (getvar "clayer"))

  (setq dcl_id (load_dialog "lisp48v.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48v" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (set_tile "txt1" la)

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)
```

```
);defun

(princ)
```

boxed_column.



Syntax:

```
: boxed_column {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLisp coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48w : dialog {                          //dialog name
          label = "boxed_column" ;          //give it a label

        : boxed_column {                    //define boxed column
          label = "boxed column";           //give it a label

        : toggle {                          //define a toggle
          key = "tog1";                     //give it a name
          label = "Toggle 1";               //give it a label
          value = "1";                      //switch it on
        }                                   //end toggle

        : toggle {                          //define a toggle
          key = "tog2";                     //give it a name
          label = "Toggle 1";               //give it a label
          value = "2";                      //switch it off
        }                                   //end toggle

        : toggle {                          //define a toggle
          key = "tog3";                     //give it a name
          label = "Toggle 1";               //give it a label
          value = "1";                      //switch it on
        }                                   //end toggle

        }                                   //end boxed column
```

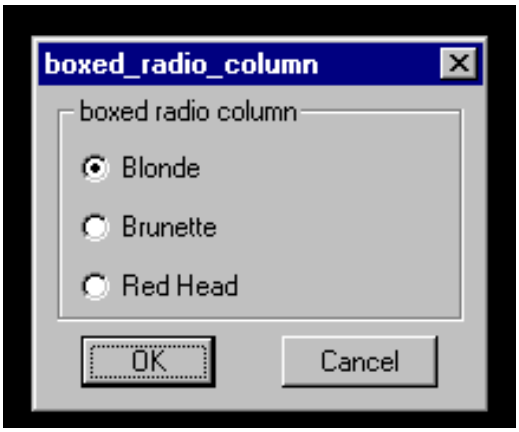
```
      ok_cancel ;                                  //predefined OK/Cancel button

      }                                            //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48w ()
;define function

  (setq dcl_id (load_dialog "lisp48w.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48w" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user
```

```
    (alert "You selected Cancel")
    ;inform the user

  );if

 (princ)

);defun

(princ)
```

boxed_radio_column.



Syntax:

```
: boxed_radio_column {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLisp coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48x : dialog {                          //dialog name
          label = "boxed_radio_column";     //give it a label

        : boxed_radio_column {              //define boxed radio column
          label = "boxed radio column";     //give it a label

        : radio_button {                    //define a radio button
          key = "rad1";                     //give it a name
          label = "Blonde";                 //give it a label
          value = "1";                      //switch it on
        }                                   //end radio button

        : radio_button {                    //define a radio button
          key = "rad2";                     //give it a name
          label = "Brunette";               //give it a label
        }                                   //end radio button

        : radio_button {                    //define a radio button
          key = "rad3";                     //give it a name
          label = "Red Head";               //give it a label
        }                                   //end radio button

        }                                   //end boxed radio column
```

```
      ok_cancel ;                                  //predefined OK/Cancel button

      }                                            //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48x ()
;define function

  (setq dcl_id (load_dialog "lisp48x.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48x" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user
```

```
    (alert "You selected Cancel")
    ;inform the user

  );if

 (princ)

);defun

(princ)
```

boxed_radio_row.

Syntax:

```
: boxed_radio_row {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLisp coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48y : dialog {                          //dialog name
            label = "boxed_radio_row";      //give it a label

          : boxed_radio_row {               //define boxed radio row
            label = "boxed radio row";      //give it a label

          : radio_button {                  //define a radio button
            key = "rad1";                   //give it a name
            label = "Blonde";               //give it a label
            value = "1";                    //switch it on
          }                                 //end radio button

          : radio_button {                  //define a radio button
            key = "rad2";                   //give it a name
            label = "Brunette";             //give it a label
          }                                 //end radio button

          : radio_button {                  //define a radio button
            key = "rad3";                   //give it a name
            label = "Red Head";             //give it a label
          }                                 //end radio button

          }                                 //end boxed radio row


          ok_cancel ;                       //predefined OK/Cancel button
```

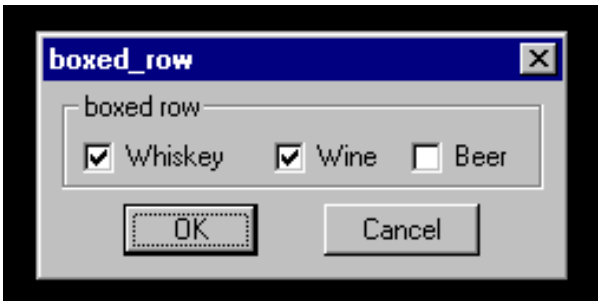
```
    }                                          //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48y ()
;define function

  (setq dcl_id (load_dialog "lisp48y.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48y" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user
```

```
  );if
 (princ)
);defun
(princ)
```

boxed_row.



Syntax:

```
: boxed_row {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLisp coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48z : dialog {                          //dialog name
            label = "boxed_row";            //give it a label

          : boxed_row {                     //define boxed row
            label = "boxed row";            //give it a label

          : toggle {                        //define a toggle
            key = "tog1";                   //give it a name
            label = "Whiskey";              //give it a label
            value = "1";                    //switch it on
          }                                 //end toggle

          : toggle {                        //define a toggle
            key = "tog2";                   //give it a name
            label = "Wine";                 //give it a label
            value = "1";                    //switch it on
          }                                 //end toggle

          : toggle {                        //define a toggle
            key = "tog3";                   //give it a name
            label = "Beer";                 //give it a label
            value = "0";                    //switch it on
          }                                 //end toggle

          }                                 //end boxed row
```

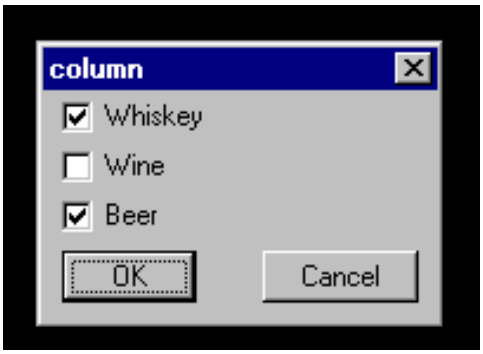
```
      ok_cancel ;                              //predefined OK/Cancel button

      }                                        //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48z ()
;define function

  (setq dcl_id (load_dialog "lisp48z.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48z" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
```

```
    ;inform the user
  );if
 (princ)
);defun
(princ)
```

column.



Syntax:

```
: column {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLips coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48aa : dialog {                          //dialog name
           label = "column" ;                //give it a label

         : column {                          //define column


         : toggle {                          //define a toggle
           key = "tog1";                     //give it a name
           label = "Toggle 1";               //give it a label
           value = "1";                      //switch it on
         }                                   //end toggle

         : toggle {                          //define a toggle
           key = "tog2";                     //give it a name
           label = "Toggle 1";               //give it a label
           value = "2";                      //switch it off
         }                                   //end toggle

         : toggle {                          //define a toggle
           key = "tog3";                     //give it a name
           label = "Toggle 1";               //give it a label
           value = "1";                      //switch it on
         }                                   //end toggle

         }                                   //end column
```

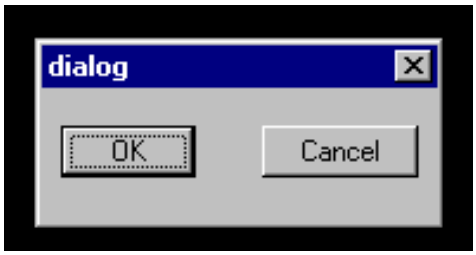
```
      ok_cancel ;                              //predefined OK/Cancel button

      }                                        //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48aa ()
;define function

  (setq dcl_id (load_dialog "lisp48aa.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48aa" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user
```

```
    (alert "You selected Cancel")
    ;inform the user

  );if

 (princ)

);defun

(princ)
```

dialog.



Syntax:

```
: dialog {
  initial_focus label value
}
```

DCL Coding:

```
lisp48ab : dialog {                          //dialog name
          label = "dialog" ;                 //give it a label

        ok_cancel ;                          //predefined OK/Cancel button

        }                                    //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48ab ()
;define function

  (setq dcl_id (load_dialog "lisp48ab.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48ab" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag
```

```
   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

radio_column.

Syntax:

```
: radio_column {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLips coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48ac : dialog {                          //dialog name
          label = "radio_column";            //give it a label

        : radio_column {                     //define radio column

        : radio_button {                     //define a radio button
          key = "rad1";                      //give it a name
          label = "Blonde";                  //give it a label
          value = "1";                       //switch it on
        }                                    //end radio button

        : radio_button {                     //define a radio button
          key = "rad2";                      //give it a name
          label = "Brunette";                //give it a label
        }                                    //end radio button

        : radio_button {                     //define a radio button
          key = "rad3";                      //give it a name
          label = "Red Head";                //give it a label
        }                                    //end radio button

        }                                    //end radio column


        ok_cancel ;                          //predefined OK/Cancel button
```

```
    }                                          //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48ac ()
;define function

  (setq dcl_id (load_dialog "lisp48ac.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48ac" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user
```

```
  );if
 (princ)
);defun
(princ)
```

radio_row.

Syntax:

```
: radio_row {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLisp coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48ad : dialog {                          //dialog name
           label = "radio_row";              //give it a label

         : radio_row {                       //define radio row

         : radio_button {                    //define a radio button
           key = "rad1";                     //give it a name
           label = "Blonde";                 //give it a label
           value = "1";                      //switch it on
         }                                   //end radio button

         : radio_button {                    //define a radio button
           key = "rad2";                     //give it a name
           label = "Brunette";               //give it a label
         }                                   //end radio button

         : radio_button {                    //define a radio button
           key = "rad3";                     //give it a name
           label = "Red Head";               //give it a label
         }                                   //end radio button

         }                                   //end radio row


         ok_cancel ;                         //predefined OK/Cancel button

         }                                   //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48ad ()
;define function

  (setq dcl_id (load_dialog "lisp48ad.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48ad" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)
```

```
);defun

(princ)
```

row.

Syntax:

```
: row {
  alignment children_alignment children_fixed_height
  children_fixed_width fixed_height fixed_width
  height label width
}
```

Note: This Tile Cluster is for display purpose only.
No AutoLisp coding has been included for the Active Children Tiles.

DCL Coding:

```
lisp48ae : dialog {                          //dialog name
          label = "row";                     //give it a label

        : row {                              //define row

        : toggle {                           //define a toggle
          key = "tog1";                      //give it a name
          label = "Whiskey";                 //give it a label
          value = "1";                       //switch it on
        }                                    //end toggle

        : toggle {                           //define a toggle
          key = "tog2";                      //give it a name
          label = "Wine";                    //give it a label
          value = "1";                       //switch it on
        }                                    //end toggle

        : toggle {                           //define a toggle
          key = "tog3";                      //give it a name
          label = "Beer";                    //give it a label
          value = "0";                       //switch it on
        }                                    //end toggle

        }                                    //end row


        ok_cancel ;                          //predefined OK/Cancel button
```

```
    }                                          //end dialog
```

AutoLisp Coding:

```
(defun C:lisp48ae ()
;define function

  (setq dcl_id (load_dialog "lisp48ae.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp48ae" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

  (if userclick
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user
```

```
  );if
 (princ)
);defun
(princ)
```

DCL Tile Attributes

This tutorial will give you a detailed explanation of all attributes used in the design and implementation of dialog boxes used within AutoCAD/AutoLisp.
I have provided a sample dialog image of each of the various aspects of attributes, and the sample DCL Coding that was used to create the dialog. The sample AutoLisp coding required to display each dialog can be found in the downloadable tutorial.

Attributes in DCL define the layout and functionality of tiles. If you are familiar with Visual Basic, you will find that they are very similar to properties.
DCL Attributes consist of a name and a value.

Note: The value of certain attributes can change at run-time, with user input or 'set_tile' calls.

The value of a tile's attribute must be one of the following type:

Integer

- Numeric Values (integer or real number) that represent distances such as width, or height of a tile.

Real Number

- A fractional real number must have a leading digit.
  e.g. 0.1 *not* .1.

Quoted String

- A quoted string consists of text enclosed in quotation marks (" "). Attribute values are case sensitive. "RB1" is not the same as "rb1".

Reserved Word

- A reserved word is an identifier made up of alphanumeric characters, beginning with a letter. e.g. *true* or *false.* Reserved words are case sensitive: *True* does not equal *true.*

Global Attributes.

Attribute :

- alignment
- fixed_height
- fixed_width
- height
- width

Applies To :

- All Tiles

Attributes Associated with Action Tiles

Attribute :

- action
- is_enabled
- is_tab_stop
- key
- mnemonic

Applies To :

- button; edit_box; image_button; list_box; popup_list; radio_button; slider; toggle; radio_column; radio_row.

Attributes Associated with Tile Clusters

Attribute :

- children_alignment
- children_fixed_height
- children_fixed_width

Applies To :

- row; column; radio_row; radio_column; boxed_row; boxed_column; boxed_radio_row; boxed_radio_column.

Attributes Associative with Specific Tiles

Attribute :

- allow_accept

Applies To :

- edit_box; image_button; list_box

Attribute :

- aspect_ration

Applies To :

- image; image_button.

Attribute :

- big_increment

Applies To :

- slider.

Attribute :

- color

Applies To :

- image; image_button.

Attribute :

- edit_limit

Applies To :

- edit_box.

Attribute :

- edit_width

Applies To :

- edit_box; popup_list.

Attribute :

- fixed_width_font

Applies To :

- edit_box; popup_list.

Attribute :

- initial_focus

Applies To :

- dialog.

Attribute :

- is_cancel

Applies To :

- button.

Attribute :

- is_default

Applies To :

- button.

Attribute :

- label

Applies To :

- boxed_row; boxed_column; boxed_radio_row; boxed_radio_column; button; dialog; edit_box list_box; popup_list; radio_button; text; toggle.

Attribute :

- layout

Applies To :

- slider.

Attribute :

- list

Applies To :

- list_box; popup_list.

Attribute :

- max_value

Applies To :

- slider.

Attribute :

- min_value

Applies To :

- slider.

Attribute :

- multiple_select

Applies To :

- list_box.

Attribute :

- password_char

Applies To :

- edit_box.

Attribute :

- small_increment

Applies To :

- slider.

Attribute :

- tabs

Applies To :

- list_box; popup_list.

Attribute :

- tab_truncate

Applies To :

- list_box; popup_list

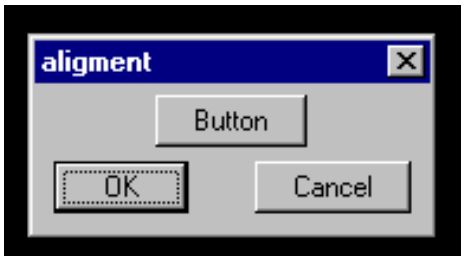
Attribute :

- value

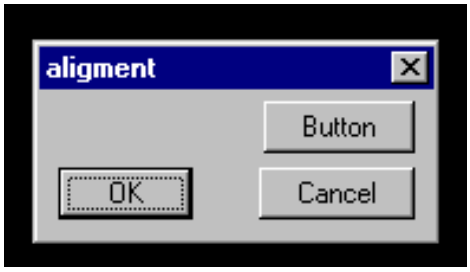
Applies To :

- text; all active tiles (except buttons and image_buttons).

alignment



Aligned Centered.



Aligned Right.

This attribute specifies the horizontal or vertical positioning of a tile within its cluster. If a tile is within a column, the possible values are:

left, right or centered. (default: left)

If a tile is within a row, the possible values are:

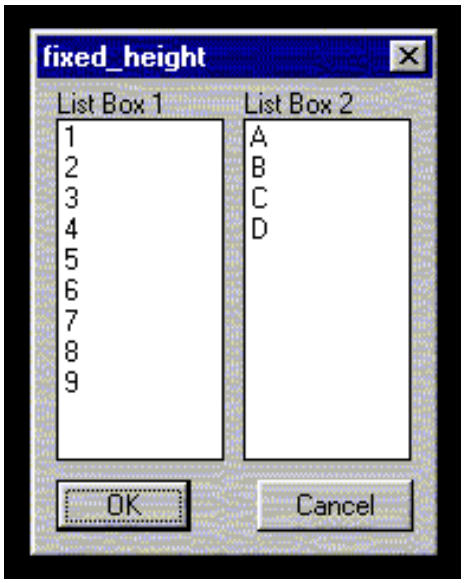
top, bottom or centered. (default: centered)

You cannot specify the alignment along the long axis of a cluster. The first and last tiles in the cluster always align themselves to the ends of the column or row.

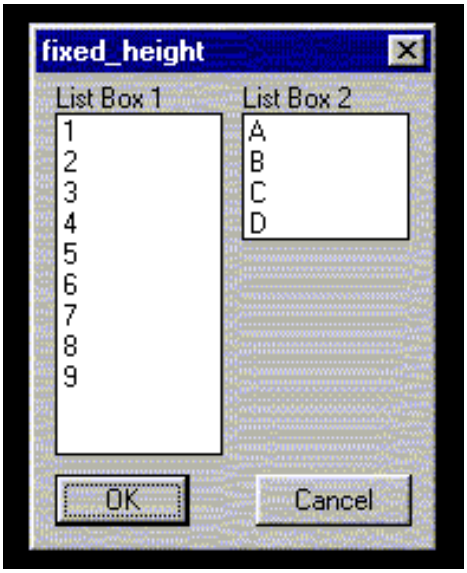
DCL Code Sample

```
lisp49a : dialog {                          //dialog name
            label = "aligment" ;            //give it a label

          : button {                        //define button
            label="Button";                 //give it a label
            fixed_width=true;               //fix the width
            alignment=right;                //align right
          }                                 //end button

          ok_cancel ;                       //predefined OK/Cancel button

          }                                 //end dialog
```

fixed_height



Fixed_Height Not Set.



Fixed_Height Set to True.

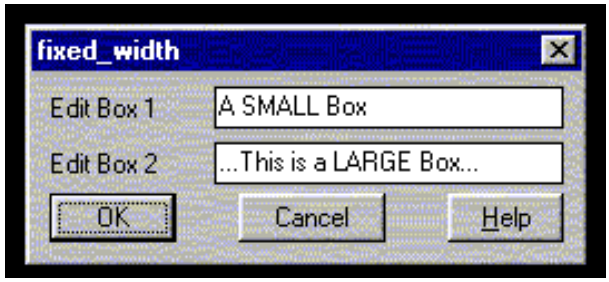
This attribute specifies whether a tiles height is allowed to fill the available white space. If this attribute is true, the tile will be restricted to it's preset height attribute.
Possible values are: *true* or *false*. (default: *false*)

DCL Code Sample

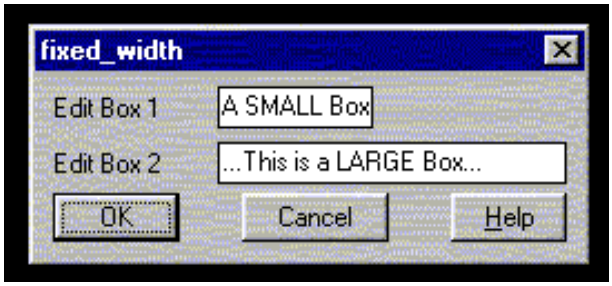
```
lisp49b : dialog {                              //dialog name
          label = "fixed_height";               //give it a label

        : row {                                 //define a row

        : list_box {                            //define a list box
          key = "lb1";                          //give it a name
```

```
  label = "List Box 1";                  //give it a label
  list = "1\n2\n3\n4\n5\n6\n7\n8\n9";    //make a list
}

: list_box {                             //define a list box
  key = "lb2";                           //give it a name
  label = "List Box 2";                  //give it a label
  list = "A\nB\nC\nD";                   //make a list
  height = 5;                            //set the height
  fixed_height = true;                   //fix the height
  alignment = top;                       //align to top
}                                        //end list box

}                                        //end row

ok_cancel ;                              //predefined OK/Cancel button

}                                        //end dialog
```

fixed_width



Fixed_Width Not Set.



Fixed_Width Set to True.

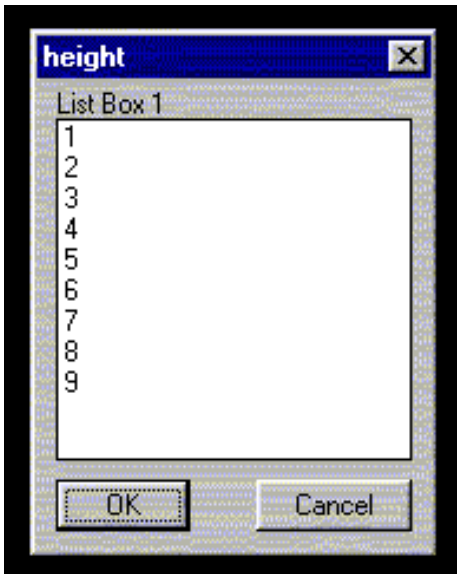
This attribute specifies whether a tiles width is allowed to fill the available white space. If this attribute is true, the tile will be restricted to it's preset width attribute.

Possible values are: *true* or *false*. (default = *false*)

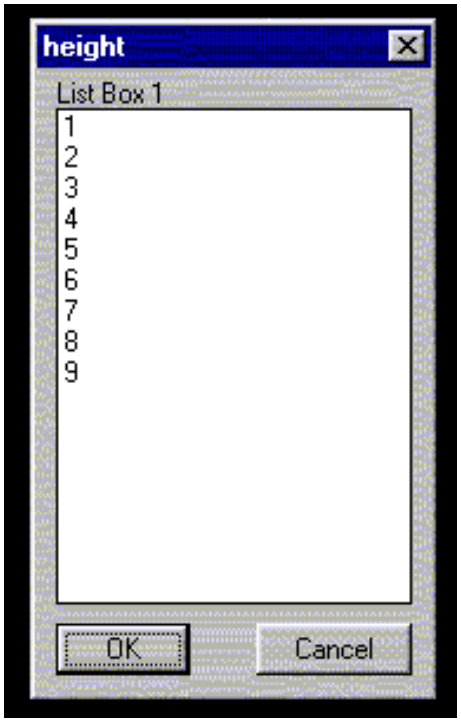
DCL Code Sample

```
lisp49c : dialog {                              //dialog name
            label = "fixed_width";              //give it a label

          : edit_box {                          //define an edit box
            key = "eb1";                        //give it a name
            label = "Edit Box 1";               //give it a label
            width = 24;                         //give it a width
            value = "A SMALL Box";              //give it a value
            fixed_width = true;                 //fix the width
          }                                     //end edit box

          : edit_box {                          //define an edit box
            key = "eb2";                        //give it a name
            label = "Edit Box 2";               //give it a label
            value = "...This is a LARGE Box..."; //give it a value
          }                                     //end edit box

          ok_cancel_help ;                      //predefined OK/Cancel/Help button

          }                                     //end dialog
```

height



Height Not Set.



Height Set to 15.

This attribute specifies the height of tile. The height value must be an integer or a real number. The height of image tiles and image buttons must be specified.
Note: The *height* of a tile is the MINIMUM value.

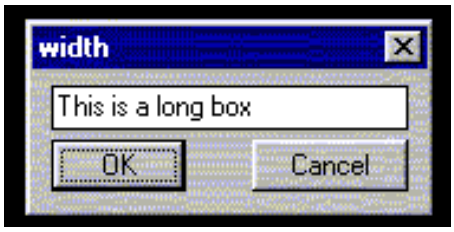
Possible values are platform dependent.

DCL Code Sample

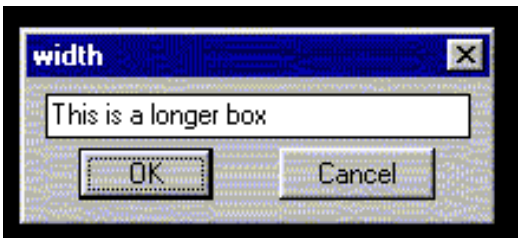
```
lisp49d : dialog {                                  //dialog name
```

```
  label = "height";                          //give it a label

: list_box {                                 //define a list box
  key = "lb1";                               //give it a name
  label = "List Box 1";                      //give it a label
  list = "1\n2\n3\n4\n5\n6\n7\n8\n9";        //make a list
  height = 15;                               //give it a height
}                                            //end list box

ok_cancel ;                                  //predefined OK/Cancel button

}                                            //end dialog
```

width



Width Not Set.



Width Set to 30.

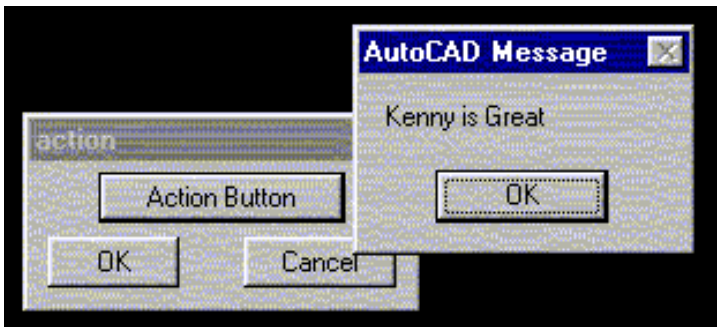
This attribute specifies the width of tile. The width value must be an integer or a real value. The width of image tiles and image buttons must be specified.
Note: The *width* of a tile is the MINIMUM value.

Possible values are platform dependent.

DCL Code Sample

```
lisp49e : dialog {                              //dialog name
            label = "width";                    //give it a label

          : edit_box {                          //define an edit box
            key = "eb1";                        //give it a name
            value = "This is a longer box";     //give it a value
            width = 30;                         //give it a width
          }                                     //end edit box

          ok_cancel ;                           //predefined OK/Cancel button

          }                                     //end dialog
```

action



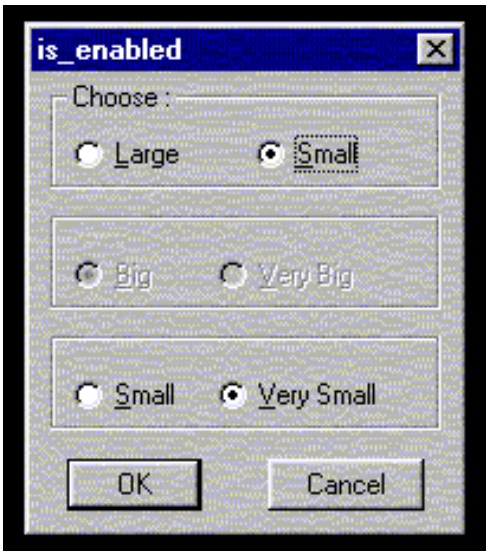
Action Attribute in action.

This attribute specifies an AutoLisp expression to activate. The value must be quoted. eg. "(acad_colordlg 3)"

DCL Code Sample

```
lisp49f : dialog {                                  //dialog name
            label = "action";                       //give it a label

          : button {                                //define a button
            fixed_width = true;                     //fix it's width
            alignment = centered;                   //center it
            label = "Action Button";                //give it a label
            action = "(alert \"Kenny is Great\")"; //Trigger an AutoLisp Function
          }                                         //end button

          ok_cancel ;                               //predefined OK/Cancel button

          }                                         //end dialog
```

is_enabled



Enabled/Disabled.

This attribute specifies whether a tile is initially grayed out. (Unavailable.)
Possible values are: *true* or *false*. (default: *true*).

DCL Code Sample

```
lisp49g : dialog {                          //dialog name
            label = "is_enabled";           //give it a label

          : boxed_radio_row {               //define boxed radio row
            label = "Choose :";             //give it a label

          : radio_button {                  //define radio button
            label = "&Large";               //give it a label
            key = "rb1";                    //give it a name
            value = "1";                    //make it default
          }                                 //end radio button

          : radio_button {                  //define radio button
            label = "&Small";               //give it a label
            key = "rb2";                    //give it a name
          }                                 //end radio button

          }                                 //end boxed radio row

          : boxed_radio_row {               //define boxed radio row

          : radio_button {                  //define radio button
            label = "&Big";                 //give it a label
            key = "rb3";                    //give it a name
            value = "1";                    //make it default
          }                                 //end radio button
```

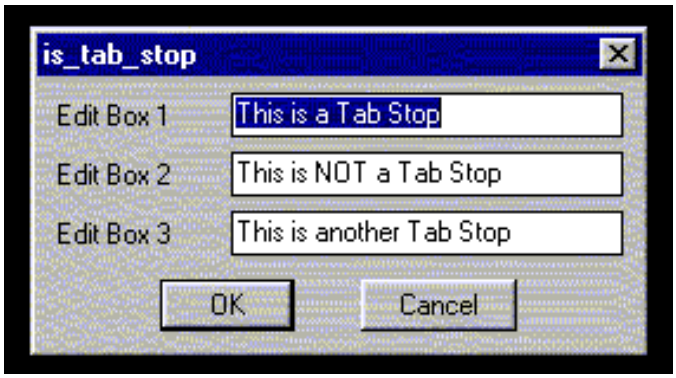
```
      : radio_button {                          //define radio button
        label = "&Very Big";                    //give it a label
        key = "rb4";                            //give it a name
      }                                         //end radio button

      }                                         //end boxed radio row

      : boxed_radio_row {                       //define boxed radio row

      : radio_button {                          //define radio button
        label = "&Small";                       //give it a label
        key = "rb5";                            //give it a name
        is_enabled = false;                     //switch off
      }                                         //end radio button

      : radio_button {                          //define radio button
        label = "&Very Small";                  //give it a label
        key = "rb6";                            //give it a name
        value = "1";                            //make it default
        is_enabled = false;                     //switch off
      }                                         //end radio button

      }                                         //end boxed radio row

      spacer;                                   //define spacer

      ok_cancel ;                               //predefined OK/Cancel button

      }                                         //end dialog
```

AutoLisp Coding:

```
(defun C:lisp49g ()
;define function

  (setq dcl_id (load_dialog "lisp49g.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp49g" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (action_tile "rb2"
     "(mode_tile \"rb3\" 1)
      (mode_tile \"rb4\" 1)
      (mode_tile \"rb5\" 0)
      (mode_tile \"rb6\"0)"
```

```
  )

  (action_tile "rb1"
      "(mode_tile \"rb3\" 0)
      (mode_tile \"rb4\" 0)
      (mode_tile \"rb5\" 1)
      (mode_tile \"rb6\" 1)"
  )


  (action_tile
      "accept"
      ;if O.K. pressed

      "(done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

 (princ)

);defun

(princ)
```

is_tab_stop



Tab Stops

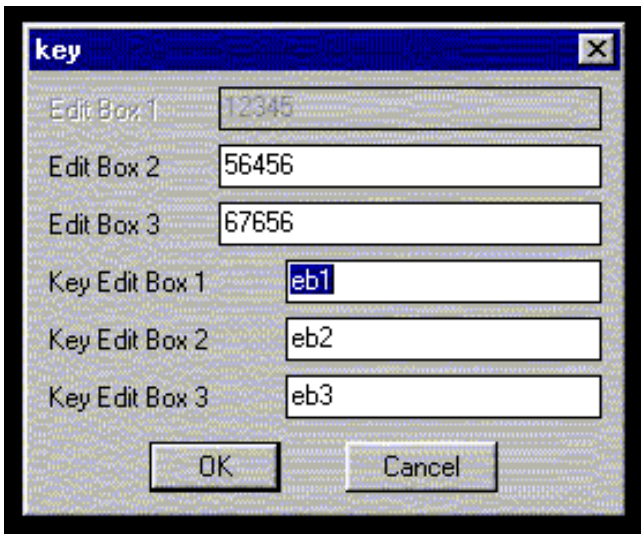
This attribute specifies whether a tile is a tab stop. (receives keyboard focus when the user moves between tiles by pressing the "Tab" key.) If a tile is disabled, it isn't a tab stop even if this attribute is set to *true*.
If *false*, the tile is not a tab stop.
Possible values are: *true* or *false*. (default: *true*).

DCL Code Sample

```
lisp49h : dialog {                              //dialog name
            label = "is_tab_stop";              //give it a label

          : edit_box {                          //define edit box
            key = "eb1";                        //give it a name
            label = "Edit Box 1";               //give it a label
            value = "This is a Tab Stop";       //give it a value
            width = 40;                         //give it a width
            fixed_width = true;                 //fix the width
          }                                     //end edit box

          : edit_box {                          //define edit box
            key = "eb2";                        //give it a name
            label = "Edit Box 2";               //give it a label
            value = "This is NOT a Tab Stop";   //give it a value
            width = 40;                         //give it a width
            fixed_width = true;                 //fix the width
            is_tab_stop = false;                //no tab stop
          }                                     //end edit box

          : edit_box {                          //define edit box
            key = "eb3";                        //give it a name
            label = "Edit Box 3";               //give it a label
            value = "This is another Tab Stop"; //give it a value
            width = 40;                         //give it a width
            fixed_width = true;                 //fix the width
          }                                     //end edit box
```

```
spacer;                                 //define spacer
ok_cancel ;                             //predefined OK/Cancel button
}                                       //end dialog
```

key



A List of Keys

This attribute specifies an ASCII name that your program uses to refer to a specific tile. It must be a quoted string (no default) and is case sensitive.
(*"eb1"* does not equal *"Eb1"*).
Each key value MUST be unique within a dialogue definition.

DCL Code Sample

```
lisp49i : dialog {                                  //dialog name
            label = "key";                          //give it a label

          : edit_box {                              //define edit box
            key = "eb1";                            //give it a name
            label = "Edit Box 1";                   //give it a label
            width = 40;                             //give it a width
            fixed_width = true;                     //fix the width
            value = "Enter a Number and press Tab"; //give it a value
          }                                         //end edit box

          : edit_box {                              //define edit box
            key = "eb2";                            //give it a name
            label = "Edit Box 2";                   //give it a label
            width = 40;                             //give it a width
            fixed_width = true;                     //fix the width
            value = "Enter a Number and press Tab"; //give it a value
          }                                         //end edit box

          : edit_box {                              //define edit box
            key = "eb3";                            //give it a name
            label = "Edit Box 3";                   //give it a label
            width = 40;                             //give it a width
            fixed_width = true;                     //fix the width
            value = "Enter a Number and press Tab"; //give it a value
          }                                         //end edit box
```

```
    : edit_box {                          //define edit box
      key = "eb4";                        //give it a name
      label = "Key Edit Box 1";           //give it a label
      width = 40;                         //give it a width
      fixed_width = true;                 //fix the width
    }                                     //end edit box

    : edit_box {                          //define edit box
      key = "eb5";                        //give it a name
      label = "Key Edit Box 2";           //give it a label
      width = 40;                         //give it a width
      fixed_width = true;                 //fix the width
    }                                     //end edit box

    : edit_box {                          //define edit box
      key = "eb6";                        //give it a name
      label = "Key Edit Box 3";           //give it a label
      width = 40;                         //give it a width
      fixed_width = true;                 //fix the width
    }                                     //end edit box


    spacer;                               //define spacer

    ok_cancel ;                           //predefined OK/Cancel button

    }                                     //end dialog
```

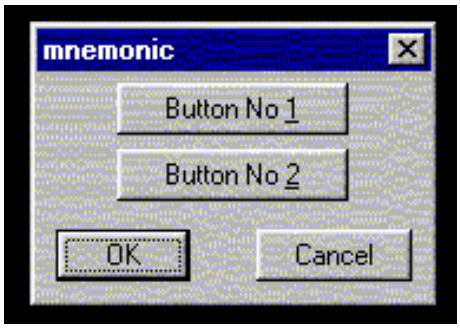
AutoLisp Code Sample

```
(defun C:lisp49i ()
;define function

  (setq dcl_id (load_dialog "lisp49i.dcl"))
  ;load dialog

  (if (not (new_dialog "lisp49i" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

  (mode_tile "eb1" 2)
  (mode_tile "eb1" 3)

     (action_tile "eb1"
     ;if this tile is selected

       "(setq a $key)
```

```
      ;retrieve it's name

      (set_tile \"eb4\" a)"
      ;set the tile 'eb4' to it's name

   )

   (action_tile "eb2"
   ;if this tile is selected

     "(setq b $key)
      ;retrieve it's name

      (set_tile \"eb5\" b)
      ;set the tile 'eb5' to it's name

      (mode_tile a 1)"
      ;switch off the tile 'eb2'

   )

   (action_tile "eb3"
   ;if this tile is selected

     "(setq c $key)
      ;retrieve it's name

      (set_tile \"eb6\" c)"
      ;set the tile 'eb6' to it's name

   )

(action_tile
    "accept"
    ;if O.K. pressed

    "(done_dialog) (setq userclick T)"
    ;close dialog, set flag

 );action tile

  (action_tile
  "cancel"
  ;if cancel button pressed

  "(done_dialog) (setq userclick nil)"
  ;close dialog, lower flag

  );action_tile

(start_dialog)
;start dialog

(unload_dialog dcl_id)
;unload

(if userclick
```

```
  ;if OK selected

     (alert "You selected OK")
     ;inform the user

     (alert "You selected Cancel")
     ;inform the user

  );if

 (princ)

);defun

(princ)
```

mnemonic



Mnemonic

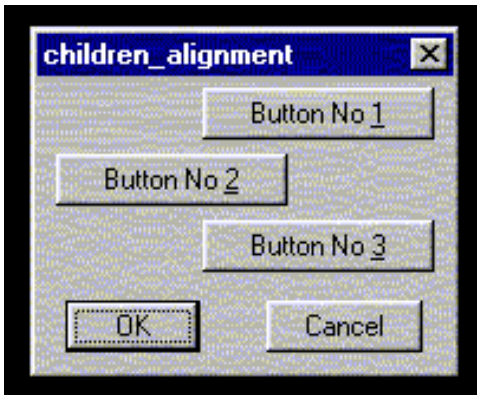
This attribute specifies a keyboard mnemonic character for a tile. The mnemonic character is underlined in the tile's label. The value is a quoted string of a single character that must be one of the letters of the tile's label. (No default).
The character does not have to be unique to the dialogue box. If more that one tile has the same mnemonic, you press Tab to cycle through the tiles sequentially.

The *label* attribute can also specify a mnemonic by preceding a character with an ampersand: (&).

DCL Code Sample

```
lisp49j : dialog {                          //dialog name
            label = "mnemonic";             //give it a label

          : button {                        //define a button
            key = "btn1";                   //give it a name
            label = "Button No 1";          //give it a label
            fixed_width = true;             //fix it's width
            alignment = centered;           //centre it
            mnemonic = "1";                 //the mnemonic
          }                                 //end button

          : button {                        //define a button
            key = "btn2";                   //give it a name
            label = "Button No &2";         //give it a label
                                            //and define the
                                            //mnemonic
            fixed_width = true;             //fix it's width
            alignment = centered;           //centre it
          }                                 //end button


          spacer;                           //add a spacer

          ok_cancel ;                       //predefined OK/Cancel button

          }                                 //end dialog
```

children_alignment



Aligned Center

This attribute specifies the default alignment for all tiles in a cluster.
It does not override a child's alignment attribute if that is set.

Possible values for columns are *left*; *right* or *centered*.
default: *left*

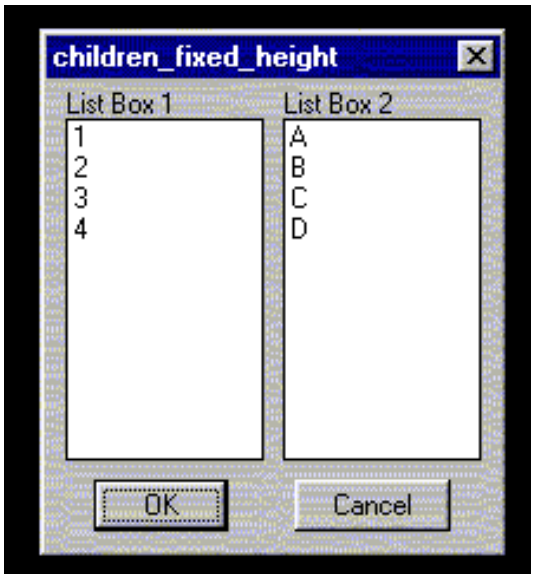
Possible values for rows are *top*; *bottom* or *centered*.
default: *centered*

DCL Code Sample

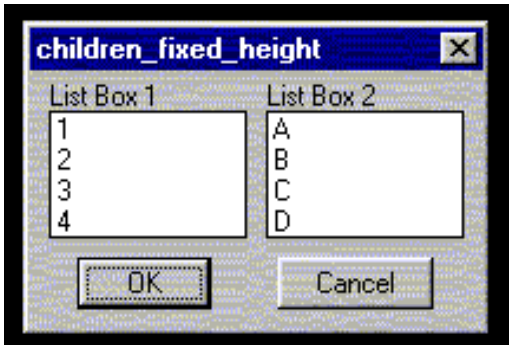
```
lisp49k : dialog {                              //dialog name
            label = "children_alignment";       //give it a label

        : column {                              //define a column
            children_alignment = right;         //put tiles on the right

        : button {                              //define a button
            key = "btn1";                       //give it a name
            label = "Button No &1";             //give it a label
            fixed_width = true;                 //fix it's width
          }                                     //end button

        : button {                              //define a button
            key = "btn2";                       //give it a name
            label = "Button No &2";             //give it a label
            fixed_width = true;                 //fix it's width
            alignment = left;                   //override default
                                                //alignment
          }                                     //end button

        : button {                              //define a button
            key = "btn3";                       //give it a name
            label = "Button No &3";             //give it a label
            fixed_width = true;                 //fix it's width
```

```
}                                   //end button
}                                   //end column
spacer;                             //add a spacer
ok_cancel ;                         //predefined OK/Cancel button
}                                   //end dialog
```

children_fixed_height



Children_Fixed_Height Not Set.



Children_Fixed_Height Set to True.

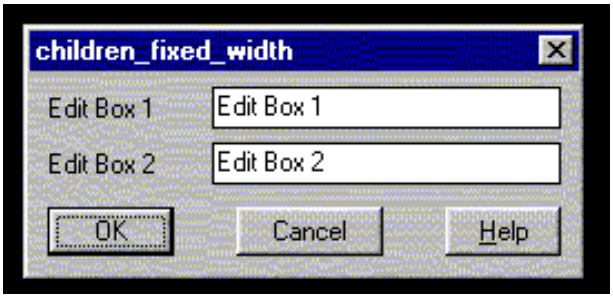
This attribute specifies the default height for all the tiles in a cluster.
It does not override a child's height attribute if it is specified. Possible values are: *true* or *false*.
(default: *false*)

DCL Code Sample

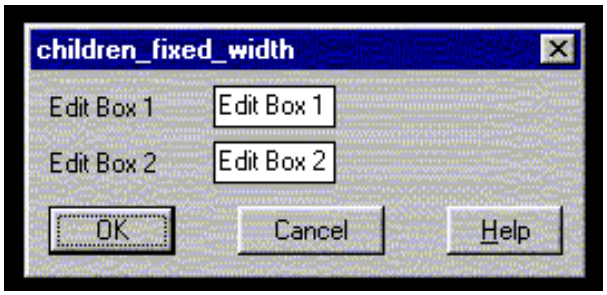
```
lisp49l : dialog {                              //dialog name
            label = "children_fixed_height";    //give it a label

          : row {                               //define a row
            children_fixed_height = true;       //fix the height

          : list_box {                          //define a list box
            key = "lb1";                        //give it a name
            label = "List Box 1";               //give it a label
            list = "1\n2\n3\n4";                //make a list
            height = 5;                         //define height
          }
```

```
: list_box {                          //define a list box
  key = "lb2";                        //give it a name
  label = "List Box 2";               //give it a label
  list = "A\nB\nC\nD";                //make a list
  height = 5;                         //define height
}

}                                     //end row

ok_cancel ;                           //predefined OK/Cancel button

}                                     //end dialog
```

children_fixed_width



Children_Fixed_Width Not Set.



Children_Fixed_Width Set to True.

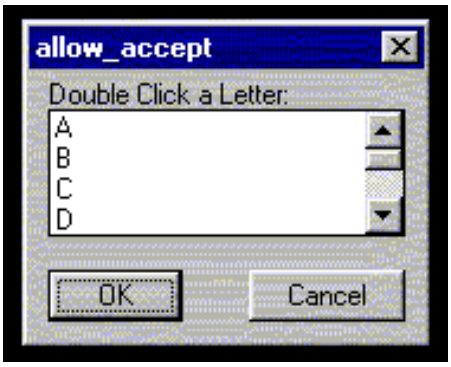
This attribute specifies the default width for all the tiles in a cluster.
It does not override a child's width attribute if it is specified. Possible values are: *true* or *false*. (default: *false*)

DCL Code Sample

```
lisp49m : dialog {                              //dialog name
            label = "children_fixed_width";     //give it a label

          : column {                            //define column
            children_fixed_width = true;        //fix the width

          : edit_box {                          //define an edit box
            key = "eb1";                        //give it a name
            label = "Edit Box 1";               //give it a label
            width = 10;                         //give it a width
            value = "Edit Box 1";               //fill it
          }                                     //end edit box

          : edit_box {                          //define an edit box
            key = "eb2";                        //give it a name
            label = "Edit Box 2";               //give it a label
            width = 10;                         //give it a width
            value = "Edit Box 2";               //fill it
          }                                     //end edit box

          }                                     //end column

          spacer;                               //a spacer
```

```
ok_cancel_help ;                    //predefined OK/Cancel/Help button
}                                   //end dialog
```

allow_accept



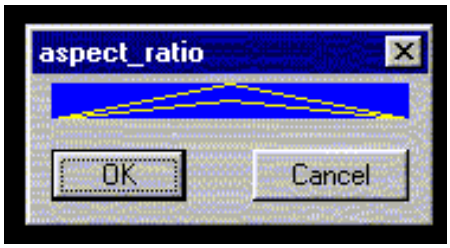
Allow_Accept.

This attribute specifies whether a particular tile can be double-clicked.
If *true* and the user double-clicks the tile, the tile who's key is "accept" is selected. (Normally the O.K. tile.)
This attribute defaults to *false*.

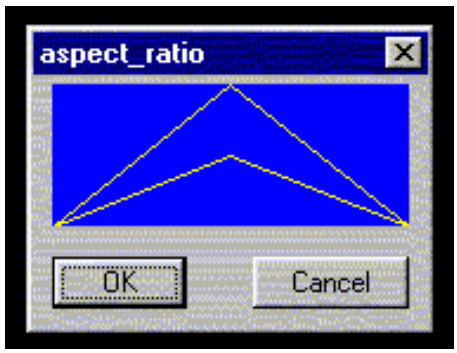
DCL Code Sample

```
lisp49n : dialog {                                  //dialog name
            label = "allow_accept";                 //give it a label

          : list_box {                              //define a list box
            key = "lb1";                            //give it a name
            label = "Double Click a Letter:";       //give it a label
            list = "A\nB\nC\nD\nE\nF\nG\nH\nI";      //make a list
            height = 5;                             //define height
            fixed_height = true;                    //fix the height
            allow_accept = true;                    //allow double click
          }                                         //end list box

          spacer;                                   //add a space

          ok_cancel ;                               //predefined OK/Cancel button

          }                                         //end dialog
```

aspect_ratio



Aspect_Ratio set to 0.5.



Aspect_Ratio set to 2.0.

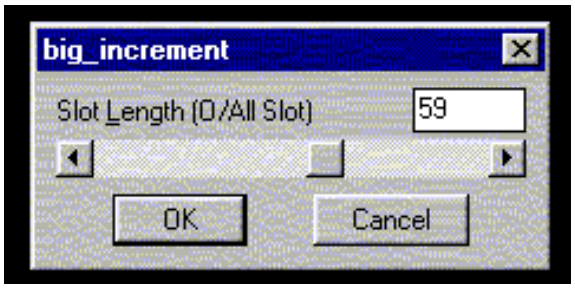
This attribute specifies the ratio of the width of an image to it's height. (width divided by height.)
If zero (0.0), the tile is fitted to the size of the image.
Possible values are floating-point values.

default: *none*.

DCL Code Sample

```
lisp49o : dialog {                          //dialog name
            label = "aspect_ratio" ;        //give it a label

          : image {                         //define image tile
            key = "im1" ;                   //give it a name
            width = 5.0 ;                   //set width
            aspect_ratio = 2.0;             //set the ratio
            alignment = centered;           //center the image
          }                                 //end image

          spacer;                           //add a space

          ok_cancel ;                       //predefined OK/Cancel button

          }                                 //end dialog
```

big_increment



Big Increment.

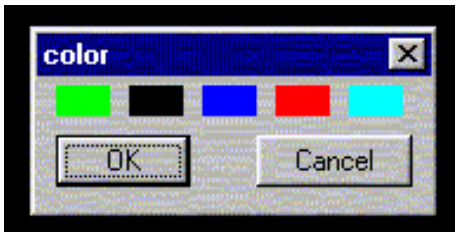
This attribute specifies the value that the slider will increase or decrease by, if you click the slider bar. This value must be between the *min_value*, and the *max_value*.

Default value: One-tenth of the total range.

DCL Code Sample

```
lisp49p : dialog {                                  //dialog name
          label = "big_increment" ;                 //give it a label

        : edit_box {                                //define edit box
         key = "eb1" ;                              //give it a name
         label = "Slot &Length (O/All Slot)" ;      //give it a label
         edit_width = 6 ;                           //6 characters only
        }                                           //end edit box

        : slider {                                  //define slider
        key = "myslider" ;                          //give it a name
        max_value = 100;                            //upper value
        min_value = 0;                              //lower value
        value = "50";                               //initial value
        big_increment = 5;                          //define big increment
        }                                           //end slider

        ok_cancel ;                                 //predefined OK/Cancel button

        }                                           //end dialog
```

color



Color.

This attribute specifies the background (fill) colour of an image.
Possible values are an integer or reserved word (default: 7) specified as an AutoCAD colour number or as one of the symbolic names as shown in the following table:

dialog_line	Current dialog box line colour
dialog_foreground	Current dialog box foreground colour
dialog_background	Current dialog box background colour
graphics_background	Current background of the AutoCAD graphics screen. (Usually equivalent to 0)
black	AutoCAD colour = 0 (black) (appears light on a black background)
red	AutoCAD colour = 1 (red)
yellow	AutoCAD colour = 2 (yellow)
green	AutoCAD colour = 3 (green)
cyan	AutoCAD colour = 4 (cyan)
blue	AutoCAD colour = 5 (blue)
magenta	AutoCAD colour = 6 (magenta)
white	AutoCAD colour = 7 (white)
graphics_foreground	Appears black on a light background

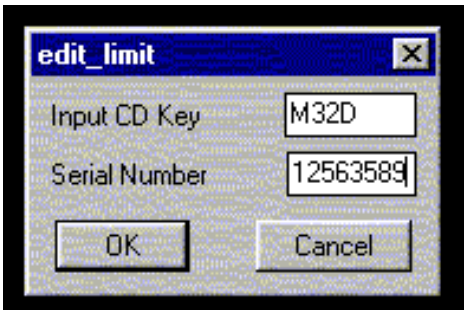
Note: If an image tile containing slides is blank when you first display it, try changing it's *color* attribute to *graphics_background* or *graphics_foreground.*

DCL Code Sample

```
lisp49q : dialog {                              //dialog name
            label = "color" ;                   //give it a label

          : row {                               //begin row
            children_fixed_height = true;       //fix the height
            children_alignment = centered;

          : image {                             //define image tile
            key = "im1" ;                       //give it a name
            height = 1.0 ;                      //and a height
            width = 2.0 ;                       //and now a width
            color = green;                      //set the colour
          }                                     //end image
```

```
: image {                          //define image tile
  key = "im2" ;                    //give it a name
  height = 1.0 ;                   //and a height
  width = 2.0 ;                    //and now a width
  color = dialog_line;             //set the colour
}                                  //end image

: image {                          //define image tile
  key = "im3" ;                    //give it a name
  height = 1.0 ;                   //and a height
  width = 2.0 ;                    //and now a width
  color = blue;                    //set the colour
}                                  //end image

: image {                          //define image tile
  key = "im4" ;                    //give it a name
  height = 1.0 ;                   //and a height
  width = 2.0 ;                    //and now a width
  color = red;                     //set the colour
}                                  //end image

: image {                          //define image tile
  key = "im5" ;                    //give it a name
  height = 1.0 ;                   //and a height
  width = 2.0 ;                    //and now a width
  color = cyan;                    //set the colour
}                                  //end image

}                                  //end row


ok_cancel ;                        //predefined OK/Cancel button

}                                  //end dialog
```

edit_limit



Edit Limit.

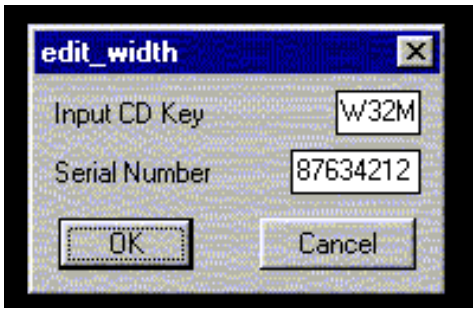
This attribute specifies the maximum number of characters a user is allowed to enter into an edit box. Possible value is an integer.

Default: 132.

DCL Code Sample

```
lisp49r : dialog {                          //dialog name
            label = "edit_limit" ;          //give it a label

          : edit_box {                      //define edit box
            key = "eb1";                    //give it a name
            label = "Input CD Key  ";       //give it a label
            edit_limit = 4;                 //limit characters
          }                                 //end edit box

          : edit_box {                      //define edit box
            key = "eb2";                    //give it a name
            label = "Serial Number";        //give it a label
            edit_limit = 8;                 //limit characters
          }                                 //end edit box

          spacer;                           //add a space

          ok_cancel ;                       //predefined OK/Cancel button

          }                                 //end dialog
```

edit_width



Edit Width.

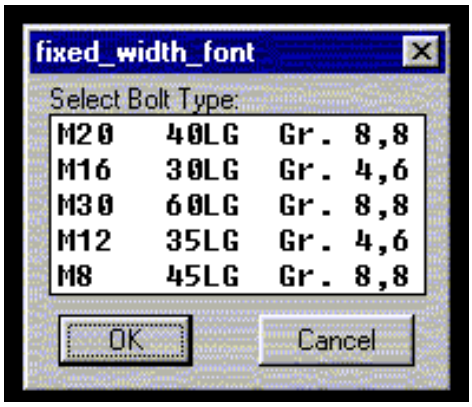
This attribute specifies the width in character units of an edit box.
Possible values are an integer or a real number.

Default: ***None.***

DCL Code Sample

```
lisp49s : dialog {                          //dialog name
            label = "edit_width" ;          //give it a label

          : edit_box {                      //define edit box
            key = "eb1";                    //give it a name
            label = "Input CD Key  ";       //give it a label
            edit_limit = 4;                 //limit characters
            edit_width = 4;                 //fix the width
          }                                 //end edit box

          : edit_box {                      //define edit box
            key = "eb2";                    //give it a name
            label = "Serial Number";        //give it a label
            edit_limit = 8;                 //limit characters
            edit_width = 8;                 //fix the width
          }                                 //end edit box

          spacer;                           //add a space

          ok_cancel ;                       //predefined OK/Cancel button

          }                                 //end dialog
```

fixed_width_font



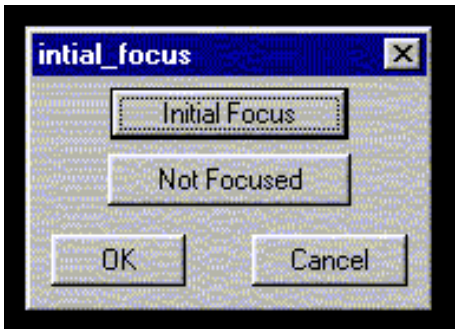
Dialogue with Fixed Width Font.

This attribute specifies whether a 'list_box' or 'popup_list' will display text in a fixed pitch font.
This allows for easier spacing and tab alignment of columns.
This attribute is valid only for Windows/NT.

DCL Code Sample

```
lisp49ai : dialog {                              //dialog name
           label = "fixed_width_font" ;          //give it a label

         : list_box {                            //define a list box
           key = "lb1";                          //give it a name
           label = "Select Bolt Type:";          //give it a label
           allow_accept = true;                  //allow double clicking
           tabs = "8 16 24";                     //set tabs
           list = "M20\t40LG\tGr. 8,8\n
                   M16\t30LG\tGr. 4,6\n
                   M30\t60LG\tGr. 8,8\n
                   M12\t35LG\tGr. 4,6\n
                   M8\t45LG\tGr. 8,8";           //define the list
           height = 6;                           //give it a height
           fixed_height = true;                  //fix the height
           width = 20;                           //give it a width
           fixed_width_font = true;              //fix the font width
         }                                       //end list box

         ok_cancel ;                             //predefined OK/Cancel button

         }                                       //end dialog
```

initial_focus



Initial Focus.

This attribute specifies the the key of the tile within the dialogue box that receives the initial focus when the dialogue is started.
Possible value is a quoted string.

Default: ***None.***

DCL Code Sample

```
lisp49t : dialog {                          //dialog name
            label = "intial_focus" ;        //give it a label
            initial_focus = "btn1";         //set focus

          : button {                        //define button
            key = "btn1";                   //give it a name
            label = "Initial Focus";        //give it a label
            alignment = centered;           //center it
            fixed_width = true;             //fix the width
          }                                 //end button

          : button {                        //define button
            key = "btn2";                   //give it a name
            label = "Not Focused";          //give it a label
            alignment = centered;           //center it
            fixed_width = true;             //fix the width
          }                                 //end button

          spacer;                           //add a space

          ok_cancel ;                       //predefined OK/Cancel button

          }                                 //end dialog
```

is_cancel

Is_Cancel.

This attribute specifies whether the button is selected when the user presses the cancel key (Esc or Ctrl+C). Possible values are *true* or *false*.
Default: *false*.
Only one button in a dialogue box can have the 'is_cancel' attribute set to *true*.

DCL Code Sample

```
lisp49v : dialog {                          //dialog name
          label = "is_cancel" ;             //give it a label

        : button {                          //define button
          label = "Press Escape";           //give it a label
          key = "cancel";                   //give it a name
          fixed_width = true;               //fix the width
          is_cancel = true;                 //make it the cancel key
        }                                   //end button

        }                                   //end dialog
```

is_default

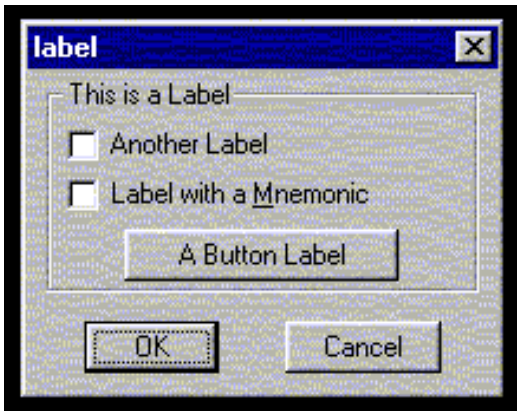
Is_Default.

This attribute specifies whether the button is the default button selected ("pushed") when the user presses the accept key. (normally "Enter").
Possible values are *true* or *false*.
Default: *false*.
Only one button in a dialogue box can have the 'is_default' attribute set to *true*.

DCL Code Sample

```
lisp49w : dialog {                          //dialog name
          label = "is_default" ;            //give it a label

        : button {                          //define button
          label = "Press Enter";            //give it a label
          key = "accept";                   //give it a name
          fixed_width = true;               //fix the width
          is_default = true;                //make it the default key
        }                                   //end button

        }                                   //end dialog
```

label



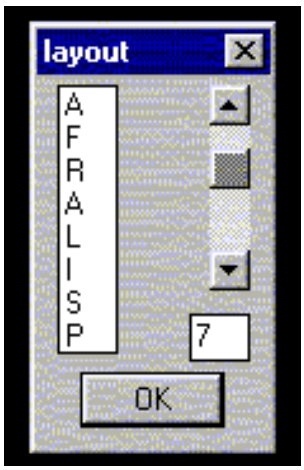
A couple of Labels.

This attribute specifies the the text displayed within tile. Possible value is a quoted string.
Default: A blank string, " ".
The label can specify the mnemonic for a tile by preceding one of the letters of the label with the ampersand character. (&)

DCL Code Sample

```
lisp49x : dialog {                              //dialog name
            label = "label" ;                   //give it a label

          : boxed_column {                      //define a boxed column
            label = "This is a Label";          //give it a label

          : toggle {                            //define a toggle
            label = "Another Label";            //give it a label
            key = "tog1";                       //give it a name
          }                                     //end toggle

          : toggle {                            //define a toggle
            label = "Label with a &Mnemonic";   //give it a label
            key = "tog2";                       //give it a name
          }                                     //end toggle

          : button {                            //define button
            label="A Button Label";             //give it a label
            fixed_width=true;                   //fix the width
            alignment=centered;                 //align right
          }                                     //end button

          }                                     //end boxed column

          spacer;                               //add a space

          ok_cancel ;                           //predefined OK/Cancel button

          }                                     //end dialog
```

layout



Vertical Layout.

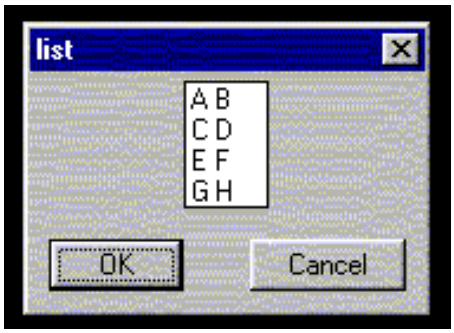
This attribute specifies the orientation of a slider. Possible values are *horizontal* or *vertical*. For horizontal sliders, the value increases from left to right. For vertical sliders, it increases from bottom to top. Default : *horizontal*.

DCL Code Sample

```
lisp49y : dialog {                                  //dialog name
          label = "layout" ;                        //give it a label

        : row {                                     //define row

        : list_box {                                //define list box
          key = "lb1";                              //give it a name
          list = "A\nF\nR\nA\nL\nI\nS\nP";           //the list
          height = 8;                               //give it a height
          fixed_height = true;                      //fix the height
          width = 4;                                //give it a width
          fixed_width = true;                       //fix the width
        }                                           //end list box

        : column {                                  //define column

        : slider {                                  //define slider
        layout = vertical;                          //make it vertical
        key = "myslider" ;                          //give it a name
        max_value = 10;                             //upper value
        min_value = 0;                              //lower value
        value = "5";                                //initial value
        big_increment = 2;                          //define big increment
```

```
alignment = right;                    //align it right
}                                     //end slider

: edit_box {                          //define edit box
 key = "eb1" ;                        //give it a name
 edit_width = 2 ;                     //2 characters only
}                                     //end edit box

}                                     //end column

}                                     //end row

ok_only ;                             //predefined OK button

}                                     //end dialog
```

list



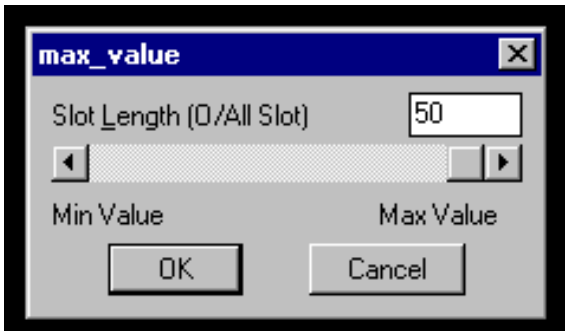
List with Tabs.

This attribute specifies the initial set of lines (choices) to be placed in the 'popup_list' or 'list_box'. Possible value is a quoted string.
Default : None.
Lines are separated by a new line symbol (\n). Tab characters (\t) can occur within each line.

DCL Code Sample

```
lisp49z : dialog {                              //dialog name
            label = "list" ;                    //give it a label

          : list_box {                          //define list box
            key = "lb1";                        //give it a name
            list = "A\tB\nC\tD\nE\tF\nG\tH";     //the list
            tabs = "2 4";                       //set tabs
            height = 5;                         //give it a height
            width = 6;                          //give it a width
            fixed_width = true;                 //fix the width
            fixed_height = true;                //fix the height
            alignment = centered;               //center it
          }                                     //end list box

          spacer ;                              //add a space

          ok_cancel ;                           //predefined OK/Cancel button

          }                                     //end dialog
```

max_value



Maximum Value = 50.

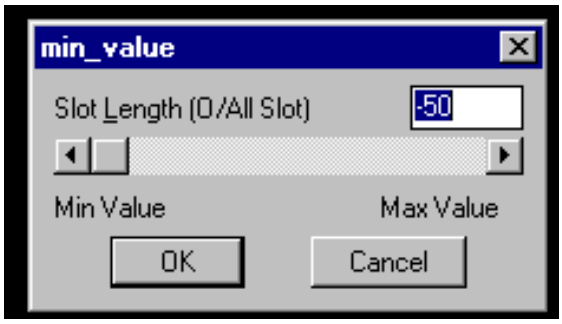
This attribute specifies the upper range of values that a slider returns.
Default: 10000.
This value must be no greater than 32767.

DCL Code Sample

```
lisp49aa : dialog {                              //dialog name
          label = "max_value" ;                  //give it a label

        : edit_box {                             //define edit box
         key = "eb1" ;                           //give it a name
         label = "Slot &Length (O/All Slot)" ;   //give it a label
         edit_width = 6 ;                        //6 characters only
        }                                        //end edit box

        : slider {                               //define slider
        key = "myslider" ;                       //give it a name
        max_value = 50;                          //upper value
        min_value = -50;                         //lower value
        value = "50";                            //initial value
        big_increment = 5;                       //define big increment
        }                                        //end slider

        : row {                                  //define row

        : text {                                 //define text
          value = "Min Value";                   //give it a value
        }                                        //end text

        : text {                                 //define text
          value = " ";                           //give it a value
        }                                        //end text

        : text {                                 //define text
          value = "Max Value";                   //give it a value
        }                                        //end text
```

```
}                                       //end row
ok_cancel ;                             //predefined OK/Cancel button
}                                       //end dialog
```

min_value



Minimum Value = -50.

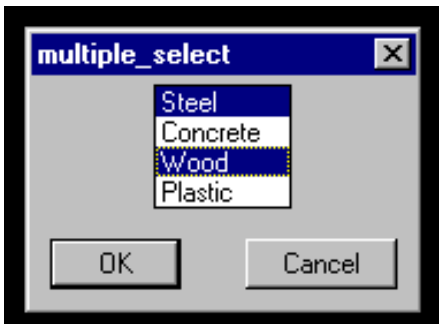
This attribute specifies the lower range of values that a slider returns.
Default: 0.
This value must be no less than -32767. The 'min_value' can be greater than the 'max_value'. This reverses the order in which these values appear on the screen. (Platform dependent).

DCL Code Sample

```
lisp49ab : dialog {                                  //dialog name
           label = "min_value" ;                     //give it a label

         : edit_box {                                //define edit box
          key = "eb1" ;                              //give it a name
          label = "Slot &Length (0/All Slot)" ;      //give it a label
          edit_width = 6 ;                           //6 characters only
         }                                           //end edit box

         : slider {                                  //define slider
         key = "myslider" ;                          //give it a name
         max_value = 50;                             //upper value
         min_value = -50;                            //lower value
         value = "50";                               //initial value
         big_increment = 5;                          //define big increment
         }                                           //end slider

         : row {                                     //define row

         : text {                                    //define text
           value = "Min Value";                      //give it a value
         }                                           //end text

         : text {                                    //define text
           value = " ";                              //give it a value
         }                                           //end text

         : text {                                    //define text
           value = "Max Value";                      //give it a value
         }                                           //end text
```

```
}                                       //end row
ok_cancel ;                             //predefined OK/Cancel button
}                                       //end dialog
```

multiple_select



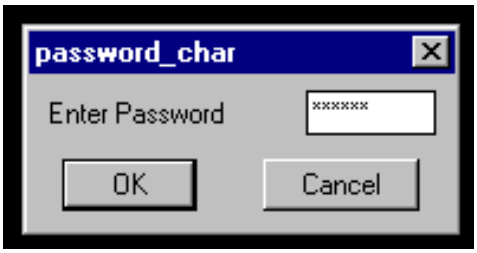
Multiple Select.

This attribute specifies whether multiple items in a 'list_box' can be selected (and highlighted) at the same time. Possible values are *true* or *false*.
Default : *false*.
The method of selected multiple items is platform dependent. (On Windows you hold down the 'Ctrl' or 'Shift' key whilst selecting.)

DCL Code Sample

```
lisp49ac : dialog {                                    //dialog name
           label = "multiple_select" ;                 //give it a label

         : list_box {                                  //define list box
           key = "lb1";                                //give it a name
           list = "Steel\nConcrete\nWood\nPlastic";//the list
           height = 5;                                 //give it a height
           width = 10;                                 //give it a width
           fixed_width = true;                         //fix the width
           fixed_height = true;                        //fix the height
           alignment = centered;                       //center it
           multiple_select = true;                     //allow multiple select
         }                                             //end list box

         spacer ;                                      //add a space

         ok_cancel ;                                   //predefined OK/Cancel button

         }                                             //end dialog
```

password_char



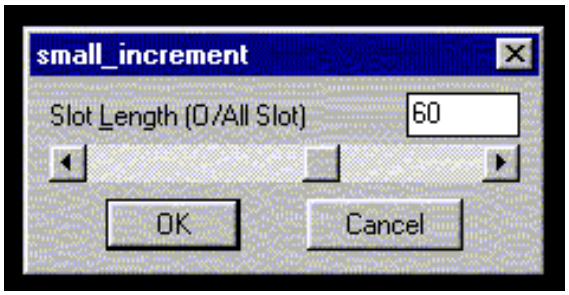
Password Character.

This attribute specifies the character to be used as a password character. If the 'password_char' is specified, that character is displayed in the 'edit_box' instead of the characters entered by the user. The use of this attribute has no effect on the retrieval of the value entered by the user. It alters only the display of the characters in the 'edit_box'.

DCL Code Sample

```
lisp49ad : dialog {                          //dialog name
          label = "password_char" ;          //give it a label

        : edit_box {                         //define edit box
          label = "Enter Password";          //give it a label
          key = "eb1";                       //give it a name
          password_char = "*";               //password character
          width = 6;                         //give it a width
          edit_limit = 6;                    //limit to six characters
          fixed_width = true;                //fix the width
        }                                    //end edit box

        spacer ;                             //add a space

        ok_cancel ;                          //predefined OK/Cancel button

        }                                    //end dialog
```

small_increment



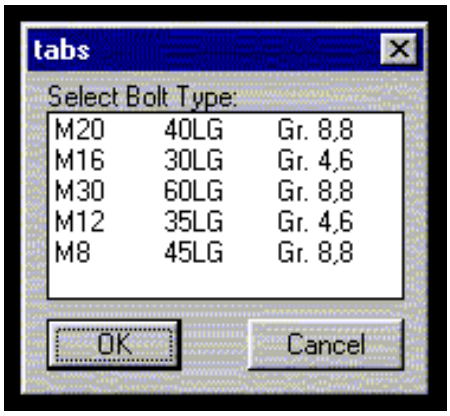
Small Increment.

This attribute specifies the value that the slider will increase or decrease by, if you click the slider bar arrow controls located at each end of the slider.
This value must be between the *min_value*, and the *max_value*.
Default value: One-hundredth of the total range.

DCL Code Sample

```
lisp49ae : dialog {                              //dialog name
          label = "small_increment" ;            //give it a label

        : edit_box {                             //define edit box
         key = "eb1" ;                           //give it a name
         label = "Slot &Length (O/All Slot)" ;   //give it a label
         edit_width = 6 ;                        //6 characters only
        }                                        //end edit box

        : slider {                               //define slider
        key = "myslider" ;                       //give it a name
        max_value = 100;                         //upper value
        min_value = 0;                           //lower value
        value = "50";                            //initial value
        small_increment = 2;                     //define small increment
        big_increment = 5;                       //define big increment
        }                                        //end slider

        ok_cancel ;                              //predefined OK/Cancel button

        }                                        //end dialog
```

tabs



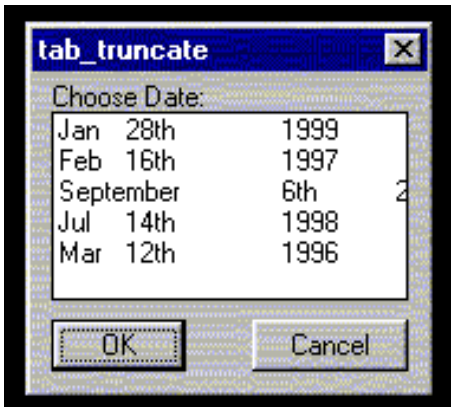
List Box with Tabs.

This attribute specifies the placement of tabs in character width units.
Possible value is a quoted string containing integers or floating-point numbers separated by spaces.
Default = *None*.
These values are used for vertically aligning columns of text in a 'popup_list' or 'list_box'.

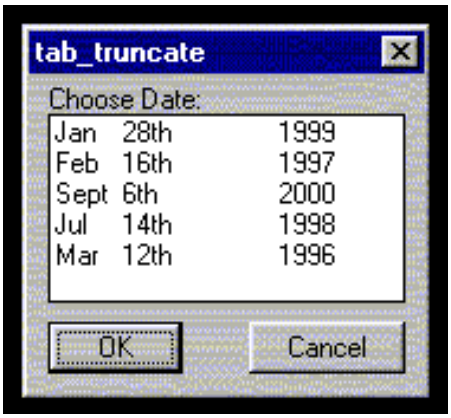
DCL Code Sample

```
lisp49af : dialog {                              //dialog name
          label = "tabs" ;                       //give it a label

        : list_box {                             //define a list box
          key = "lb1";                           //give it a name
          label = "Select Bolt Type:";           //give it a label
          allow_accept = true;                   //allow double clicking
          tabs = "8 16 24";                      //set tabs
          list = "M20\t40LG\tGr. 8,8\n
                  M16\t30LG\tGr. 4,6\n
                  M30\t60LG\tGr. 8,8\n
                  M12\t35LG\tGr. 4,6\n
                  M8\t45LG\tGr. 8,8";            //define the list
          height = 6;                            //give it a height
          fixed_height = true;                   //fix the height
        }                                        //end list box

        ok_cancel ;                              //predefined OK/Cancel button

        }                                        //end dialog
```

tab_truncate



List Box with Tabs NOT Truncated.



List Box with Tabs Truncated.

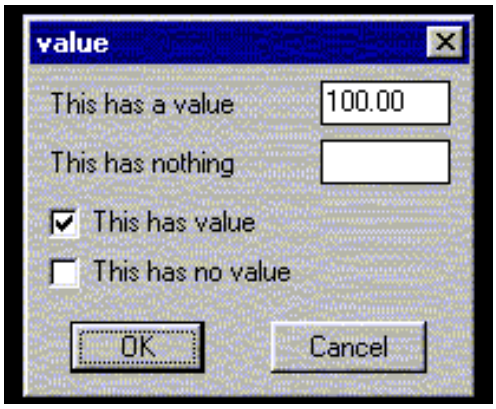
This attribute specifies whether the text in a 'list_box' or 'popup_list' is truncated if it is larger than the associated tab stop.
Possible values are *true* or *false*.
Default : *false*.

DCL Code Sample

```
lisp49ag : dialog {                              //dialog name
          label = "tab_truncate" ;               //give it a label

         : list_box {                            //define a list box
           key = "lb1";                          //give it a name
           label = "Choose Date:";               //give it a label
           allow_accept = true;                  //allow double clicking
           tabs = "5 16 24";                     //set tabs
           list = "Jan\t28th\t1999\n
                   Feb\t16th\t1997\n
                   September\t6th\t2000\n
                   Jul\t14th\t1998\n
                   Mar\t12th\t1996";             //define list
           tab_truncate = true;                  //truncate
```

```
  height = 6;                          //give it a height
  fixed_height = true;                 //fix the height
}                                      //end list box

ok_cancel ;                            //predefined OK/Cancel button

}                                      //end dialog
```

value



Value.

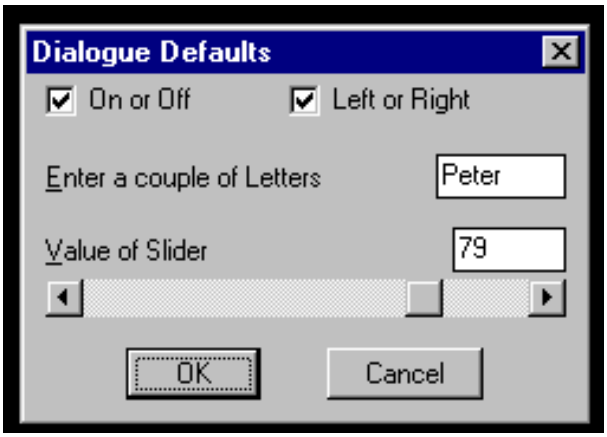
This attribute specifies the initial value of a tile. Possible value is a quoted string. The meaning of a tile's value varies depending on the kind of tile. The value of a tile can change at run-time, with user input or 'set_tile' calls.

DCL Code Sample

```
lisp49ah : dialog {                          //dialog name
           label = "value" ;                 //give it a label

         : edit_box {                        //define edit box
           key = "eb1";                      //give it a name
           value = "100.00";                 //give it a value
           label = "This has a value";       //give it a label
           width = 6;                        //give it a width
           fixed_width = true;               //fix the width
         }                                   //end edit box

         : edit_box {                        //define edit box
           key = "eb2";                      //give it a name
           label = "This has nothing";       //give it a label
           width = 6;                        //give it a width
           fixed_width = true;               //fix the width
         }                                   //end edit box

         spacer;                             //define a space

         : toggle {                          //define a toggle
           key = "tog1";                     //give it a name
           value = "1";                      //give it a value
           label = "This has value";         //give it a label
         }                                   //end toggle

         : toggle {                          //define a toggle
           key = "tog2";                     //give it a name
           label = "This has no value";      //give it a label
```

```
}                                   //end toggle
spacer;                             //define a space
ok_cancel ;                         //predefined OK/Cancel button
}                                   //end dialog
```

Dialog Box Default Data



When a user works with a dialog box he enters the required data and then exits the dialog box. On returning to the dialog box, the data contained in the box should be updated to reflect the users previous entries. Most users tend to re-use the same set of data time after time, and it can be very frustrating to have to retype the same data every time you re-open a dialog box.

To create defaults in a dialog box is a 5 step process :

- Retrieve the global variables containing the default data.
- If they do not exist, create some default data.
- Display the default data to the user.
- Allow the user to change the data.
- When the user exits the dialog box, store this data as defaults.

The following application will show you how to do this :

DCL Coding:

```
test1 : dialog {                                    //dialog name
          label = "Dialog Defaults" ;               //give it a label

        : row {                                     //define row

        : toggle {                                  //define toggle
          key = "tog1";                             //give it a name
          label = "On or Off";                      //give it a label
        }                                           //end toggle

        : toggle {                                  //define toggle
          key = "tog2";                             //give it a name
          label = "Left or Right";                  //give it a label
        }                                           //end toggle

        }                                           //end row

        spacer;                                     //put in a space

        : edit_box {                                //define edit box
         key = "eb2" ;                              //give it a name
```

```
       label = "&Enter a couple of Letters" ; //give it a label
      }                                       //end edit box

      spacer;                                 //put in a space

      : edit_box {                            //define edit box
       key = "eb1" ;                          //give it a name
       label = "&Value of Slider" ;           //give it a label
       edit_width = 6 ;                       //6 characters only
      }                                       //end edit box

      : slider {                              //define slider
      key = "myslider" ;                      //give it a name
      max_value = 100;                        //upper value
      min_value = 0;                          //lower value
      value = "50";                           //initial value
      small_increment = 2;                    //define small increment
      big_increment = 5;                      //define big increment

      }                                       //end slider

      spacer;                                 //put in a space

      ok_cancel ;                             //predefined OK/Cancel button

      }                                       //end dialog
```

AutoLISP Coding:

```
(defun C:test1 ()
;define function

;;;*****************************************************
;;;This section sets up the dialog box default settings.

  (if (not lngth)
  ;if no default

        (setq lngth "50.0")
        ;set a default value

  );if
  ;slider and slider edit box value

  (if (not letters)
  ;if no default

        (setq letters "ABC")
        ;set default value
  );if

  (if (not toggle1)
  ;if no default
```

```
        (setq toggle1 "1")
        ;set default value

  );if

  (if (not toggle2)
  ;if no default

        (setq toggle2 "0")
        ;set default value

  );if
;;;*************************************************************

  (setq dcl_id (load_dialog "test1.dcl"))
  ;load dialog

  (if (not (new_dialog "test1" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if
;;;*****************************************************************
;;;This section sets the dialog box default values

  (set_tile "tog1" toggle1)
  ;set toggle

  (set_tile "tog2" toggle2)
  ;set toggle

  (set_tile "eb2" letters)
  ;put data into edit box

  (set_tile "eb1" lngth)
  ;put data into edit box

  (set_tile "myslider" lngth)
  ;set the slider

;;;*****************************************************************
;;;This section retrieves the dialog box values

  (action_tile "myslider"
  ;if user moves slider

         "(slider_action $value $reason)")
         ;pass arguments to slider_action

  (action_tile "eb1"
  ;if user enters slot length
```

```
       "(ebox_action $value $reason)")
       ;pass arguments to ebox_action

(defun slider_action (val why)
;define function

    (if (or (= why 2) (= why 1))
    ;check values

    (set_tile "eb1" val)))
    ;update edit box

(defun ebox_action (val why)
;define function

    (if (or (= why 2) (= why 1))
    ;check values

    (set_tile "myslider" val)))
    ;update slider

 (action_tile "tog1"
 ;if toggle 1 selected

    "(setq toggle1 $value)"
     ;get the value

 );action_tile

 (action_tile "tog2"
 ;if toggle 2 selected

    "(setq toggle2 $value)"
     ;get the value

 );action_tile

 (action_tile
    "accept"
    ;if O.K. pressed

    "(progn
     ;do the following

    (setq lngth (get_tile \"eb1\"))
    ;get the edit box value

    (setq letters (get_tile \"eb2\"))
    ;get the slider value

     (done_dialog) (setq userclick T))"
     ;close dialog, set flag

 );action tile
```

```
    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog and clear flag

    );action_tile
;;*****************************************************************

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

   (if userclick
   ;check O.K. was selected

      (alert (strcat "You Selected: " lngth " and " letters))
      ;display the Data

  );if userclick

 (princ)

);defun

(princ)
```

Now load and run this application in AutoCAD. Enter some new data and exit the dialog box. Run the application again. You will find that the dialog box has 'remembered' your entries.

Now this is fine, but if you start a new drawing, or exit AutoCAD these values will be lost. Let's re-write this routine to "permanently" remember these values.
To do this we will make use of Application Data. Here's how :

DCL Coding:

```
test2 : dialog {                                  //dialog name
          label = "Dialog Defaults" ;             //give it a label

        : row {                                   //define row

        : toggle {                                //define toggle
          key = "tog1";                           //give it a name
          label = "On or Off";                    //give it a label
        }                                         //end toggle

        : toggle {                                //define toggle
          key = "tog2";                           //give it a name
          label = "Left or Right";                //give it a label
```

```
    }                                          //end toggle

    }                                          //end row

    spacer;                                    //put in a space

    : edit_box {                               //define edit box
     key = "eb2" ;                             //give it a name
     label = "&Enter a couple of Letters" ;    //give it a label
    }                                          //end edit box

    spacer;                                    //put in a space

    : edit_box {                               //define edit box
     key = "eb1" ;                             //give it a name
     label = "&Value of Slider" ;              //give it a label
     edit_width = 6 ;                          //6 characters only
    }                                          //end edit box

    : slider {                                 //define slider
    key = "myslider" ;                         //give it a name
    max_value = 100;                           //upper value
    min_value = 0;                             //lower value
    value = "50";                              //initial value
    small_increment = 2;                       //define small increment
    big_increment = 5;                         //define big increment

    }                                          //end slider

    spacer;                                    //put in a space

    ok_cancel ;                                //predefined OK/Cancel button

    }                                          //end dialog
```

AutoLISP Coding:

```
(defun C:test2 ()
;define function

;;;*****************************************************
;;;This section retrieves the default data from the
;;;ACADR14.CFG file.


   (setq lngth (getcfg "AppData/CadKen/default1"))
   ;get the apllication data

      (if (= lngth nil)
     ;if it is nil

       (progn
       ;do the following
```

```
      (setcfg "AppData/CadKen/default1" "50.0")
      ;set the default data

      (setq lngth (getcfg "AppData/CadKen/default1"))
      ;retrieve the default data

     );progn

  );if

(setq letters (getcfg "AppData/CadKen/default2"))
;get the aplication data

   (if (= letters nil)
  ;if it is nil

     (progn
     ;do the following

      (setcfg "AppData/CadKen/default2" "ABC")
      ;set the default data

      (setq letters (getcfg "AppData/CadKen/default2"))
      ;retrieve the default data

     );progn

  );if


(setq toggle1 (getcfg "AppData/CadKen/default3"))
;get the apllication data

   (if (= toggle1 nil)
  ;if it is nil

     (progn
     ;do the following

      (setcfg "AppData/CadKen/default3" "1")
      ;set the default data

      (setq toggle1 (getcfg "AppData/CadKen/default3"))
      ;retrieve the default data

     );progn

  );if

(setq toggle2 (getcfg "AppData/CadKen/default4"))
;get the apllication data

   (if (= toggle2 nil)
  ;if it is nil
```

```
        (progn
        ;do the following

         (setcfg "AppData/CadKen/default4" "0")
         ;set the default data

         (setq toggle2 (getcfg "AppData/CadKen/default4"))
         ;retrieve the default data

        );progn

      );if

;;;*************************************************************

  (setq dcl_id (load_dialog "test2.dcl"))
  ;load dialog

  (if (not (new_dialog "test2" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if
;;;*****************************************************************
;;;This section sets the dialog box default values

  (set_tile "tog1" toggle1)
  ;set toggle

  (set_tile "tog2" toggle2)
  ;set toggle

  (set_tile "eb2" letters)
  ;put data into edit box

  (set_tile "eb1" lngth)
  ;put data into edit box

  (set_tile "myslider" lngth)
  ;set the slider

;;;*****************************************************************
;;;This section retrieves the dialog box values

  (action_tile "myslider"
  ;if user moves slider

         "(slider_action $value $reason)")
         ;pass arguments to slider_action

  (action_tile "eb1"
  ;if user enters slot length
```

```
       "(ebox_action $value $reason)")
       ;pass arguments to ebox_action

(defun slider_action (val why)
;define function

    (if (or (= why 2) (= why 1))
    ;check values

    (set_tile "eb1" val)))
    ;update edit box

(defun ebox_action (val why)
;define function

    (if (or (= why 2) (= why 1))
    ;check values

    (set_tile "myslider" val)))
    ;update slider

 (action_tile "tog1"
 ;if toggle 1 selected

    "(setq toggle1 $value)"
     ;get the value

 );action_tile

 (action_tile "tog2"
 ;if toggle 2 selected

    "(setq toggle2 $value)"
     ;get the value

 );action_tile

 (action_tile
    "accept"
    ;if O.K. pressed

    "(progn
     ;do the following

    (setq lngth (get_tile \"eb1\"))
    ;get the slider value

    (setq letters (get_tile \"eb2\"))
    ;get the edit box value

     (done_dialog) (setq userclick T))"
     ;close dialog, set flag

 );action tile
```

```
    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog and clear flag

    );action_tile
;;*****************************************************************

  (start_dialog)
  ;start dialog

  (unload_dialog dcl_id)
  ;unload

   (if userclick
   ;check O.K. was selected
;;;*****************************************************************
;;;This section stores the data into the AACDR14.CFG file and displays
;;;them to the user.

    (progn

      (setcfg "AppData/CadKen/default1" lngth)
      ;store the data

      (setcfg "AppData/CadKen/default2" letters)
      ;store the data

      (setcfg "AppData/CadKen/default3" toggle1)
      ;store the data

      (setcfg "AppData/CadKen/default4" toggle2)
      ;store the data

      (alert (strcat "You Entered: " lngth " and " letters))
      ;display the Data

    );progn

  );if userclick

 (princ)

);defun

(princ)
```

Now load and run this application. Again, add some new values and exit the dialog.
Now exit AutoCAD. Re-enter AutoCAD and load and run the application again.
Your values have been retained....

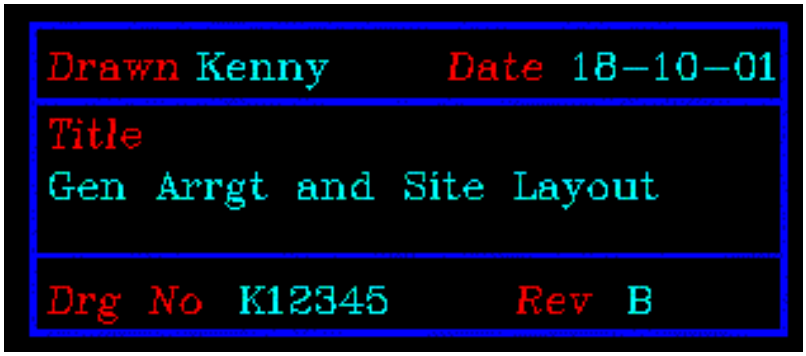
Attributes and Dialog Boxes

When you want to edit attributes in AutoCAD most of us use the "Attedit" command. Firstly, we must select the attribute we would like to edit. Then the "Edit Attribute" dialog box appears which allows us to add or change the values of our attribute. Personally, I think this dialog leaves a lot to be desired. You cannot customise it in any way, and it displays all attributes whether you want them or not. As well, if you have a lot of attributes you need to page your way through numerous dialogs before reaching the attribute you want to edit.

In this tutorial we are going to have a look at extracting attribute data from a block, displaying the data in a custom dialog box, and then updating the attribute data on exit. Right, what do we need to do?

- Find the block containing the attribute data. (Why select it when we can get AutoCAD to find it for us.)
- Extract the attribute data and display it in a dialog box.
- Allow the user to change the data if he so wishes.
- Update the attribute data with the new information entered into the dialog box.

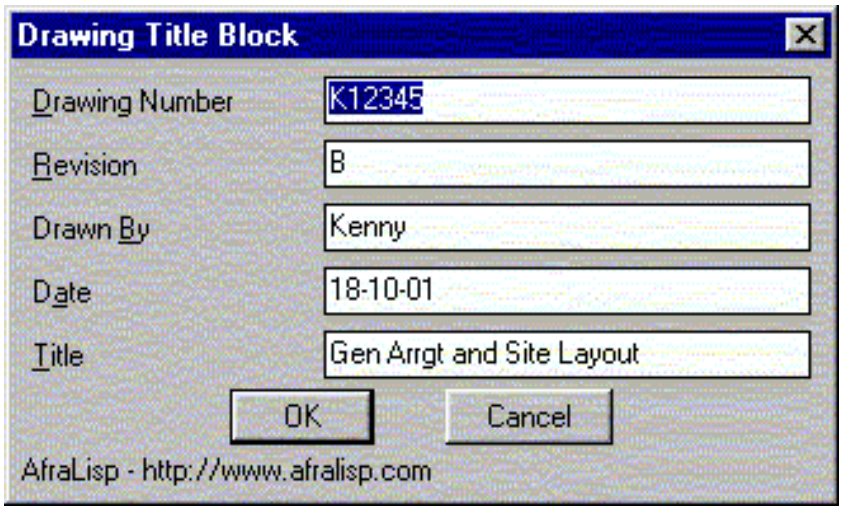
O.K. fire up AutoCAD and open the drawing Attab.dwg.



Alright, I admit that it's not much of a title block, but it's enough to give you the general idea.

Now, at the command prompt type *(load "Addat")* and then enter.
Now, type *"Addat"* and press enter again.

This dialog should appear :



Change some of the data and then press the "OK" button.
The title block data should be updated. Clever hey?

You can expand on this routine as much as you like using the following coding as a template.
Hint : You don't have to display all the attribute data stored in a block. Only display what you want the user to modify. As well, you can split your data over multiple dialog boxes. eg. One for title block, one for revisions, one for reference drawings, etc. All the data though is contained in one attribute.

Here's the coding, DCL code first :

```
attab : dialog {
      label = "Drawing Title Block";

    : edit_box {
      label = "&Drawing Number";
      key = "eb1";
      edit_width = 30;
    }

    : edit_box {
      label = "&Revision";
      key = "eb2";
      edit_width = 30;
    }

    : edit_box {
      label = "Drawn &By";
      key = "eb3";
      edit_width = 30;
    }

    : edit_box {
      label = "D&ate";
      key = "eb4";
```

```
    edit_width = 30;
   }

  : edit_box {
    label = "&Title";
    key = "eb5";
    edit_width = 30;
   }

 ok_cancel ;

 :text_part {
    label = "AfraLisp - http://www.afralisp.com";
  }


 }
```

And now the AutoLisp coding with plenty of in-line comments to assist you :

```
;CODING STARTS HERE
;
;All Tutorials and Code are provided "as-is" for purposes of instruction and
;utility and may be used by anyone for any purpose entirely at their own risk.
;Please respect the intellectual rights of others.
;All material provided here is unsupported and without warranty of any kind.
;No responsibility will be taken for any direct or indirect consequences
;resulting from or associated with the use of these Tutorials or Code.
;*****************************************************************************
;                    AfraLisp
;              http://www.afralisp.com
;               afralisp@afralisp.com
;               afralisp@mweb.com.na
;*****************************************************************************
;This application will extract attributes from a block and display them in a
;dialog box. The attributes will then be updated. Attab.dcl and Attab.dwg are
;required and must be within the AutoCAD search path.
;
;Usage : Open Attab.dwg then load and run Attab.lsp.
;*****************************************************************************

(prompt "\nATTAB Loaded....Type ATTAB to run.....")

(defun c:attab (/)

;first find all the blocks with attributes in the drawing
(setq ss1 (ssget "X" '((0 . "INSERT")(66 . 1))))
```

```
;if there are blocks
(if ss1

  ;do the following
  (progn

    ;set up the counter
    (setq count 0
          emax  (sslength ss1)
    )

    ;extract the block name
    (while (< count emax)

      (setq en   (ssname ss1 count)
            ed   (entget en)
            blkn (dxf 2 ed)
      )

      ;check if it is our block
      (if (= "attab-info" blkn)

              ;if it is, switch off the counter
        ;and set the found flag
        (setq count emax
              found T
        )

        ;it's not our block, increment the counter
        ;and loop
        (setq count (1+ count))

      );if

    );while

    ;if we find our block
    (if found

      ;display the dialog
      (ddisplay)

      ;cannot find our block
      (alert
        "\nDrawing Sheet has No Attributes
          \n     Use Manual Edit"
      )

    );if

  );progn
```

```
   ;there are no blocks in the drawing
   (alert
     "\nIncorrect Drawing Sheet
     \n    Use Manual Edit"
   )

 );if

;finish clean
(princ)

);defun

;;;********************************************************

(defun ddisplay (/)

      ;load the dialog
      (setq dcl_id (load_dialog "attab.dcl"))

      ;check it exists
      (if (not (new_dialog "attab" dcl_id))

      (exit)

      );if

      ;get the block entity data
      (setq edata (entget en))

      ;get the first attribute entity list
      (setq edata (entget (entnext (dxf -1 edata))))

      ;get the drawing number
      (setq eb1 (dxf 1 edata))

      ;get the second attribute entity list
      (setq edata (entget (entnext (dxf -1 edata))))

      ;get the revision
      (setq eb2 (dxf 1 edata))

      ;get the third attribute entity list
      (setq edata (entget (entnext (dxf -1 edata))))

      ;get the name
      (setq eb3 (dxf 1 edata))

      ;get the fourth attribute entity list
      (setq edata (entget (entnext (dxf -1 edata))))
```

```
    ;get the date
    (setq eb4 (dxf 1 edata))

    ;get the fifth attribute entity list
    (setq edata (entget (entnext (dxf -1 edata))))

    ;get the title
    (setq eb5 (dxf 1 edata))


;put the info into the dialog
(set_tile "eb1" eb1)
(set_tile "eb2" eb2)
(set_tile "eb3" eb3)
(set_tile "eb4" eb4)
(set_tile "eb5" eb5)

;set the focus to the drawing number
(mode_tile "eb1" 2)

;if cancel selected exit
(action_tile
  "cancel"
  "(done_dialog) (setq userclick nil)"
)

;if OK selected, retrieve the tile values
(action_tile
  "accept"
  (strcat
    "(progn (setq eb1a (get_tile \"eb1\"))"
    "(setq eb2a (get_tile \"eb2\"))"
    "(setq eb3a (get_tile \"eb3\"))"
    "(setq eb4a (get_tile \"eb4\"))"
    "(setq eb5a (get_tile \"eb5\"))"
   " (done_dialog)(setq userclick T))"
  )
)

;start the dialog
(start_dialog)

;unload the dialog
(unload_dialog dcl_id)


;if OK was selected
(if userclick

    (progn
```

```
  ;get the block entity data
  (setq edata (entget en))

  ;get the first attribute entity list
  (setq edata (entget (entnext (dxf -1 edata))))

  ;change the list with the new values, if any
  (setq el (subst (cons 1 eb1a) (assoc 1 edata) edata))

  ;update the attribute
  (entmod el)

  ;get the second attribute entity list
  (setq edata (entget (entnext (dxf -1 edata))))

  ;change the list with the new values, if any
  (setq el (subst (cons 1 eb2a) (assoc 1 edata) edata))

  ;update the attribute
  (entmod el)

  ;get the third attribute entity list
  (setq edata (entget (entnext (dxf -1 edata))))

  ;change the list with the new values, if any
  (setq el (subst (cons 1 eb3a) (assoc 1 edata) edata))

  ;update the attribute
  (entmod el)

  ;get the fourth attribute entity list
  (setq edata (entget (entnext (dxf -1 edata))))

  ;change the list with the new values, if any
  (setq el (subst (cons 1 eb4a) (assoc 1 edata) edata))

  ;update the attribute
  (entmod el)

  ;get the fifth attribute entity list
  (setq edata (entget (entnext (dxf -1 edata))))

  ;change the list with the new values, if any
  (setq el (subst (cons 1 eb5a) (assoc 1 edata) edata))

  ;update the attribute
  (entmod el)

  ;regen the drawing
       (command "REGEN")

);progn
```

```
  );if

(princ)

);defun

;;;*******************************************************

(defun dxf (code elist)

  (cdr (assoc code elist))

);defun
;;;*******************************************************

;load clean
(princ)

;;;*******************************************************
;
;CODING ENDS HERE
```

Please note that there is no error checking in this routine and I have left all variables as global to assist you in checking their values whilst you are analyzing the code.

DCL without the DCL File

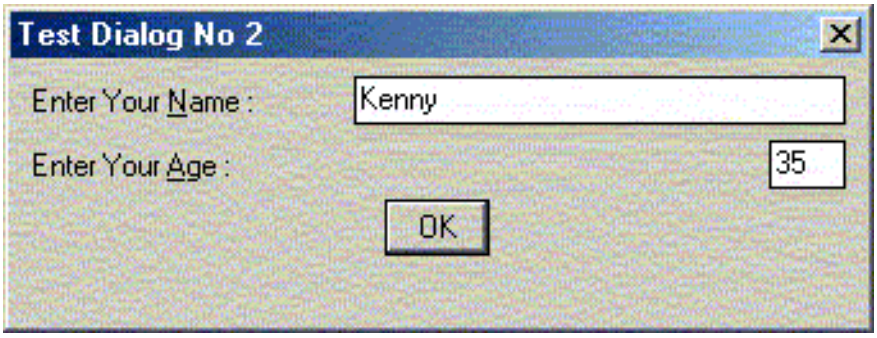
Have you ever wanted to load and run a DCL file without having a DCL file?
"What are you talking about Kenny?"

Easy, let's take a simple AutoLisp dialog example and tweek the coding a little bit so that our program "writes" the DCL file for us. Copy and paste the following and save it as "TEST_DCL3.LSP".

```
;AUTOLISP CODING STARTS HERE
(prompt "\nType TEST_DCL3 to run.....")

(defun C:TEST_DCL3 ( / dcl_id fn fname)

(vl-load-com)

(create_dialog)

(setq dcl_id (load_dialog fname))
     (if (not (new_dialog "temp" dcl_id))
         (exit )
     );if

(set_tile "name" "Enter Name Here")
(mode_tile "name" 2)
(action_tile "name" "(setq name $value)")
(action_tile "age" "(setq age $value)")
(action_tile "accept" "(val1)")

(start_dialog)
(unload_dialog dcl_id)

(alert (strcat "Your name is " name
               "\nand you are " age " years of age."))

(vl-file-delete fname)

(princ)

);defun

-----------------------

(defun val1 ()

(if (= (get_tile "name") "Enter Name Here")
        (progn
           (set_tile "error" "You must enter a name!")
           (mode_tile "name" 2)
        );progn
        (val2)
```

```
);if

);defun

-------------------

(defun val2 ()

(if (< (atoi (get_tile "age")) 1)
        (progn
          (set_tile "error" "Invalid Age - Please Try Again!!")
          (mode_tile "age" 2)
        );progn
        (done_dialog)
);if

);defun

-------------------

(defun create_dialog ()

(setq fname (vl-filename-mktemp "dcl.dcl"))

(setq fn (open fname "w"))

(write-line "temp : dialog { label = \"Test Dialog No 3\";
 : edit_box { label = \"Enter Your Name :\"; mnemonic = \"N\";
 key = \"name\"; alignment = centered; edit_limit = 30;
 edit_width = 30; } : edit_box { label = \"Enter Your Age :\";
 mnemonic = \"A\"; key =\"age\"; alignment = centered;
 edit_limit = 3; edit_width = 3; value = \"22\";
 } : button { key = \"accept\"; label = \"OK\";
 is_default = true; fixed_width = true; alignment = centered;
 } : errtile { width = 34; } }" fn)

(close fn)

);defun

(princ)
;AUTOLISP CODING ENDS HERE
```

Notice that there is no DCL file this time. Now load and run it.



See, what did I tell you? We've just loaded and ran a DCL file even though we haven't got one!!!
You could now compile this into a FAS file if you wished. (Hey, a FAS file with DCL - cool!)

In the next section, we'll have a closer look at this truly amazing phenomenon and have a closer look at the coding. At the same time, we'll get really clever and design ourselves a *"Variable-auto-self-sizing-Dialog-Box-without-a-DCL-file".*

Variable-auto-self-sizing-Dialog-Box-without-a-DCL-file

In this tutorial we're going to create an application that extracts attributes from a block and displays these attributes within a dialog box. The dialog will be created "on-the-fly" and the number of attribute edit boxes will be determined by the number of attributes.

We will be using a lot of Visual Lisp coding within this routine as I also want to demonstrate one method of extracting and updating attributes using Visual Lisp.

No further explanation is necessary as the coding is well commented :

```
;;;DCLATT.LSP
;This program is for demonstration and tutorial purposes only.
------------------------------------------
;This program, using Visual Lisp will extract attributes from
;a block and display them in a dialog box.
;The dialog box will be created "on the fly" with the relevant
;number of edit boxes ;to suit the number of attributes within
;the block.
;The new attribute data will then be retrieved from the dialog
;and the block updated.
;Written by Kenny Ramage - May 2002
;afralisp@mweb.com.na
;http://www.afralisp.com
------------------------------------------
;Usage :
;Load by typing (load "DCLATT") at the command prompt.
;Type DCLATT at the command prompt to run.
;Select any block containing attributes.
;Replace any of the values in the text boxes to updated
;the attributes.
------------------------------------------
;Dependencies : None that I know of.
;Not much in the way of error checking I'm afraid.
------------------------------------------
;Limitations : Will only edit a certain number of attributes.
;System dependent.
;I don't recommend more than 10.
;I've had up to 14 on my display.
------------------------------------------
------------------------------------------

(prompt "\nType DCLATT to run.....")

(defun c:dclatt ( / theblock thelist n taglist
                    txtlist lg fname fn nu dcl_id l relist)

;load the visual lisp extensions
(vl-load-com)
```

```
;get the entity and entity name
(setq theblock (car (entsel)))

;convert to vl object
(setq theblock (vlax-ename->vla-object theblock))

;check if it's a block
(if (= (vlax-get-property theblock 'ObjectName)
       "AcDbBlockReference")

       ;if it is, do the following
       (progn

              ;check if it has attributes
              (if (= (vlax-get-property theblock
                  'HasAttributes) :vlax-true)

                   ;if it has attributes, do the following
                   (progn

                      ;get the attributes
                      (getatt theblock)

                      ;create the dialog
                      (create_dialog)

                      ;run the dialog
                      (run_the_dialog)

                      ;update the attributes
                      (upatt)

                   );progn

                      ;No attributes, inform the user
                      (alert "This Block has No Attributes!!
                              - Please try again.")
              );if

       );progn

       ;it's not a block, inform the user
       (alert "This is not a Block!! - Please try again.")

);if

 (princ)

);defun

------------------------
```

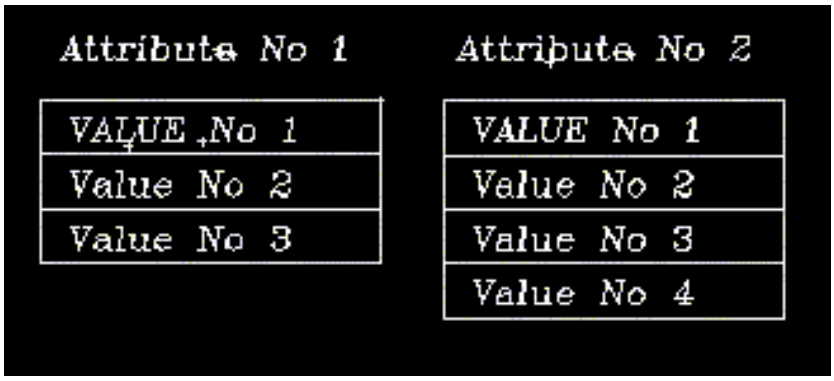
```
(defun getatt (enam)

;retrieve the attributes
(setq thelist (vlax-safearray->list
                (variant-value
                       (vla-getattributes enam))))

        ;process each attribute
        (foreach n thelist

          ;get the tag attribute data
          (setq taglist (cons (vla-get-tagString n) taglist)

          ;get the text attribute data
                   txtlist (cons (vla-get-textString n) txtlist)

                ;how many attributes?
                lg (length taglist)

          );setq
        );foreach

        ;reverse the lists
        (setq taglist (reverse taglist)
              txtlist (reverse txtlist))

 );defun
-------------------------
(defun create_dialog ()

;create a temp DCL file
(setq fname (vl-filename-mktemp "dcl.dcl"))

;open it to write
(setq fn (open fname "w"))

;write the dialog header coding
(write-line "temp : dialog { label = \"Edit Attributes\";" fn)

;reset the incremental control number
(setq nu 0)

;start the loop to create the edit boxes
(repeat lg

;create the edit boxes
(write-line ": edit_box {" fn)
(setq l (strcat "\"" "eb" (itoa nu) "\"" ";"))
(write-line (strcat "key = " l)  fn)
(setq l (nth nu taglist))
```

```
(write-line (strcat "label = " "\"" l "\"" ";") fn)
(setq l (nth nu txtlist))
(write-line (strcat "value = " "\"" l "\"" ";") fn)
(write-line "alignment = centered; edit_width = 20; }" fn)

;increment the counter
(setq nu (1+ nu))

);repeat

;ok and cancel button
(write-line "ok_only; }" fn)

;close the temp DCL file
(close fn)

);defun
-----------------------------

(defun run_the_dialog ()

;load the dialog file and definition
(setq dcl_id (load_dialog fname))
     (if (not (new_dialog "temp" dcl_id))
         (exit )
     );if

(mode_tile "eb0" 2)

;if the OK button is selected
(action_tile "accept" "(retatt)")


;start the dialog
(start_dialog)

;unload the dialog
(unload_dialog dcl_id)

;delete the temp DCL file
(vl-file-delete fname)

);defun
--------------------------

(defun retatt ()

;reset the increment counter
(setq nu 0)

start the loop
(repeat lg
```

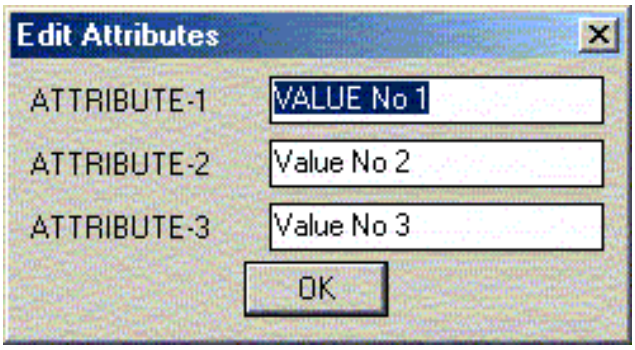
```
        ;retrieve the tile value
        (setq l (get_tile (strcat "eb" (itoa nu))))

        ;add it to the list
        (setq relist (cons l relist))

        ;increment the counter
        (setq nu (1+ nu))

);repeat

(setq relist (reverse relist))

;close the dialog
(done_dialog)

);defun
----------------------------------------

(defun upatt ()

;reset the increment counter
(setq nu 0)

;start the loop
(repeat lg

  ;update the attribute
  (vla-put-textstring (nth nu thelist) (nth nu relist))

  ;increment the counter
  (setq nu (1+ nu))

);repeat

;update the block
(vla-update theblock)

);defun

----------------------------
;clean loading
(princ)
----------------------------
;End of ATTDCL.LSP
----------------------------
```

Now you need to create 2 or three blocks containing a varying number of attributes.

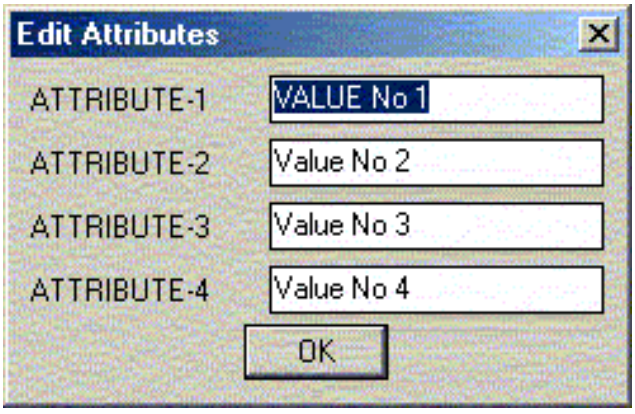
Here's what mine look like :



Load and run "DCLATT.LSP" and select "Attribute No 1".
You dialog should appear looking like this with three attribute edit boxes :



Now run the program again, but this time select "Attribute No 2".



The dialog will appear, but this time with four attribute edit boxes.
Clever hey?

A big thanks to *Stig Madsen* for the help with the coding in this project.

Set Dialogue Default Values.

Load Dialogue.

Call New Dialogue.

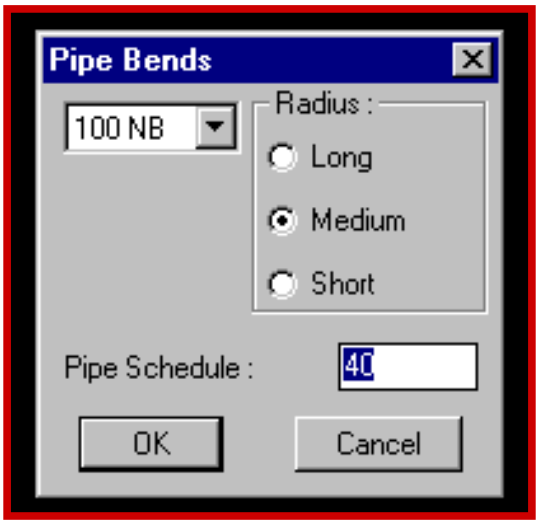
Initialize Dialogue

Setup Retrieval of User Input.

Start Dialogue.

Unload Dialogue.

Process User Input



DCL Model

****Click on Relevant Image for Coding****

****Click here for DCL Coding****

This model lays out the function sequence required to design a dialogue box using AutoLisp and Dialogue Control Language. (DCL)

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.
-
;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

;;;*************************************************************
;;;This section initializes the dialogue box tiles.
```

```
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.

  (if userclick
```

```
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

-
;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

;;;*************************************************************
```

```
;;;This section initializes the dialogue box tiles.
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*************************************************************
;;;If the O.K. tile was selected, process the users input values.
```

```
  (if userclick
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

-

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if
```

```
;;;*****************************************************************
;;;This section initializes the dialogue box tiles.
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*****************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box
```

```
   );action_tile

  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.
```

```
  (if userclick
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if
-
;;;*************************************************************
;;;This section initializes the dialogue box tiles.
```

```
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.

  (if userclick
```

```
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

;;;*************************************************************
;;;This section initializes the dialogue box tiles.
```

```
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule
-
;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

     "(setq rad \"Short\")
      ;store radius

      (mode_tile \"eb1\" 2)"
     ;set focus to edit box

   );action_tile

  (action_tile
     "accept"
     ;if O.K. pressed

      "(setq schedule (get_tile \"eb1\"))
      ;get the schedule entered

      (setq index (atof (get_tile \"lb1\")))
      ;get the index of the pipe diameter entered

      (done_dialog) (setq userclick T)"
     ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.

  (if userclick
```

```
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

;;;*************************************************************
;;;This section initializes the dialogue box tiles.
```

```
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile
-
;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.

  (if userclick
```

```
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

;;;*************************************************************
;;;This section initializes the dialogue box tiles.
```

```
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog
-
;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload

;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.

  (if userclick
```

```
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - AutoLisp Coding

```
(defun C:lisp51 ()
;define function - variables have been left as local
;for inspection purposes.

;;;*************************************************************
;;;This section assigns all the default values of the dialogue
;;;box into their relevant variables.

  (setq names '("50 NB" "60 NB" "70 NB" "80 NB" "90 NB"
                "100 NB" "125 NB" "150 NB" "200 NB" "250 NB"))
  ;default list for pipe diameter list box

  (setq rad "Long")
  ;set default for radius radio buttons

  (setq schedule "40")
  ;set default pipe schedule for edit box

  (setq index "3")
  ;set default pipe size index

;;;*************************************************************
;;;This section loads the relevant DCL file containing the
;;;dialogue box definition.

  (setq dcl_id (load_dialog "lisp51.dcl"))
  ;load dialog

;;;*************************************************************
;;;This section loads the particular dialogue box definition
;;;contained within the DCL file. It first checks that the
;;;DCL file has been loaded.

  (if (not (new_dialog "lisp51" dcl_id)
  ;test for dialog

      );not

    (exit)
    ;exit if no dialog

  );if

;;;*************************************************************
;;;This section initializes the dialogue box tiles.
```

```
;;;It fills the list box, pre-selects an item in the
;;;list box, switches on the default radio button and
;;;puts a default value into the edit box.

  (start_list "lb1")
  ;open the list box

  (mapcar 'add_list names)
  ;add the names to the list box

  (end_list)
  ;end the list box

  (set_tile "lb1" "3")
  ;select 4th item in list box

  (set_tile "rb1" "1")
  ;switch on radio button No 1

  (set_tile "eb1" "40")
  ;set default pipe schedule

;;;*************************************************************
;;;This section retrieves the values that the user selects
;;;or inputs. In effect, what we are saying here is :
;;;"Remember the coding that is in this section. When start_dialog
;;;is called, and  a tile is selected, proceed with the relevant
;;;coding.

  (action_tile "rb1"
  ;if radio button No 1 selected

      "(setq rad \"Long\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile "rb2"
  ;if radio button No 2 selected

      "(setq rad \"Medium\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile
```

```
  (action_tile "rb3"
  ;if radio button No 3 selected

      "(setq rad \"Short\")
       ;store radius

       (mode_tile \"eb1\" 2)"
      ;set focus to edit box

   );action_tile

  (action_tile
      "accept"
      ;if O.K. pressed

       "(setq schedule (get_tile \"eb1\"))
       ;get the schedule entered

       (setq index (atof (get_tile \"lb1\")))
       ;get the index of the pipe diameter entered

       (done_dialog) (setq userclick T)"
      ;close dialog, set flag

   );action tile

    (action_tile
    "cancel"
    ;if cancel button pressed

    "(done_dialog) (setq userclick nil)"
    ;close dialog, lower flag

    );action_tile

;;;*****************************************************************
;;;This section displays the dialogue box and waits for user input

  (start_dialog)
  ;start dialog

;;;*****************************************************************
;;;done_dialogue has been called, so the dialogue can be unloaded

  (unload_dialog dcl_id)
  ;unload
-
;;;*****************************************************************
;;;If the O.K. tile was selected, process the users input values.

  (if userclick
```

```
  ;if OK selected

  (progn
  ;do the following

        (setq index (fix index))
        ;convert index to integer

        (setq size (nth index names))
        ;get the pipe size from the list

        (alert (strcat "You Choose a : " size"\n"
                       rad " : Radius" "\n"
                       "Sched : " schedule " Pipe Bend"))
        ;string the results together and display them to the user

  );progn

  );if

;;;*****************************************************************

 (princ)
 ;finish clean

);defun

(princ)
;load clean

;;;*****************************************************************
```

Back to Model....

DCL Model - DCL Coding

```
lisp51  : dialog {                              //dialog name
          label = "Pipe Bends" ;                //give it a label

        : row {                                 //define a row
          children_alignment = top ;            //align to top

        : popup_list {                          //define list box
          key = "lb1" ;                         //give it a name
        }                                       //end list box

        : radio_column {                        //define radio column
          label = "Radius :";                   //give it a label

        : radio_button {                        //define radio button
          label = "Long" ;                      //give it a label
          key = "rb1" ;                         //give it a name
        }                                       //end radio button

        : radio_button {                        //define radio button
          label = "Medium";                     //give it a label
          key = "rb2" ;                         //give it a name
        }                                       //end radio button

        : radio_button {                        //define radio button
          label = "Short";                      //give it a label
          key = "rb3" ;                         //give it a name
        }                                       //end radio button

        }                                       //end radio column

        }                                       //end row

        spacer ;                                //add a space

        : edit_box {                            //define edit box
          key = "eb1" ;                         //give it a name
          label = "Pipe Schedule :" ;           //give it a label
        }                                       //end edit box

        spacer ;                                //add a space

        ok_cancel ;                             //predefined OK/Cancel button

        }                                       //end dialog
```

Acknowledgments and Links

A big thanks to :

My wife Heather.
My Mum and Dad
***EVERYONE* at VBA Expresso.**
The lads from BedRock - Pete, Eddie and Mike.
Frank Zander for his generosity and support.

And a BIG thank you to Marie and Jessie Rath for at least trying to control that reprobate Randall. (Thanks for everything pal.)

If I've forgotten someone, hey life's tough.....

Some of the best Links on the Web :

AfraLisp - http://www.afralisp.com

VBA Expresso - http://www.vbdesign.net/cgi-bin/ikonboard.cgi

CAD Encoding - The Journal - http://www.cadencoding.com

VB Design - http://www.vbdesign.net

Contract CADD Group - http://www.contractcaddgroup.com

CopyPaste Code - http://www.copypastecode.com

DsxCad - http://www.dsxcad.com

QnA~Tnt - http://www.dczaland.com/appinatt

BedRock - http://www.bedrockband.com