dlgHELP

Version 1.0a

**A Library of Dialog Box
Tools and Utilities
for Paradox for DOS**

(Demo Included!)

**Developed By
William A. Bailey, Jr., Ph.D.**

**9017 Gamesford Drive
Charlotte, NC 28277**

**Phone: (704) 541-8930
CompuServe: 102011,707
Internet: bailey@vnet.net
AOL: bbaileyjr**

26 Sep 1995

CONTENTS

1. OVERVIEW

If you have attempted to create dialog boxes for your Paradox for DOS applications, you already know that it can be among the most tedious tasks required by PAL. **dlgHELP** can alleviate much of this headache by reducing the creation of common dialog box types to simple procedure calls.

dlgHelp is library of procedures that can be divided into two types: 1) general purpose dialog box generation tools 2) and special purpose dialog box utilities. The first type allows you to define the parameters of the dialog box and the procedure will create the layout (spacing and alignment of text, lines, and controls) automatically. The following procedures fall into this first category:

1) PushDlg (1 to 6 pushbuttons, 1 to 18 lines of text)
2) AcceptDlg (1 to 6 type-in boxes, 2 or 3 pushbuttons)
3) CheckDlg (1 to 10 check boxes)
4) RadioDlg (1 to 10 radio buttons)
5) PickDlg (picklist, type-in box, 1 to 6 pushbuttons)
6) MsgWin (1 to 18 lines of text, non-modal)

The second type of procedure simply needs to be called. There are no parameters. The following procedures fall into this second category:

1) ClockDlg
2) ColorDlg
3) ExportDlg
4) FindRecord
5) SearchAndReplace
6) SysInfoDlg

<u>Global Variables</u>

Values are returned to the calling program via predefined global variables, not via the parameters. The following variables and dynamic arrays are used by one or more of the procedures to assign return values. <u>If your program uses one of them, conflicts may occur</u>. You should either change your variable names (preferable) or edit the procedure (which requires a password that you receive upon registering).

```
AcceptValue[], CheckValue[]  (dynamic arrays)
AcceptValue
PickValue
RadioValue
ExportFileName, ExportDirectory, ExportType
ButtonValue
dlgRow, dlgCol
```

<u>How to Use</u>

To incorporate these utilities and tools into your application, simply copy the library file **dlgHELP.lib** to your application directory (or whatever directory your libraries are in if different from your tables) and include it in your library definition (see your PAL programmer's manual for more information on libraries if you are not already familiar with them). Typically, you will simply assign it to the *autolib* system variable. For example, if you are not using any other libraries, then the statement

```
autolib="dlghelp"
```

should be placed at or near the beginning of your main script (technically, as long as it is placed before the first call to any of the library procedures, it doesn't really matter where, although it is good design practice to place library declarations at the beginning). As long as *autolib* is global to your application, then you shouldn't have any problems. However, if you are using "closed" procedures or if *autolib* is somehow not global to some procedures, then you need to place the above statement in each of those procedures to make the **dlgHELP** tools available.

It should probably go without saying, but I will say it anyway, because SOMEONE won't know this. The dialog box features were introduced with Paradox for DOS 4.0, which this library was compiled with. You cannot use this library with Paradox 3.5 or earlier since it will not recognize the new PAL dialog commands.

If you wish to remove some of the procedures from the library in order to make **dlgHELP.lib** smaller, you can remove the appropriate scripts from the **dlgHELP.db** table and recompile the library using the **makelib.sc** script (which reads the scripts to be included in the library from the **dlgHELP.db** table). Be careful that you do not remove scripts that are required by other procedures (see the requirements below). You can also use this script to recompile the procedures after editing them (however, you must register in order to get the password which will enable you to uprotect the scripts, otherwise you cannot edit them).

Contents of the **dlgHELP.zip** file:
dlgdemo.zip - NEW demo program. Unzip this file and run **dlgdemo.sc**.
readme.txt - Initial documentation.
dlgHELP.wri - This file: the **dlgHELP** manual. Contains the documentation for **dlgHELP**. Sorry it's so large, but I wanted to include some graphic bitmap screen captures to illustrate sample dialog boxes.
dlgHELP.db - Table containing the names of the scripts to be compiled into **dlgHELP** library file **dlgHELP.lib**
dlgHELP.lib - This is all that needs to be included with your application (see How to Use above).
makelib.sc - Library compilation script. Compiles all scripts in **dlgHELP.db** table into **dlgHELP.lib** library. You can edit this script if you want to compile the scripts (or a subset of them) into a different library file.
register.sc - You can run this script to decrypt the **dlgHELP** scripts below. You will be required to enter a password which you will receive when you register. Otherwise, they cannot be viewed, editted or debugged.
They can, however, still be run or recompiled into a new library file name.
accptdlg.sc contains the **AcceptDlg** procedure. This is used by the **FindRecord** procedure.
checkdlg.sc contains the **CheckDlg** procedure
clockdlg.sc contains the **ClockDlg** procedure
colordlg.sc contains the **ColorDlg** procedure
expdlg.sc contains the **ExportDlg** procedure
findrec.sc contains the **FindRecord** procedure. Requires the **AcceptDlg** procedure to run.
msgwin.sc contains the **MsgWin** procedure
pickdlg.sc contains the **PickDlg** procedure
pushdlg.sc contains the **PushDlg** procedure. This is used by other procedures (**SearchAndReplace**, **SysInfoDlg**).
repldlg.sc contains the **ReplaceDlg** procedure. This is used by the **SearchAndReplace** procedure
srch&rep.sc contains the **SearchAndReplace** procedure. Requires both the **ReplaceDlg** and **PushDlg** procedures.
sysinfo.sc contains the **SysInfoDlg** procedure. This requires the **PushDlg** procedure.
textcolr.sc contains the **SetTextColor** procedure which is used by most of the other procedures. Removing it from the **dlgHELP.db** file will probably result in a run-time error message (i.e. DEBUG | CANCEL).

Disclaimer & Copyright

2. REGISTRATION INFORMATION

dlgHELP is shareware, which means if you like it and use it, you should pay for it. The registration fee is very reasonable:

$15.

I have considered different marketing approaches to enlarge my base of registered users. I could tell you that 2000 starving children with AIDS will die in South Central Bangladesh without your help (the Sally Struthers guilt-manipulation strategy). Or I could could threaten you and your family (or, even worse, your computer) with physical destruction (the Jimmy Swaggart hell-fire and brimstone approach). Or I could promise you unimaginable riches (including a time-share condominium at Heritage USA) for your generous faith contribution (the Jim and Tammy Faye method). Or I could say that I will die if I don't receive 250 registrations within six weeks (the Oral Roberts "take me home" strategy).

The truth is, however, that I desperately need tennis lessons to improve my backhand (well... yeah, my serve and forehand could use a little help also) so I can finally beat my wife. My friends have told me that this approach will never work. Users will never be persuaded to register their shareware if they know that the fee is going to be used for tennis lessons. So....

I have decided to offer an incentive. If you register, I will send you a password that will "decrypt" (or, in PAL parlance, "unprotect") the scripts that are in the **dlgHELP** library. This will allow you to examine the code for ideas, or customize the code for your own applications. For example, some people find the flexibility (as determined by the number of parameter variables) to be more than they need. You could "hard code" some parameter values (such as the text color and row/column position on the screen or even the number of buttons) into the procedure and simply use a couple of parameters, thus simplifying your procedure calls.

In addition, I am willing to answer specific questions concerning the procedures for registered users (as long as it doesn't get out of hand). I guess you would call this technical support. In any case, the registration fee is well worth it. Once you receive the password, simply run the **register.sc** script and enter the password when prompted.

You can register via **CompuServe's Shareware Registration (GO SWREG; Registration ID: 7456)** or send check or money order to:

William A. Bailey, Jr.
9017 Gamesford Drive
Charlotte, NC 28277

Please include your e-mail address so I can send you the password. If you don't have an e-mail address, please include a stamped, self-addressed envelope and I'll send it "snail mail."

If you are really in a hurry, just send an e-note indicating that you have registered (i.e. it is in the mail or you have used CompuServe) and I'll send you the password right away. I'm a trusting soul. My e-mail address is:

Internet: bailey@vnet.net **CompuServe: 102011,707** **AOL: bbaileyjr**

3. PushDlg(boxtitle,msgline,buttonlabel,align,r,c,textcolor)

This procedure quickly and easily creates the most common of all dialog boxes: pushbuttons and text. Examples (see below) include Abort/Retry/Ignore (text and 3 buttons), Info or Stop (text and OK button), Yes/No Question (text and 2 buttons), Retry/Cancel (text and 2 buttons), Yes/No/Cancel (text and 3 buttons). You can define 1 to 6 buttons and up to 18 lines of text. All of the buttons except the last one are automatically defined as OK buttons; the last is defined as a CANCEL button. This means that regardless of the text labels placed on the buttons, pressing any button but the last one will result in the entered values being accepted. Pressing the last button will cause any entries to be ignored. The first button is defined as the DEFAULT.

The message is assigned to the *msgline* parameter. If the message is more than one line you must use an array, with each line requiring one array element (including blank lines). Each message line will be centered. If you want them to all appear left-aligned, make sure each element is the same number of characters.

You can identify which button was pressed via the global variable *buttonvalue*. The text label for the pressed button is assigned to *buttonvalue*, minus any '~' characters that may have been used to defined the hotkey character. For example, if **~O~K** is the label for the pressed button, then *buttonvalue* will be assigned the string value "OK". *Buttonvalue* will be empty if the last button is pressed (since it is always a CANCEL button).

Parameter Descriptions
boxtitle - An alphanumeric (string) variable defining the title of the dialog box. It will be centered automatically.

msgline - Text of message. Use an alphanumeric variable for a single-line message, fixed or dynamic array for multiple lines (one element per line). Blank lines must also be assigned an array element.

buttonlabel - Defines the label text for each pushbutton. Use an alphanumeric variable for a single button; fixed array or dynamic array for multiple buttons. If you wish to define hotkeys for each button, remember to include a '~' character before and after the hotkey character. For, example, **~C~ancel** could be the label for a pushbutton, where the letter **C** would be the hotkey.

align - "V" for vertically-aligned buttons, "H" for horizontally-aligned buttons. Actually, anything but a "V" will result in horizontal alignment.

r, *c* - Defines the upper left coordinates (row, column) of dialog box. To center the dialog box, use r=0 and c=0.

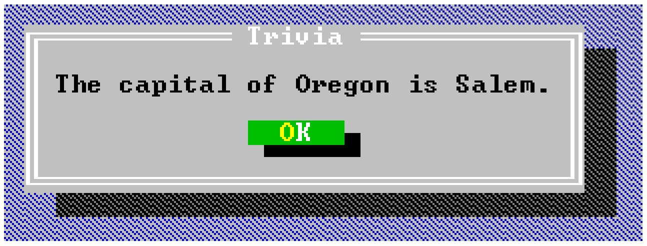
textcolor - Defines the foreground color of dialog canvas (affects text and lines). You can enter either a number from 0 to 15 or the color description (see appendix for list of color codes and descriptions). A blank value ("") defaults to the color black.

Example Usages

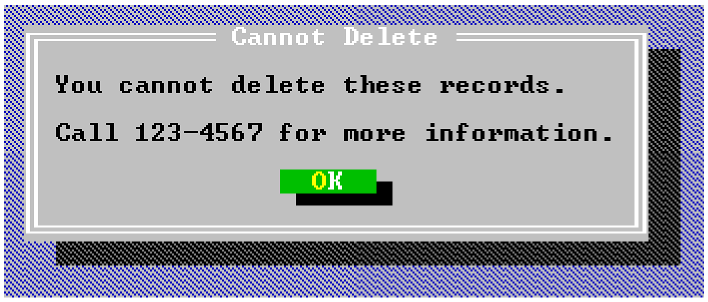
```
autolib="dlghelp"
; Example 1
pushDlg("Trivia","The capital of Oregon is Salem.","~O~K","H",0,0,"")
; Example 2
ARRAY msg[3]
msg[1]="You cannot delete these records.    "
msg[3]="Call 123-4567 for more information. "
pushDlg("Cannot Delete",msg,"~O~K","H",0,0,"")
(cont.)
```

```
; Example 3
ARRAY BL[2]
BL[1]="~Y~es"
BL[2]="~N~o"
pushDlg("Confirm","Do you wish to delete this record?",BL,"H",0,0,"")
; Example 4 - this one is a little more unusual; no message; vertical buttons
ARRAY BL[5]
BL[1]="Menu ~1~"
BL[2]="Menu ~2~"
BL[3]="Menu ~3~"
BL[4]="Menu ~4~"
BL[5]="~E~xit"
pushDlg("Main Menu","",BL,"V",0,0,"Red")
```

<u>Screen Captures of Examples Above</u>

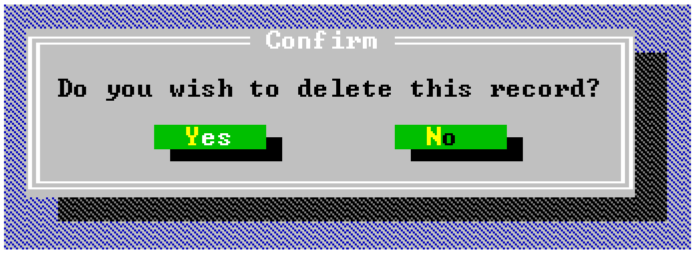


The capital of Oregon is Salem.

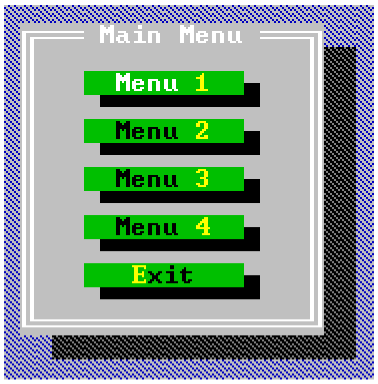


You cannot delete these records.

Call 123-4567 for more information.



Do you wish to delete this record?



4. AcceptDlg(boxtitle,acceptlabel,acceptwidth,buttonlabel,r,c,textcolor)

This procedure displays a dialog box with ACCEPT (type-in box) controls. This allows the user to enter values which can then be captured by the script, similar to the use of the ACCEPT statement.

You can define 1 to 6 type-in boxes and 2 or 3 pushbuttons. All of the buttons except the last one are automatically defined as OK buttons; the last is defined as a CANCEL button. This means that regardless of the text labels placed on the buttons, pressing any button but the last one will result in the entered values being accepted, pressing the last button will cause any entries to be ignored. The first button is defined as the DEFAULT.

If the entries are accepted, they will be assigned to the global dynamic array variable *AcceptValue[]*. If only a single type-in box is defined, then the entry will be assigned to *AcceptValue[1]*; two entries will be assigned to *AcceptValue[1]* and *AcceptValue[2]*, respectively, and so on. Tip: Assigning values to the elements of *AcceptValue[]* prior to calling **AcceptDlg** will result in those values being displayed in the corresponding type-in boxes as defaults.

In addition, you can identify which button was pressed via the global variable *ButtonValue*. The text label for the pressed button is assigned to *ButtonValue*, minus any '~' characters that may have been used to defined the hotkey character. For example, if **~O~K** is the label for the last button, then *ButtonValue* will be assigned the string value "OK". *Buttonvalue* will be empty if the last button is pressed (since it is always a CANCEL button).

Note: If your program already uses the variables *AcceptValue[]* or *ButtonValue*, conflicts may occur. You should either change your variables (preferable) or edit the **AcceptDlg** script (which requires a password that you receive upon registering).

Parameter Descriptions
boxtitle - An alphanumeric (string) variable defining the title of the dialog box. It will be centered automatically.

acceptlabel - Defines the label text for each type-in box. Use an alphanumeric variable for a single type-in box, fixed array or dynamic array for multiple type-in boxes.

acceptwidth - Defines the size of the edit region for each type-in box. Use an alphanumeric variable for a single type-in box, fixed array or dynamic array for multiple type-in boxes. This variable must be the same size and type as *acceptlabel*.

buttonlabel - Defines the label text for each pushbutton. Use an alphanumeric variable for a single button, fixed array or dynamic array for multiple buttons. If you wish to define hotkeys for each button, remember to include a '~' character before and after the hotkey character. For, example, **~C~ancel** could be the label for a pushbutton, where the letter **C** would be the hotkey.

r, *c* - Defines the upper left coordinates (row, column) of dialog box. To center the dialog box, use r=0 and c=0.

textcolor - Defines the foreground color of dialog canvas (affects text and lines). You can enter either a number from 0 to 15 or the color description (see appendix for list of color codes and descriptions). A blank value ("") defaults to the color black.

Example Usage (2 buttons, 2 type-in boxes, centered, with blue text)

```
autolib="dlghelp"
DYNARRAY AL[]
DYNARRAY AW[]
DYNARRAY BL[]
AL[1]="First Name"
AL[2]="Last Name"
AW[1]=15    ; 15 characters allowed for first name
AW[2]=20    ; 20 characters allowed for last name
BL[1]="~O~K"
BL[2]="~C~ancel"
AcceptDlg("Customer Name",AL,AW,BL,0,0,"Blue")
MESSAGE "First Name: "+AcceptValue[1]+"  Last Name: "+AcceptValue[2]
x=getchar()
```

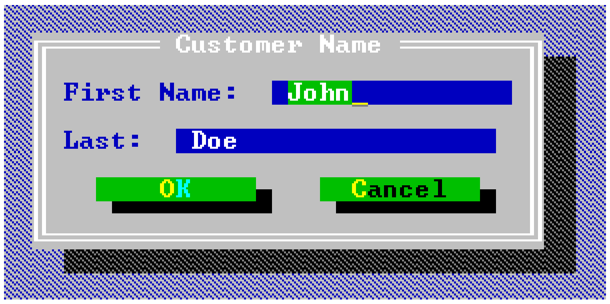
If the first button is pressed, then the entry for the first name will be assigned to *AcceptValue[1]* and the entry for the last name will be assigned to *AcceptValue[2]*. *ButtonValue* will have the value "OK".

If the second button is pressed, then *AcceptValue[1]* and *AcceptValue[2]* will be empty (but they *will* exist) and *ButtonValue* will equal "Cancel".

If you wish to assign default values for the first and last names, add the following code prior to calling the **AcceptDlg** procedure (if the user presses the CANCEL button, these values will be retained -- they will not be emptied):

```
DYNARRAY AcceptValue[]
AcceptValue[1]="John"  ; obviously you can supply your own default values
AcceptValue[2]="Doe"
```

Screen Capture



5. CheckDlg(boxtitle,checktitle,checklabel,checkrows,r,c,textcolor)

This procedure displays a dialog box with 1 to 10 check box controls. The user can select any combination of check boxes. The dialog box also contains two pushbuttons with the labels "OK" and "Cancel". As the labels imply, the first button is an OK button (the DEFAULT) and the second is a CANCEL button.

If the check box entries are accepted (by pressing the OK button), the value True or False will be assigned to each of the elements of the global dynamic array variable *CheckValue[]*. If the first check box is "checked," then *CheckValue[1]* will be True, otherwise it will be False, and so on for each check box. Tip: Assigning a value of True to the elements of *CheckValue[]* prior to calling **CheckDlg** will result in the corresponding check boxes being checked as defaults.

In addition, you can identify which button was pressed via the global variable *ButtonValue*. If the OK button is pressed, *ButtonValue* will equal "OK". If the CANCEL button is pressed, *ButtonValue* will be empty.

Note: If your program already uses the variables *CheckValue[]* or *ButtonValue*, conflicts may occur. You should either change your variables (preferable) or edit the **CheckDlg** script (which requires a password that you receive upon registering).

Parameter Descriptions

boxtitle - An alphanumeric (string) variable defining the title of the dialog box. It will be centered automatically.

checktitle - An alphanumeric variable defining the title for the set of check boxes. The grouping of check boxes is "framed," that is, it is enclosed within a single-line box. The title will be centered at the top of the frame.

checklabel - Defines the label text for each check box. Use an alphanumeric variable for a single check box, fixed array or dynamic array for multiple check boxes.

checkrows - Defines the number of rows to be used in arranging the group of check boxes. For example, if there are 8 check boxes, specifying 2 rows will result in an arrangement of 4 check boxes per row.

r, *c* - Defines the upper left coordinates (row, column) of dialog box. To center the dialog box, use r=0 and c=0.

textcolor - Defines the foreground color of dialog canvas (affects text and lines). You can enter either a number from 0 to 15 or the color description (see appendix for list of color codes and descriptions). A blank value ("") defaults to the color black.

Example Usage (6 check boxes, 3 rows, centered, with black text)

```
autolib="dlghelp"
DYNARRAY CL[]
CL[1]="Customer"
CL[2]="Orders"
CL[3]="BackOrd"
CL[4]="Employee"
CL[5]="Products"
CL[6]="Sales"
CheckDlg("Tables to Copy","Select Tables",CL,3,0,0,"")
MESSAGE STRVAL(CheckValue[1])+STRVAL(CheckValue[2])+STRVAL(CheckValue[3])+
       STRVAL(CheckValue[4])+STRVAL(CheckValue[5])+STRVAL(CheckValue[6])
x=getchar()
```

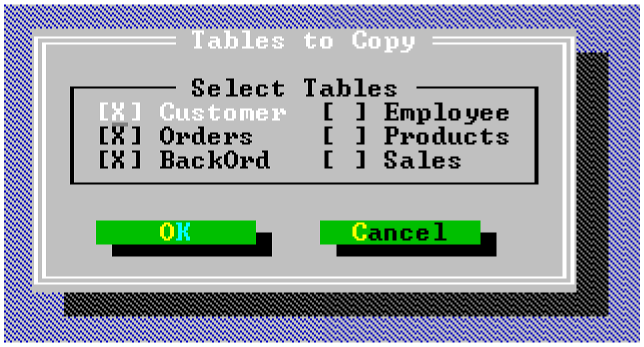
If the OK button is pressed, then the value True will be assigned to *CheckValue[1]* if the "Customer" check box is selected, to *CheckValue[2]* if the "Orders" check box is checked, and so on. If the check box is not selected, the value of the corresponding *CheckValue[]* element will be False. *ButtonValue* will equal "OK".

If the second button is pressed, then the *CheckValue[]* elements will all be False and *ButtonValue* will equal "Cancel".

If you wish to assign default selections for the check boxes, add the following code prior to calling the **CheckDlg** procedure (if the user presses the CANCEL button, these values will be retained -- they will not be reset to False):

```
DYNARRAY CheckValue[]
CheckValue[1]=True      ; obviously you can assign your own True/False values
CheckValue[2]=True
CheckValue[3]=True
CheckValue[4]=False
CheckValue[5]=False
CheckValue[6]=False
```

Screen Capture



6. RadioDlg(boxtitle, radiotitle, radiolabel, radiorows, r, c, textcolor)

This procedure displays a dialog box with 1 to 10 radio button controls. The user can select only one of the radio buttons. The dialog box also contains two pushbuttons with the labels "OK" and "Cancel". As the labels imply, the first button is an OK button (the DEFAULT) and the second is a CANCEL button.

If the radio button selection is accepted (by pressing the OK button), the ordinal value of the radio button is assigned to the global variable *radiovalue*. If the first radio button is selected, then *radiovalue* will equal 1; if the second button is selected, then *radiovalue* will equal 2, and so on. Tip: Assigning a numeric value to *radiovalue* prior to calling **RadioDlg** will result in the corresponding button appearing selected by default.

In addition, you can identify which button was pressed via the global variable *ButtonValue*. If the OK button is pressed, *ButtonValue* will equal "OK". If the CANCEL button is pressed, *ButtonValue* will be empty.

Note: If your program already uses the variables *radiovalue* or *ButtonValue*, conflicts may occur. You should either change your variables (preferable) or edit the **RadioDlg** script (which requires a password that you receive upon registering).

Parameter Descriptions
boxtitle - An alphanumeric (string) variable defining the title of the dialog box. It will be centered automatically.

radiotitle - An alphanumeric variable defining the title for the set of radio buttons. The grouping of radio buttons is "framed", that is, it is enclosed within a single-line box. The title will be centered.

radiolabel - Defines the label text for each radio button. Use an alphanumeric variable for a single radio button; fixed array or dynamic array for multiple radio buttons.

radiorows - Defines the number of rows to be used in arranging the group of radio buttons. For example, if there are 8 radio buttons, specifying 2 rows will result in an arrangement of 4 radio buttons per row.

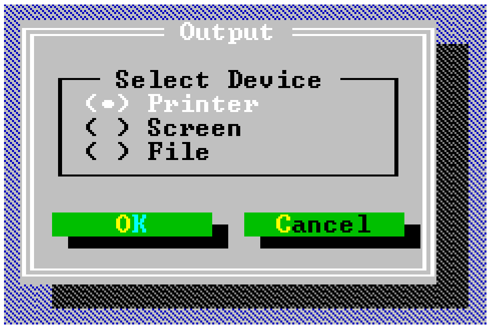
r, *c* - Defines the upper left coordinates (row, column) of dialog box. To center the dialog box, use r=0 and c=0.

textcolor - Defines the foreground color of dialog canvas (affects text and lines). You can enter either a number from 0 to 15 or the color description (see appendix for list of color codes and descriptions). A blank value ("") defaults to the color black.

Example Usage (3 radio buttons, 3 rows, centered, with black text)

```
autolib="dlghelp"
DYNARRAY RL[]
RL[1]="Printer"
RL[2]="Screen"
RL[3]="File"
RadioDlg("Output","Select Device",RL,3,0,0,"")
MESSAGE STRVAL(radiovalue)
x=getchar()
```

Screen Capture



7. PickDlg(boxtitle, picktitle, acceptlabel, picklist, buttonlabel, pickrows, pickcolumns, align, r, c, textcolor)

This procedure is a more flexible version of the type-in box/picklist dialog box used by Paradox for selecting tables (or other files). For example, if you select VIEW from the main Paradox menu, a dialog box appears containing a single type-in box (with the label "Table: ") and an empty pick list (with four columns) below it. If you press ENTER while in the empty type-in box, a list of tables in the default directory appears in the pick list. The first table is highlighted and you can use the arrow keys to navigate the picklist and highlight other tables. As you do this, the name of the currently highlighted table is displayed in the type-in box.

PickDlg does exactly the same thing, except that, in addition, you can specify the title of the dialog box, a title for the picklist frame, a label for the type-in box, up to six pushbuttons (with custom labels), how many rows and columns for the picklist, whether the buttons will be displayed horizontally along the bottom or vertically along the right edge, the location of the dialog box on the screen, and the color of the foreground text. Whew! This was challenging to create but very useful. By the way, if you just want a picklist without the type-in box, simply pass an empty string for the *acceptlabel* parameter. The type-in box will be removed., leaving only a picklist which is useful for field lookups.

You can define 1 to 6 pushbuttons. All of the buttons except the last one are automatically defined as OK buttons. The last is defined as a CANCEL button. This means that regardless of the text labels assigned to the buttons, pressing any button but the last one will result in the entered or selected value being accepted. Pressing the last button will cause any entry or selection to be ignored. The first button is defined as the DEFAULT.

If the entry or selection is accepted, the string value is assigned to the global variable *acceptvalue* while the index value of the corresponding element in the picklist array (see parameter list) is assigned to *pickvalue*. If the *picklist* parameter is a fixed array, then *pickvalue* will be numeric. If the *picklist* parameter is a dynamic array, then *pickvalue* will be the same datatype as the tag.

In addition, as with the other procedures, you can identify which button was pressed via the global variable *buttonvalue*. The text label for the pressed button is assigned to *buttonvalue*, minus any '~' characters that may have been used to defined the hotkey character. For example, if **~O~K** is the label for the last button, then *buttonvalue* will be assigned the string value "OK". *Buttonvalue* will be empty if the last button is pressed (since it is always a CANCEL button).

Parameter Descriptions
boxtitle - An alphanumeric (string) variable defining the title of the dialog box. It will be centered automatically.

picktitle - Title for picklist frame. A single-line box is placed around the picklist table. The title, which can be blank, will be centered.

acceptlabel - An alphanumeric variable defining the label text for the type-in box. Use an empty string ("") to exclude the type-in box.

picklist - A fixed or dynamic array containing the items for the picklist.

buttonlabel - Defines the label text for each pushbutton. Use an alphanumeric variable for a single button, fixed array or dynamic array for multiple buttons. If you wish to define hotkeys for each button, remember to include a '~' character before and after the hotkey character. For, example, **~C~ancel** could be the label for a pushbutton, where the letter **C** would be the hotkey.

pickrows - Defines the number of picklist rows.

pickcolumns - Defines the number of picklist columns.

align - "V" for vertically-aligned buttons, "H" for horizontally-aligned buttons. Actually, anything but a "V" will result in horizontal alignment.

r, *c* - Defines the upper left coordinates (row, column) of dialog box. To center the dialog box, use r=0 and c=0.

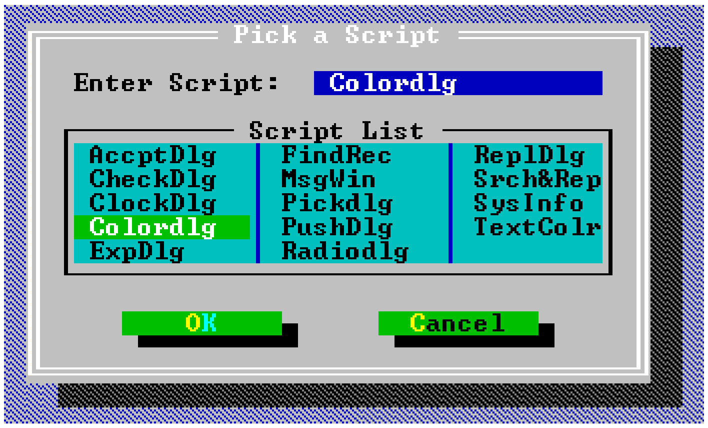
textcolor - Defines the foreground color of dialog canvas (affects text and lines). You can enter either a number from 0 to 15 or the color description (see appendix for list of color codes and descriptions). A blank value ("") defaults to the color black.

<u>Example Usage</u>

```
autolib="dlghelp"
; the picklist array is loaded with values from a table; in this case, I just used the script names in the dlghelp table
DYNARRAY picklist[]
VIEW "dlghelp"
i=0
scan
  i=i+1
  picklist[i]=[Script]
endscan
CLEARALL   ; removes the dlghelp table from the screen
DYNARRAY BL[]   ; button labels
BL[1]="~O~K"
BL[2]="~C~ancel"
PickDlg("Pick a Script","Script List","Enter Script",picklist,BL,5,3,"H",0,0,"")
MESSAGE acceptvalue+" "+STRVAL(pickvalue)
x=getchar()
```

<u>Screen Capture</u>

This is what is looks like <u>after</u> pressing the ENTER key (while in the type-in box) and scrolling down to the Colordlg choice. Prior to pressing the ENTER key, the type-in box and script list appear empty.



Enter Script: Colordlg

8. MsgWin(boxtitle,msgline)

This procedure differs from all the others in that it is a non-modal window and not a dialog box (which is almost always modal). What that means is that this window can appear on the screen and stay there while the program continues processing. Typically, the MESSAGE statement is used to indicate the status of a long process by displaying a line of text near the bottom right-hand corner of the screen. The **MsgWin** procedure simulates a Windows Status dialog box instead, providing a much more attractive interface (see screen captures below).

It is very easy to use. There are two parameters: *boxtitle* and *msgline*. *Boxtitle* requires a string value and will be centered at the top of the window frame. *Msgline* requires either a string value or a fixed or dynamic array with string elements, one for each line of text in the window (an example of each type is given below). Once the procedure is called, the size and shape of the window will be determined automatically based on the length of the text and the number of lines. Each line will be centered within the window so if you want the text to be left-aligned, make sure that each line of text is the same length (pad it with blanks if you have to). You are allowed a maximum of 18 lines.

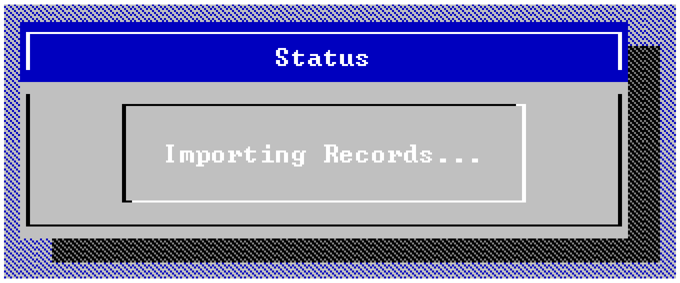
One global variable is used to assign a handle to the window: *msgwinhandle*. You need to use this to close the window at some point. In the examples below you can see that the following two commands are necessary:

```
WINDOW SELECT msgwinhandle
WINDOW CLOSE.
```

Example Usage 1

```
autolib="dlghelp"
; since message is a single line, text can be directly passed as string value
MsgWin("Status","Importing File...")
; these lines are simply used to pass time to illustrate the message window
for i from 1 to 1000
  message "Record "+strval(i)
endfor
; the message window can now be closed
WINDOW SELECT msgwinhandle
WINDOW CLOSE
```

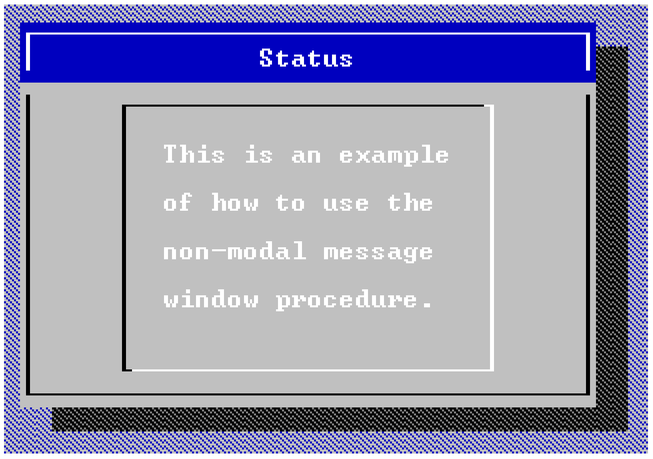
Screen Capture



Example Usage 2

```
autolib="dlghelp"
; Since message is multiple lines, an array must be used.
; Notice that index values are skipped to create blank lines.
; This is harder to do with Dynamic Arrays, so I recommend
; sticking with Fixed arrays.
ARRAY msg[8]
msg[1]="This is an example "
msg[3]="of how to use the  "
msg[5]="non-modal message  "
msg[7]="window procedure.  "
MsgWin("Status",msg)
for i from 1 to 1000
  message "Record "+strval(i)
endfor
WINDOW SELECT msgwinhandle
WINDOW CLOSE
```

Screen Capture



```
Status

This is an example

of how to use the

non-modal message

window procedure.
```

9. ClockDlg()

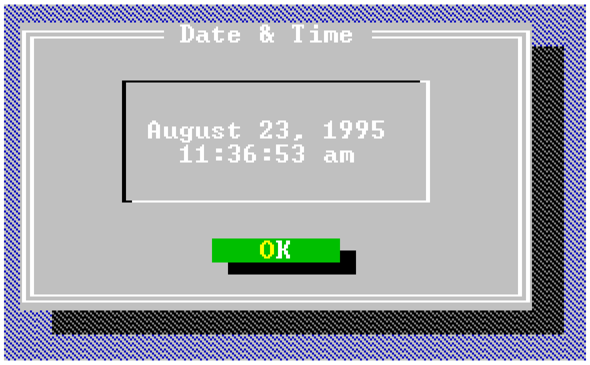
This procedure simply displays the current date and time. However, it is not just a snapshot of the time, but an actual running clock.

There are no parameters, but there are two global variables that are useful for positioning the dialog box on the screen: *dlgRow* and *dlgCol*. By default, the clock will be centered on the screen each time it is called. If you want it to be positioned somewhere else, you can assign values to the variables *dlgRow* and *dlgCol* prior to calling **ClockDlg**. Finally, you can also have the clock reappear at its last position each time it is called because **ClockDlg** updates the values of *dlgRow* and *dlgCol* whenever you move the dialog box on the screen. Thus, if you use the following code, you can specify the initial position of the clock on the screen and then let the program determine future positions based on where it was at last.

Example Usage

```
autolib="dlghelp"
if not isassigned(dlgRow) then ;this must be the first running of the script
  dlgRow=5   ; these coordinates will center the clock
  dlgCol=26
endif
ClockDlg()
; at this point dlgRow and dlgCol will contain the most recent coordinates
; of the clock.  If you rerun the script, the clock should reappear in its
; most recent position.  This is true only as long as the values of dlgRow and
; dlgCol are not "released" either by using the RELEASE VARS statement or by
; limiting their scope in a way that results in their values not being
; maintained between calls to the ClockDlg procedure.
```

Screen Capture



10. ColorDlg()

This procedure is a special application of the more general **PickDlg** procedure. It allows you to see the effect of different foreground and background color selections and then returns the Paradox color code for the selection that is made. I don't know how often you'll use it, but it is so cool I had to include it.

Instead of requiring parameter values to determine the characteristics of the dialog box (as would be necessary with the **PickDlg** procedure), I have already assigned default values to the key parameters within the ColorDlg script, however, you can easily edit them within the script if you want your picklist to take on a different size and shape (of course, you have to register first before you can view or edit the script).

There are two ways to select a color: 1) type the color combination (e.g. Red on Green) in the type-in box or 2) press ENTER while the cursor is in the type-in box to display a pick list of color combinations (there are 256 - 16 foreground x 16 background).

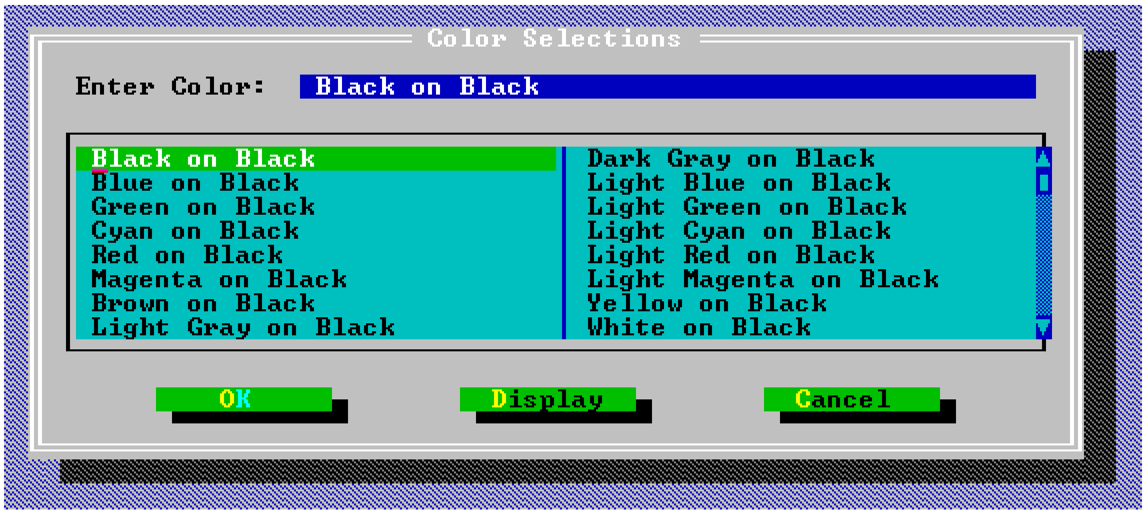
There are three predefined buttons on the dialog box: OK, Display, and Cancel. Pressing "Display" when a color has been selected will cause the color picklist frame to display that color selection. Pressing OK will cause the text of the color selection to be assigned to the global variable *acceptvalue* and the Paradox color code value to be assigned to *pickvalue*. For example, if the color combination 'Red on Green' is selected, *acceptvalue* will equal "Red on Green" and *pickvalue* will equal 36. Pressing CANCEL will result in both of these variables remaining empty. Also, as with other procedures, the global variable *buttonvalue* is assigned the value of the button that was pressed: "OK" or "" (for CANCEL).

Note: If your program already uses the variables *acceptvalue* or *pickvalue*, conflicts may occur. You should either change your variables (preferable) or edit the **ColorDlg** script (which requires a password that you receive upon registering).

<u>Example Usage</u>

```
autolib="dlghelp"
ColorDlg()
MESSAGE "Description: "+acceptvalue+"  Code: "+strval(pickvalue)
x=getchar()
```

<u>Screen Capture</u>



```
Color Selections

Enter Color:  Black on Black

Black on Black            Dark Gray on Black
Blue on Black             Light Blue on Black
Green on Black            Light Green on Black
Cyan on Black             Light Cyan on Black
Red on Black              Light Red on Black
Magenta on Black          Light Magenta on Black
Brown on Black            Yellow on Black
Light Gray on Black       White on Black
```

11. ExportDlg()

This procedure displays a special-purpose dialog box that allows the user to specify the filename and directory for the target of a file export activity. In addition, two export "types" can be selected: 1) all fields or 2) just fields for mailing labels. Of course, if you would like to define different export types, you can easily modify the script (again, you need to register in order to edit any script). Different export types may be necessary if you need to transfer different subsets of data other users.

There are six global variables that are used with **ExportDlg**. The three primary ones correspond to the three controls on the dialog box: 1) *exportfilename* contains the name of the file entered in the first type-in box (up to 8 characters), 2) *exportdirectory* contains the name of the directory entered in the second type-in box (up to 40 characters), and 3) *exporttype* contains the number that refers to which export-type radio button was selected (1 refers to the first radio button, 2 the second, and so on). Once you have these values you can perform your export operation. If you wish to assign default values to any of these variables prior to calling this procedure, you can do so.

There are two pushbuttons: OK and CANCEL. As with the other dialog boxes, pressing OK accepts the entries and assigns the variables. Pressing CANCEL leaves the variables in the same state as before the procedure was called. The global variable *buttonvalue* is assigned the value "OK" or "" (for CANCEL) based upon which button is pressed.

Finally, if you wish to control the position of the export dialog box on the screen you can use the global variables *dlgRow* and *dlgCol*. By default, the dialog box will be centered, however, you can assign your own values to these variables prior to calling this procedure. In addition, if the dialog box is moved on the screen, the values of these variables are dynamically updated to reflect the current and most recent position. Thus if you use the following code, you can both specify the initial position of the dialog box and allow the user to move it (subsequent calls to this procedure will cause the dialog box to reappear in its last location).

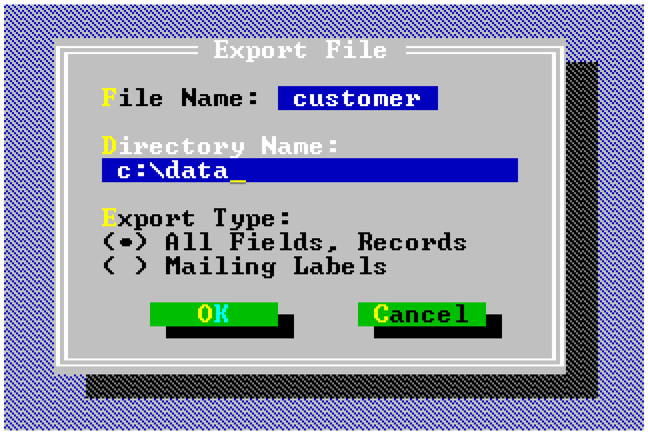
Note: If your program already uses any of these global variables, conflicts may occur. You should either change your variables (preferable) or edit the **ExportDlg** script (which requires a password that you receive upon registering).

Example Usage

```
autolib="dlghelp"
if not isassigned(dlgRow) then ;this must be the first running of the script
  dlgRow=4   ; these coordinates are arbitrary
  dlgCol=20
endif
ExportDlg()
; at this point dlgRow and dlgCol will contain the most recent coordinates
; of the dialog box.  If you rerun the script, then it should reappear in its
; most recent position.  This is true only as long as the values of dlgRow and
; dlgCol are not "released" either by using the RELEASE VARS statement or by
; limiting their scope in a way that results in their values not being
; maintained between calls to the ExportDlg procedure.

; what follows is an example of what you could do after calling ExportDlg()
if buttonvalue="OK" then
      switch
      case exporttype=1: PLAY "Query1"  ; result is in Answer table
      case exporttype=2: PLAY "Query2"  ; result is in Answer table
      endswitch
      MENU {Tools} {ExportImport} {Export} {ASCII} {Delimited} {Answer}
      TYPEIN exportdirectory+exportfilename ENTER
endif
```

Screen Capture



```
File Name: customer

Directory Name:
 c:\data_

Export Type:
(•) All Fields, Records
( ) Mailing Labels
```

12. FindRecord()

This procedure is a special case of the **AcceptDlg** procedure with one type-in box and two buttons. It allows the user to search the current field for a value. The name of the current field (as defined in the table structure) will automatically be used as the label for the type-in box. Users can use all of the same special characters and values that are permitted by the Paradox Zoom command (e.g. ".." for partial matches).

There are no parameters, but there are four global variables: *SearchValue*, *ButtonValue*, *dlgRow*, *dlgCol*. *SearchValue* returns the user's entry in the type-in box if the OK button is pressed. If the Cancel button is pressed, then *SearchValue* retains its original value (probably null). You can, if you wish, assign a default value to *SearchValue* prior to calling **FindRecord**.

Note: If your program already uses the variable *searchvalue*, conflicts may occur. You should either change your variable (preferable) or edit the **FindRecord** script (which requires a password that you receive upon registering).

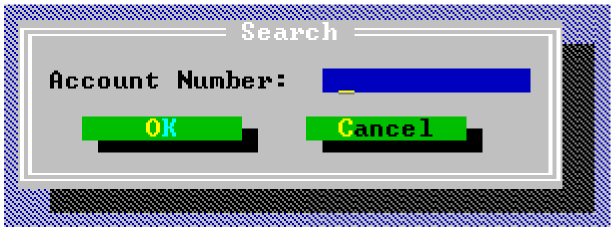
The variable *buttonvalue* is assigned the value "OK" or "" (for CANCEL) based upon which button is pressed.

Finally, if you wish to control the position of the dialog box on the screen you can use the global variables *dlgRow* and *dlgCol*. By default, the dialog box will be centered, however, you can assign your own values to these variables prior to calling this procedure. In addition, if the dialog box is moved on the screen, the values of these variables are dynamically updated to reflect the current and most recent position. Thus if you use the following code, you can both specify the initial position of the dialog box and allow the user to move it (subsequent calls to this procedure will cause the dialog box to reappear in its last location).

Example Usage

```
autolib="dlghelp"
VIEW "Customer"          ; make the Customer table the current image
MOVETO [Account Number]  ; move to the "Account Number" field
if not isassigned(dlgRow) then
  dlgRow=3    ; these coordinates are arbitrary
  dlgCol=15
endif
FindRecord()
; at this point dlgRow and dlgCol will contain the most recent coordinates
; of the dialog box.  If you rerun the script, then it should reappear in its
; most recent position.  This is true only as long as the values of dlgRow and
; dlgCol are not "released" either by using the RELEASE VARS statement or by
; limiting their scope in a way that results in their values not being
; maintained between calls to the ExportDlg procedure.
```

Screen Capture



13. SearchAndReplace()

This procedure is rather sophisticated (at least I'm kind of proud of it). It allows you to search the current field (i.e. the one the cursor is positioned on when this procedure was triggered) for a value and replace it with another value. The user has the option of specifiying case sensitivity and/or full word vs. partial matches. In addition, all matches can be replaced without prompts or can require individual verification by the user. Try it! You'll be impressed also (hopefully it works).

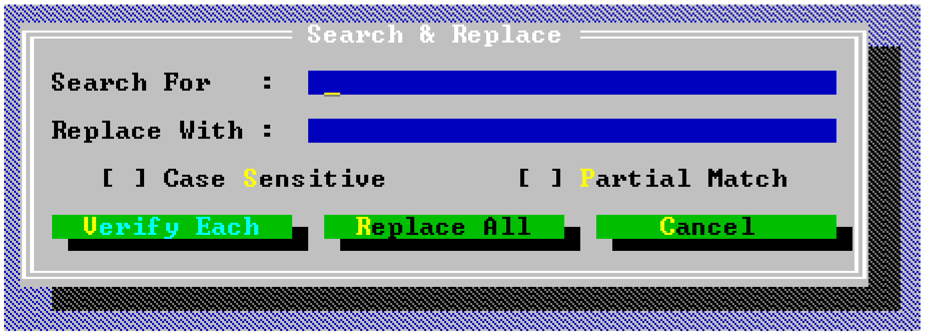
There are no parameters, but there are two global variables that are useful for positioning the dialog box on the screen: *dlgRow* and *dlgCol*. By default, the dialog box will be centered on the screen each time it is called. If you want it to be positioned somewhere else, you can assign values to the variables *dlgRow* and *dlgCol* prior to calling **SearchAndReplace**. Finally, you can also have the dialog box reappear at its last position each time it is called because **SearchAndReplace** updates the values of *dlgRow* and *dlgCol* whenever you move the dialog box on the screen. Thus, if you use the following code, you can specify the initial position of the clock on the screen and then let the program determine future positions based on where it was at last.

The variable *buttonvalue* is assigned the value "OK" or "" (for CANCEL) based upon which button is pressed.

Example Usage

```
autolib="dlghelp"
if not isassigned(dlgRow) then ;this must be the first running of the script
  dlgRow=4   ; these coordinates are arbitrary
  dlgCol=20
endif
SearchAndReplace()
; at this point dlgRow and dlgCol will contain the most recent coordinates
; of the dialog box.  If you rerun the script, it should reappear in its
; most recent position.  This is true only as long as the values of dlgRow and
; dlgCol are not "released" either by using the RELEASE VARS statement or by
; limiting their scope in a way that results in their values not being
; maintained between calls to the SysInfoDlg procedure.
```

Screen Capture



14. SysInfoDlg()

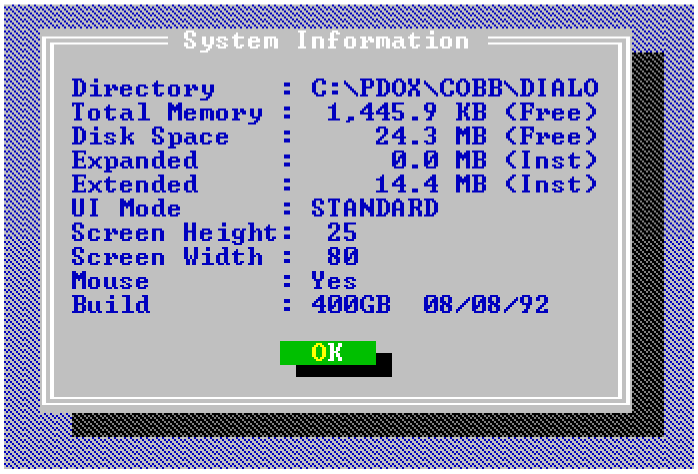
This procedure simply displays selected information about the system. I usually include it as a menu choice under the Help option on my custom menu bar.

There are no parameters, but there are two global variables that are useful for positioning the dialog box on the screen: *dlgRow* and *dlgCol*. By default, the dialog box will be centered on the screen each time it is called. If you want it to be positioned somewhere else, you can assign values to the variables *dlgRow* and *dlgCol* prior to calling **SysInfoDlg**. Finally, you can also have the dialog box reappear at its last position each time it is called <u>because **SysInfoDlg** updates the values of *dlgRow* and *dlgCol* whenever you move the dialog box on the screen</u>. Thus, if you use the following code, you can specify the initial position of the clock on the screen and then let the program determine future positions based on where it was at last.

<u>Example Usage</u>

```
autolib="dlghelp"
if not isassigned(dlgRow) then ;this must be the first running of the script
  dlgRow=4   ; these coordinates are arbitrary
  dlgCol=25
endif
SysInfoDlg()
; at this point dlgRow and dlgCol will contain the most recent coordinates
; of the dialog box.  If you rerun the script, it should reappear in its
; most recent position.  This is true only as long as the values of dlgRow and
; dlgCol are not "released" either by using the RELEASE VARS statement or by
; limiting their scope in a way that results in their values not being
; maintained between calls to the SysInfoDlg procedure.
```

<u>Screen Capture</u>



```
Directory     : C:\PDOX\COBB\DIALO
Total Memory  :  1,445.9 KB (Free)
Disk Space    :     24.3 MB (Free)
Expanded      :      0.0 MB (Inst)
Extended      :     14.4 MB (Inst)
UI Mode       : STANDARD
Screen Height:  25
Screen Width  :  80
Mouse         : Yes
Build         : 400GB   08/08/92
```

Appendix A - Color Codes and Descriptions

These color code values and descriptions can be used in assigning values to the *textcolor* parameter for some of the procedures.

0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	Light Gray
8	Dark Gray
9	Light Blue
10	Light Green
11	Light Cyan
12	Light Red
13	Light Magenta
14	Yellow
15	White

Appendix B - Demo Program

Due to several requests, **dlgHELP** now contains a demo program to illustrate its features. The demo program is contained in the **dlgdemo.zip** file which can be 'unzipped' in the **dlgHELP** directory or in another directory.

dlgdemo.zip contains the following files:

accept.db	
accept.px	
accept.f1	
check.db	
check.px	
check.f1	
customer.db	
demobox.db	
demobox.f1	
demobox.f3	
demobox.f2	
demobox.f4	
demobox.f6	
demobox.f5	
dlgdemo.sc	- Play this script to run the demo
dlghelp.lib	- This is the same library file that comes with **dlgHELP**. It is used by the demo.
label.db	
label.px	
label.f1	
msgline.db	
msgline.px	
msgline.f1	
picklist.db	
picklist.px	
picklist.f1	
readme2.txt	

In addition to demonstrating the tools and utilities of **dlgHELP**, you can also enter your registration password from the demo main menu. However, the demo files must be in the same directory as your **dlgHELP** library scripts.

This is not intended to be an 'airtight' or polished Paradox application. While it is very functional, it is not user-proof.

Appendix C - Revision History

1.0a 26 Sep 1995

dlgHELP Library

Fixed **register.sc**. It was looking to **dialog.db** (which didn't exist) for list of script names to unprotect instead of **dlghelp.db**.

Fixed **SearchAndReplace** utility. It inaccurately reported that system mode was not CoEdit or Edit when in fact it was. You must be in CoEdit or Edit mode to run this utility.

Fixed **CheckDlg** tool. There was a minor problem declaring the global variable *checkvalue[]* which returns the selected checkbox values.

Added **dlgdemo.zip** file, which contains files for a demo of **dlghelp.lib**.

Manual

Corrected inaccuracy concerning value of *buttonvalue*. *Buttonvalue* does contain the value of the button label that was pressed, EXCEPT when that button is a CANCEL button. For those dialog boxes that can have varous numbers of pushbuttons, the LAST button is always a CANCEL button (regardless of the label you assign to it). If the CANCEL button is pressed, *buttonvalue* does not equal "Cancel" (as was reported earlier). It is empty (i.e. *buttonvalue* = "") .

Minor changes in wording to improve clarity of some explanations.

1.0 9 Sep 1995

Original Manual