INTRODUCTION

Welcome to PDOXLIB!

PDOXLIB is a collection of more than 40 Paradox routines written entirely in PAL Version 3.5.

The routines range from the simple to the complex and from the workhorses to the fun.

You'll find lots of routines that prompt the user for information in different ways. You'll find routines that format data and ones that remove or replace unwanted characters.

But you'll also find a handy gauge routine that keeps the user informed of the progress of your calculations. A crawl routine prints an attractive message across the bottom of the screen.

There's even a pop-up calendar written entirely in PAL! Few busy programmers would take the time to write such a routine, but it's the type of finishing touch that adds class and convenience to your hard-working application.

PDOXLIB routines can be used as is or the source code can be modified to meet your own needs. The code is liberally commented, so making your own changes is easy.

But remember, PDOXLIB is shareware. If you use it you have a responsibility to pay for it. There are plenty of good reasons to register PDOXLIB, but the point of shareware is to give you the chance to try out the software and see if it meets your needs. If so, then it's time to put the check in the mail.

WHY SHOULD I REGISTER PDOXLIB?

The best reason to register PDOXLIB is that it's the right thing to do. Hard work went into the creation of PDOXLIB and just as you wouldn't want your work stolen, neither does the author of PDOXLIB.

But from a practical standpoint, why not register it? The cost is only $20 plus postage and handling. Compared to the cost of writing one or two of these routines yourself, it's downright cheap.

The source code alone is worth the price. You may also find you love one of the routines, but hate the choice of colors. With the source code, the full range of available colors is open to you.

From a practical standpoint, you'll find the PDOXLIB library to be quite large. In fact, it's really too large for practical use. Adding the library to an existing application would slow your program down as the computer searches through another 100,000 bytes looking for a routine.

With the source code, you add only the routines you need to your libraries and keep your

programs running at their peak speeds.

So put PDOXLIB through its paces and if it works for you, then drop a check in the mail.

We'd also welcome your comments, suggestions for changes in the routines or requests for new routines.

And if you don't like a routine, tell us that too. We want PDOXLIB to be a helpful addition to your toolbox.

And if you're a beginning PAL programmer, reading the PDOXLIB source code can spur your creativity to write routines of your own.

PDOXLIB is distributed with the source code, but it is password protected. When we receive your payment, we will provide you with the password that will give you access to the source code.

HOW DO I REGISTER PDOXLIB?

First run the routine code () that is in the PDOXLIB library. The routine provides you with a code word to give us that tells us which password unlocks your files.

Do this by selection miniscript from the debug menu (ALT-F10), then typing the following:
autolib = "pdoxlib" code ()

The press the ENTER key and the code word should appear on your screen.

Why the code? We are tracking which BBS systems give us best distribution and buyers. The code tells us on which BBS your version of PDOXLIB was originally uploaded. Since each BBS we used has a different password, it's important for us to know it.

Send us $20 plus the code word that prints on your screen. Upon receipt of your payment, we'll send you the password. If you wish to have the password telephoned to you or have it sent via CompuServe or Exec-PC, please provide the appropriate telephone numbers, mail addresses, and log-on names we need to contact you.

If you register by mail, you may use the form file included.

atprint

Description: Prints a string of text in a specified attribute at a row and column position on the canvas.

Syntax: atprint.u(text.a, attrib.n, row.n, col.n)

Where: text.ais the text to be printed
attrib.nis the attribute number for the text.
row.nis the row where the text is to be printed.
col.nis the column where the text is to be printed.

While any beginning PAL programmer can handle the writing of this code, this routine provides a code-reusable way of handling a regular chore. In fact, the process is so routine many programmers would not even think of writing such a routine.

bgetdate

Description: Prompts the user for a date via the dialog box. The procedure uses PAL's ACCEPT statement for getting data.

Syntax: newdate.d = bgetdate.d ()

Where: newdate.dreceives a new, valid date

This routine pops a dialog box into the center of the screen and prompts the user to enter a day in the MM/DD/YY format. Because ACCEPT is used, only a valid date may be entered.

bgetdate2

Description: Prompts the user via a dialog box for a date in the MM/DD/YY format. It uses a more elegant way of gathering the data than does the PAL ACCEPT command.

Syntax: newdate.d = bgetdate2.d ()

Where: newdate.dreceives a new, valid date

Although Paradox provides validation checks for field data entry, there are times, such as when prompting for query information, where a dialog box is helpful. This routine pops a box onto the center of the screen and prompts the user to enter a day in the MM/DD/YY format. The routine will only accept valid dates, although the user may press ESC to leave the routine.

bgetphone

Description: Prompts the user for a telephone number via a dialog box on the screen. It uses the PAL ACCEPT command and requires an area code entry. The routine returns a string in the format: NNN-NNN-NNNN.

Syntax: phonenumber.a = bgetphone.a ()

Where: phonenumber.areceives a formatted telephone number

Although Paradox provides validation checks for field data entry, there are times, such as when prompting for query information, where a dialog box is helpful. This routine pops a box onto the center of the screen and prompts the user to enter a telephone number in the NNN-NNN-NNNN format.

bgetphone2

Description: Prompts the user for a telephone number via a dialog box on the screen. It uses a more visual way of gathering the information than is possible with the PAL ACCEPT command.

Syntax: phonenumber.a = bgetphone2.a()

Where: phonenumber.areceives a formatted telephone number

Although Paradox provides validation checks for field data entry, there are times, such as when prompting for query information, where a dialog box is helpful. This routine pops a box onto the center of the screen and prompts the user to enter a telephone number in the (___) ___-____ format. The user may press ESC to leave the routine.

bgetssn

Description: Prompts the user for a social security number via a dialog box on the screen. It uses the PAL ACCEPT command. The routine returns a string in the format: NNN-NN-NNNN.

Syntax: ssn.a = bgetssn.a ()

Where: ssn.areceives a formatted social security number

Although Paradox provides validation checks for field data entry, there are times, such as when prompting for query information, where a dialog box is helpful. This routine pops a box onto the center of the screen and prompts the user to enter a social security number in the NNN-NN-NNNN format.

bgetssn2

Description: Prompts the user for a social security number via a dialog box on the screen. It uses a more visual way of gathering the information than is possible with the PAL ACCEPT command.

Syntax: ssn.a = bgetssn2.a ()

Where: ssn.areceives a formatted social security number

Although Paradox provides validation checks for field data entry, there are times, such as when prompting for query information, where a dialog box is helpful. This routine pops a box onto the center of the screen and prompts the user to enter a social security number in the NNN-NN-NNNN format. The user may press ESC to leave the routine.

calcage

Description: Using a date of birth, the routine returns the person's age.

Syntax: currentage.n = calcage.n (dob.d)

Where: dob.dis a date variable representing the person's date of birth.
currentage.nreceives the age calculated from the date of birth

The routine can be used when an age instead of a date is needed. It can also be used to fill in an age field if only a date of birth field exists. If procedures are allowed in forms in future versions of Paradox, the routine might also be useful to keep an accurate running age of a person when only the date of birth is available.

copyfile

Description: Copies a file to a new disk or directory.

Syntax: copyfile.u ()

This routine can be used to copy an individual file to a disk or new directory. If used on a table, it will not copy the related family. The PAL COPY command should be used for that purpose. However this does provide an easy way of copying a file without overtly shelling out to DOS. The routine provides a Paradox-style menu of files in the current directory, asks for a destination, checks to see if the destination is valid, then uses DOS to actually copy the file. The routine does not check to see if there is sufficient space to copy the file, however.

crawl

Description: Displays a message that crawls across the bottom of the screen to let the user know that work is in progress.

Syntax: crawl.u(message.a)

Where: message.ais the message to be displayed on the craw

Crawl is intended to a included in a loop such as WHILE or FOR. With each pass through the loop the program determines the current position of the message and advances it one column.

For example:

```
While true
  crawl.u("Processing data")
  ....
  ....
  ....
Endwhile
```

While the program is relatively speedy, it does include a short pause to make the message readable. For short processing times, the routine should not pose a problem, but if the loop is expected to take some time, crawl.u will probably provide too much of a delay.

delrec

Description: A routine to delete the record currently hosting the cursor.

Syntax: delrec.u ()

The routine will delete the current record after prompting the user to confirm that choice. The routine can be called from main, edit or coedit modes. If called in main mode, the routine will return the user to main mode at the conclusion.

One advantage of the routine is that is saves the most recently deleted record in memory. By using the companion routine, saverec.u, the record can be restored. This is true even if the deletion was made in coedit mode, the edit session was ended with F2 and the table was removed from the screen.

Only the most recent deletion is held in memory and that is lost when the user exits Paradox.

fullname

Description: Returns the full path name of the current directory.

Syntax: pathname.a = fullname.a(filename.a, styl.n)

Where: filename.ais the name of a selected file in the current directory
styl.nselects single or double backslashes
pathname.areceives the full path name

When changing directories within an application, it is often useful to retain valid path names for calling tables or reports later. This utility allows you to capture that information in a variable.

Instead of the filename.a, the user can use the PAL function TABLE (). The routine will then return the full path name of a table, but without the .DB file extension. Of course the table must be on the workspace for the TABLE () function to work properly.

Some Paradox functions also require a double backslash in which to work properly. By entering a value of zero, the function will return the path with double backslashes. Any other value will return single backslashes.

getdate

Description: Prompts the user for a date at the current cursor position.

Syntax: newdate.d = getdate.d()

Where: newdate.dreceives a date variable in the MM/DD/YY format

Using the ACCEPT command, getdate.d prompts the user for a date. Care must be taken before calling getdate.d to place the cursor in the correct position. The routine offers a plain vanilla way of getting a date. For a more elegant way, see getdate2.

getdate2

Description: A more visual way of getting a date from the user at the current cursor position.

Syntax: newdate.d = getdate2.d ()

Where: newdate.dreceives a date variable in the MM/DD/YY format

Unlike getdate.d, getdate2.d provides an image on the screen for the user to complete. By providing that image, it helps the user determine how the date should be entered. The routine checks to see if the date is valid before allowing the user to leave, however the user may press ESC to leave the routine. If so, the routine returns "ESC" to newdate.d.

getphone

Description: Prompts the user for a telephone number at the current cursor position.

Syntax: newnumber.a = getphone.a ()

Where: newnumber.areceives a telephone number in the ###-###-#### format.

The routine uses PAL's ACCEPT command to return a telephone number.

getphone2

Description: Provides a more visual way of prompting the user for a telephone number at the current cursor location.

Syntax: newnumber.a = getphone2.a ()

Where: newnumber.areceives a telephone number in the ###-###-#### format.

getphone2.a provides a more visual way to prompt the user for a telephone number. The routine requires numeric input, but will return ESC if the user escapes out of the routine. The routine could be easily modified to allow a default area code, thus speeding data entry.

getnum

Description: Prompts the user to enter a number at the current cursor location.

Syntax: newnumber.n = getnum.n ()

Where: newnumber.nreceives a number typed in by the user.

This is a simple ACCEPT version of getting a number from the user. Since the style or format of the number is unknown, no formatting or validity checking can be done. If the user leaves the routine without entering a number or by pressing ESC, the routine returns false.

getssn

Description: Provides a simple way of prompting the user to enter a social security number at the present cursor location.

Syntax: ssn.a = getssn.a ()

Where: ssn.areceives a social security number in the ###-##-#### format.

Using PAL's accept command, this routine provides a quick way of getting a social security number. ESC is returned if the user escapes out of the routine.

getssn2

Description: Provides a more visual way of prompting the user to enter a social security number at the current cursor location.

Syntax: ssn.a = getssn2.a ()

Where: ssn.areceives a social security number in the ###-##-#### format.

Provides a more visual way of prompting the user for a social security number. The routine puts an image of the number in the current cursor location and accepts numbers from the user. ESC is returned if the user presses the escape key.

gettext

Description: Gets a line of text at the current cursor location and formats it to all upper case, lower case or capitalizing the first letter of each word.

Syntax: newtext.a = gettext.a(length.n, styl.n)

Where: newtext.areceives a formatted string of text
length.nis the maximum length of the string
styl.nis 1, 2 or 3 to indicate the style desired

The routine prompts the user to input a string of text at the current cursor location. If the user exits the routine without entering text or by pressing ESC, the routine returns false.

The length.n variable tells the routine how much text should be received from the user. The styl.n variable should be set to 1 if the text is to be returned in all capitals, 2 if it is to be all lower case or 3 if the first letter of each word is to be capitalized. If any other value is entered, the string is returned unaltered.

The routine also calls the lrtrim.a routine to ensure that newtext.a receives a string without leading or trailing spaces.

hmsgbox

Description: Puts a box on the screen with a headline and a one line message. The user is prompted for a keypress before the routine ends.

Syntax: hmsgbox.u(hed.a, message.a)

Where: hed.ais the headline to be printed at the top of the box
message.ais the one line message to be printed in the box

The routine provides a convenient way of passing a brief message to the user. Both headline and message can be any length, but the message should be longer than the headline.

Once the message is placed on the screen, the user to told to press any key to continue as a way of allowing him or her to control program flow.

hmsgbox2

Description: Puts a box on the screen with a headline and a two line message. The user is prompted for a keypress before the routine ends.

Syntax: hmsgbox2.u(hed.a, line1.a, line2.a)

Where: hed.ais the headline to be printed at the top of the box
line1.ais the first line to be printed in the box
line2.ais the second line to be printed in the box

The routine provides a convenient was of passing a two-line message to the user. Both line1.a and line2.a can be of any length, but at least one of the lines should be longer than the headline.

Once the message is placed on the screen, the user is told to press any key to continue as a way of allowing him or her to control program flow.

insertrec

Description: Allows the user to enter a new record.

Syntax: insertrec.u ()

The routine works from main, edit or coedit modes and opens a new record for the user to enter.

The routine puts the new record into a WAIT RECORD and prompts the user to press F2 when the entry is complete. The routine also traps for DOS and DOSBIG to prevent accidental use of those commands.

isdatevalid

Description: Checks a date to see if it is valid.

Syntax: testdate.l = isdatevalid.l(date2test.a)

Where: testdate.lreceives true if the date is good or false if it is not.
date2test.dis the date to be examined

The routine takes a string of text entered in a valid Paradox date format and checks to see if it is a valid date.

Often data entry routines, such as the getdate2.d routine in this package, gathers the date as a string, then converts it to a date. Although the date routines in this package check for valid dates, this routine could be useful if you used other ways to get date information.

The routine returns a true if the date is valid and false if it is not.

It is important that the date be passed to the routine as a quoted string or in a string variable. Use of the STRVAL() function will fail if the date is not valid and cause a script error.

keygen1

Description: Generates a random number of up to five digits for use as a key. Routine also checks key field to make sure key number is not in use.

Syntax: newkey.n = keygen1.n (field.a)

Where: newkey.nreceives a unique number that can be used as a key
field.ais the name of the key field in the current table

When the nature of the key is not important, keygen1.n can provide a unique number quickly and easily.

The routine can be called from any field and will search the key field to see if the number generated exists. If it does exist, the routine will continue generating numbers until it finds one not in use.

The routine makes no network provisions, so care should be taken to lock the number into the field as soon as it is returned as possible.

keygen2

Description: Generates a number one larger than the largest number in the key field.

Syntax: newkey.n = keygen2.n(field.a)

Where: newkey.nreceives a number one larger than the largest number in the key field.
field.ais the name of the key field in the current table

The routine can be called from any field and will find the largest number in the key field, increment it by one and return it as the new key.

No network provisions are provided in this routine, care should be taken to lock the number into the table immediately after the number is returned.

If no number is in the key field, the program returns 100000. Nearly 900,000 records can be handled in this way.

lastfirst

Description: Takes a name in the form of John Doe and returns Doe, John.

Syntax: newname.a = lastfirst.a (fullname.a)

Where: newname.ais the last name followed by a comma and the first name
fullname.ais the full name to be affected

The routine will handle name suffixes such as Jr., Sr., II, III or IV.

lastlast

Description: Takes a name in the form of Doe, John and returns John Doe.

Syntax: newname.a = lastlast.a(fullname.a)

Where: newname.ais the full name of the person
fullname.ais the name in last first form

The routine makes allowances for middle initials and simply searches for the comma. Once the comma is found, the new name is built accordingly.

lrtrim

Description: Trims leading and trailing blanks from a string.

Syntax: newstring.a = lrtrim.a(teststring.a)

Where: newname.ais a string with leading and trailing blanks removed
teststring.ais a string that may contain leading or trailing blanks

Particularly when importing fixed length data, blanks can be part of the string. This routine is used to remove those ASCII 32 characters.

The routine can be part of a PAL program or can be used within a scan loop in a miniscript to clean up data in a table.

ltrim

Description: Trims leading blanks from a string

Syntax: newname.a = ltrim.a(teststring.a)

Where: newname.ais a string with leading blanks removed
teststring.ais a string that may contain leading blanks

The routine removes all ASCII 32 characters from the front of a string. When only leading blanks exist, the use of ltrim.a will be faster than lrtrim.a.

msgbox

Description: Puts a one line message on the screen and waits for a keypress to continue.

Syntax: msgbox.u(message.a)

Where: message.ais the message to be displayed

The routine provides a quick way to pass information on to the user. The routine includes a pause that requires the user to press a key to continue.

The pause gives the user the chance to read the message before moving on. Should a keypress be unacceptable, replacing the event with a sleep command is easily accomplished by editing the source code.

msgbox2

Description: Puts a two line message box on the screen and waits for a keypress before continuing.

Syntax: msgbox2.u(line1.a, line2.a)

Where: line1.ais the first line of text to be displayed
line2.ais the second line of text to be displayed

The routine provides a quick way of passing a two line message on to the user. The routine includes a pause that requires the user to press a key to continue.

namekey

Description: Generates a unique key based on the first and last name of a person.

Syntax: newkey.a = namekey.a()

Where: newkey.areceives a unique key based on a first and last name.

One of the problems with a numeric key is that it does not sort the names alphabetically. That means users have no clue as to where a name will occur in a database and must constantly use locate commands.

The namekey.a routine assumes there are two fields in the current table named "last name" and "first name." From those it constructs key of no more than 12 characters.

The key takes the first eight letters of the last name and the first four letters of the first name and puts them together. In most cases this will provide an accurate sorting. The problem comes when two people of the same name are entered into the database.

To defeat this problem, the routine goes to the first field in the database -- always the key field -- and checks to see if the newly created key exists. If so, it generates a random two-digit number and replaces the last two characters of the name field with it.

For example, if you have a person in your database by the name of Howard Johnston, the original key would be: JohnstonHowa. If a second Howard Johnston were added, the key might look like: Johnst24Howa.

While this could cause some alphabetizing problems, the solution would be to have a key field wide enough to accommodate any possibility. Since Paradox works best with narrow tables, the routine was set for a maximum of 12 characters. This still allows for up to 100 people (00 through 99) with the same name.

The routine makes no provision for network use and can only be called after the last name and first name fields have been filled. Care should be taken to post the key as soon as possible after the routine has been called to avoid potential conflicts on a network.

Changes can easily be made in the source code in case the last name and first name fields have different names.

numbox

Description: Puts a dialog box with headline on the screen so the user can enter a number.

Syntax: newnumber.n = numbox.n(headline.a)

Where: newnumber.nreceives the number entered by the user
headline.ais the headline to put on the box

The routine puts a dialog box on the screen and allows the user to enter a number. Space for a headline is provided to prompt the user for what number to enter. The box is sized to accommodate the length of the headline.

numonly

Description: Searches through a string and removes any character that is not a number except a period.

Syntax: newnumber.n = numonly.a(numberstring.a)

Where: newnumber.nreceives a number as a string variable
numberstring.ais a string which may contain both letters and numbers

There are times when non-numerical data needs to be stripped out so that a number can be placed either into a numeric variable or a numeric field.

The numonly.a routine strips out all string values except periods. A change in the source code can allow for the stripping of periods also.

An example of this use might be to strip the dashes out of a telephone number or social security number so the data can be stored in a smaller numeric field. Of course the dashes would have to be reinserted when the data is read, but the savings in disk space may be worth the extra processor time in doing so.

Note that the routine returns the data as a string. It can easily be converted to a number with the PAL NUMVAL function.

popcal

Description: A pop-up calendar program written entirely in PAL.

Syntax: popcal.u ()

The routine puts a calendar showing the current month in the upper right hand side of the screen.

The user may press the left and right arrow keys to move forward or behind in months. The user presses the ESC key to leave the routine. The system date must be accurately set for the routine to work correctly.

The routine was intended to be called from a SETKEY command, however it can just as easily be called from a program when the user is asked to input a number.

popval

Description: A pop-up valcheck menu that allows the user to select an appropriate valcheck for the current field.

Syntax: popval.a ()

The routine presents a menu of 22 possible valchecks and sets the valcheck for the current field.

The user presses the arrow keys to move the light bar to the appropriate valcheck, then presses ENTER to select it.

Once selected the routine sets that valcheck for the current field. The user must be in edit mode for the routine to work.

remblank

Description: Removes all spaces from a string.

Syntax: newstring.a = remblanks.a(oldstring.a)

Where: newstring.areceives a string without blank spaces
oldstring.ais a string that may contain blanks

Removes all ASCII 32 characters from a string.

restorerec

Description: Restores last record deleted using the delrec.u routine.

Syntax: restorerec.u ()

The most recent record deleted using the delrec.u routine is held in an array in memory until the user leaves Paradox.

The restorerec.u routine will restore that record even if the editing session has ended and the user is in main mode. The table must be on the workspace at the time the restoration is requested, but it may have been removed previously.

The routine will work only for the most recent deletion.

rtrim

Description: Trims trailing spaces from a string.

Syntax: newstring.a = rtrim.a (oldstring.a)

Where: newstring.areceives a string without trailing spaces
oldstring.ais a string that may have trailing spaces

The routine will trim all trailing ASCII 32 characters from a string. It will be faster than lrtrim.a if only trailing are expected.

statusbar

Description: A moving bar and message that graphically shows the user how far a process has progressed.

Syntax: statusbar.u(msg.a, row.n, col.n, length.n, counter.n, start.n, end.n)

Where: msg.ais the message to display above the bar
row.nis the row where the bar is to be displayed
col.nis the column where the bar is to begin
length.nis the length of the bar
counter.nis the counter position in the loop
start.nis the beginning of the loop
end.nis where the loop will end

Despite the number of variables required in a call to statusbar, it is very easy to use.

The routine is intended to be used in a FOR/NEXT or WHILE/ENDWHILE loop where the number of passes through the loop are known.

Here is a sample of how statusbar.u might be used:

```
FOR x from 1 to 500
statusbar.u("Processing data", 5, 1, 20, x, 1, 500)
....
....
....
ENDFOR
```

The routine will calculate the percentage of how far the loop the counter gone, apply that percentage to the length of the bar, then display the bar. With each pass through the loop, the calculation is updated, allowing the use of the STEP statement in the FOR loop.

While the routine will handle any length for the bar up to one screen width.

striphigh

Description: Strips all ASCII characters above 126 out of a string.

Syntax: newstring.a = striphigh.a(oldstring.a)

Where: newstring.areceives a string of ASCII characters below 127
oldstring.ais a string that may contain ASCII characters above 126

The routine will strip all high ASCII characters out of a string that may cause problems with some printers.

textbox

Description: Prompts the user to enter a line of text and formats it to all caps, all lower case or with only the first letter of each word capitalized.

Syntax: newstring.a = textbox.a(headline.a, length.n, styl.n)

Where: newstring.areceives a formatted string of text
headline.ais the headline to place on the box
length.nis the number of characters to be entered
styl.nis the style to format the new text

Using a dialog box, textbox.a prompts the user for a one line text string. A headline can be put on the box to prompt the user as to what text to enter. The length.n variable determines how long the string can be.

If styl.n is set to 1, the text will be returned to newstring.a in all capitals. If it is set to 2, the string will be returned in all lower case. If it is set to 3, the first letter of each word will be capitalized. If any other number is used, the text will be returned unaltered.

The routine also uses the lrtrim.a function to trim any leading or trailing from the string.

If the user exits without entering any text, the routine returns a value of false.

waitkeypress

Description: Creates a pause until the user presses a key.

Syntax: waitkeypress.v ()

or

charpicked.a = waitkeypress.v ()

Where: charpicked.ais the key pressed

The routine is used to halt program execution until a key is pressed. If knowing the key is important, the function version of the routine can be used.

wipes

Description: A demonstration program to show how wipes can be added to a program.

Syntax: wipes.u ()

Wipes is included only to show that wipes can be done with PAL. The source code shows how wipes can be written into programs.