TECHNOTES

TECHNICAL SUPPORT NEWSLETTER

dBASE IV
No. 87 May, 1992

FOR THE USERS OF dBASE IV

Q&A

From your BBS, I've downloaded a utility which consists of a couple of assembly-language files (.BIN) but I don't understand how to prepare them for use with dBASE IV. Do I link or compile them in some way?

.BIN files don't need *preparation*. In your dBASE IV program, use the LOAD command to load a .BIN file into memory. You may have up to 16 loaded into memory at once. Use the CALL command or the CALL() function to run the desired .BIN file. If you need the .BIN routine to return a value, the CALL() function is preferred. When you want to release them from memory, use the RELEASE MODULE command.

More Q&A on page 10

FEATURED IN THIS ISSUE

DEPARTMENTS

The Undocumented Mouse

BILL RAMOS

There is undoubtedly much to be said about the "pet of choice" for PC users now that version 1.5 has arrived

dBASE IV version 1.5 represents the first time you can "mouse around" inside of dBASE IV without an external utility. But the use of a mouse had to have as small an impact on the core of dBASE IV as possible; so don't look for sizable BROWSE windows, scroll bars, and clipboards. What you will find is a no-nonsense implementation of the mouse that allows you to click through menu pads, popups, BROWSE, EDIT, READ and WAIT commands.

There is more to say about mouse implementation than what is within the pages of the documentation. In this article you'll go beyond learning how to use the mouse in your programs. You'll also find two .BIN files and several UDFs that will give your programs more control over the mouse.

Making Your Programs Mouse Friendly

A typical application written in dBASE IV involves a number of diverse elements. But before you can use the mouse with your programs, you have to know where the traps are.

Pause Until Click

As dBASE programmers know, a common practice for pausing in a program is to use the INKEY() function. Unfortunately, INKEY() does not respond to a mouse click. Hold on, the WAIT command does. PROCEDURE Err_Box in the dBASE sample program LIBRARY.PRG demonstrates how you can display a message in a box and allow the user to either press a key or click the mouse to continue processing.

Continued on page 3

BORLAND

Making Popup Selections

Another common programming practice that can get you into trouble is using LASTKEY() after handling an ON SELECTION POPUP event to see if the user selected a POPUP bar. However, LASTKEY() does not change its value after a mouse click. If you click outside the POPUP, which is the same as pressing the Esc key, your program is not going to handle the Escape event correctly. Using the BAR() function is a more precise way of handling this. Change your code from this:

```
ON SELECTION POPUP foo DEACTIVATE POPUP
ACTIVATE POPUP foo
IF LASTKEY() = 27      && Esc pressed.
* - No selection made handling code here.
ENDIF
```

to this:

```
ON SELECTION POPUP foo DEACTIVATE POPUP
ACTIVATE POPUP foo
IF BAR() = 0    && No bar is selected
* - No selection made handling code here.
ENDIF
```

Of Mice and Edit Screens

Toggling Menu Choices One technique that is not so obvious is using the @M picture function on a GET field. Once the user has entered the field, each mouse click will toggle just as the Spacebar does.

Valid Required and When Clauses As you click around a data entry screen within Edit, Browse or Read, the mouse emulates the Tab or Shift-Tab keys to move forward or backward through the screen. If there are any GETs with WHEN, VALID REQUIRED, and/or RANGE REQUIRED clauses between the current field and the destination field, dBASE IV will execute them to insure data integrity. This can get you into trouble. Suppose you have a WHEN clause on a GET statement to display a pop-up of choices for the user when they enter the field. However, you may not want the popup to appear if they click past it.

Fortunately, dBASE IV does not physically move through the field. If the ROW() and COL() function values in the WHEN clause are not within the coordinates of the field, that logically means the user clicked past the field. You can use this same technique to avoid VALID or RANGE REQUIRED checks.

Turning the Mouse on and off (*or* Insider Trading)

Many people think that turning off the mouse cursor is like turning off the keyboard. You may want to turn off the mouse for applications that are not ready for it. Is there no way to do it short of disabling the mouse driver *before* going into dBASE IV and turning it back on when you leave?

Well, the key to overcoming this inconvenience is found in the small program called MouseMod.PRG. You see, every time dBASE IV waits for a keystroke, it makes the mouse cursor visible. When dBASE IV runs a command with no user interaction, it sets an internal mouse flag variable off and hides the cursor. Upon return, dBASE IV turns it back on and the mouse cursor reappears. To toggle the mouse off or on, pass the parameter "ON" or "OFF" when calling the program. For example, to turn the mouse off:

```
DO MOUSEMOD WITH "OFF"
```

A few cautions for using MouseMod.BIN: You should not attempt to turn on the mouse cursor if the mouse driver is not installed. Second, do not shut off the mouse cursor in a UDF or ON KEY procedure within EDIT or BROWSE. These full screen commands use the internal flag in a different way.

Capturing the Mouse Row and Column

To take full control over the mouse, your programs really need to first know if the mouse is active, and if it is, know where the mouse cursor is. GetMouse.BIN along with the UDFs IsMouse(), MRow(), and MCol() make this easy. A short cut program to create GetMouse.BIN within dBASE IV is listed at the end of this article.

The idea behind the IsMouse() function is to initialize the row and columns to a value that cannot be valid. If GetMouse.BIN sees that the mouse is not active, it will not change the parameter values.

The MRow() and MCol() functions assume that you have already loaded GetMouse.BIN into memory. That way, they do not have the overhead associated with the LOAD and RELEASE MODULE commands on each call to the UDFs.

Simulating Control Buttons in Edit

Here is an example of how to add control buttons to your data entry forms that will respond to mouse clicks. VendCond.PRG will call EDIT using the format file Vendor_M.FMT. There are four control buttons at the bottom of the screen that will Page Up, Page Down, Exit and Save, and Cancel changes for the record.

VendCont.PRG also contains the UDFs ValidStat() and WhenStat() to handle the VALID and WHEN actions on the control buttons.

Making the .BIN Files

The two small programs below, MakeMMod.PRG and MakeGetM.PRG will be essentially one-time programs that create the MouseMod and GetMouse .BIN files. By redirecting the printer output to a file and issuing several lines of cryptic ??? commands (they don't start with three question marks for nothing), you can create the .BIN files necessary to control the mouse status and get mouse coordinates. If you cannot obtain these .BIN files electronically from the download machine or Borland BBS, take care to check and double check your data entry where the series of curly-braced numbers are concerned. An incorrect sequence, in the majority of cases, would cause the .BIN file to either fail or lock your computer.

In Summary

By itself, dBASE IV version 1.5 has a lot to offer in the way of mouse support. Information about mouse behavior can be found in Appendix A of *Getting Started with dBASE IV*. With the addition of GETMOUSE.BIN and MOUSEMOD.BIN, you can gain greater control of the mouse in your program. ■

MakeMMod.PRG

```
* MakeMMod.prg - Make BIN file using a PRG
SET PRINTER TO FILE MouseMod.BIN
SET PRINTER ON
??? [{235}{4}{144}{0}{0}{0}{131}{249}]
??? [{1}{116}{14}{38}{197}{93}{4}{138}]
??? [{47}{46}{136}{46}{3}{0}{235}{12}]
??? [{144}{46}{128}{62}{3}{0}{0}{117}]
??? [{72}{235}{92}{144}{46}{128}{62}{3}]
??? [{0}{69}{117}{10}{46}{199}{6}{4}]
??? [{0}{190}{151}{235}{52}{144}{46}{128}]
??? [{62}{3}{0}{197}{117}{10}{46}{199}]
??? [{6}{4}{0}{102}{147}{235}{34}{144}]
??? [{46}{128}{62}{3}{0}{67}{117}{10}]
??? [{46}{199}{6}{4}{0}{204}{151}{235}]
??? [{16}{144}{46}{128}{62}{3}{0}{195}]
??? [{117}{29}{46}{199}{6}{4}{0}{116}]
??? [{147}{38}{197}{29}{139}{15}{232}{15}]
??? [{0}{46}{139}{30}{4}{0}{139}{23}]
??? [{137}{15}{38}{197}{29}{137}{23}{203}]
??? [{139}{220}{131}{195}{6}{142}{31}{195}]
SET PRINTER OFF
SET PRINTER TO
*-- EOP: MouseMod.prg
```

MakeGetM.PRG

```
* MakeGetM.prg - Make BIN file using a PRG
SET PRINTER TO FILE GetMouse.BIN
SET PRINTER ON
??? [{235}{8}{144}{0}{0}{0}{0}{0}]
??? [{0}{0}{131}{249}{2}{116}{14}{38}]
??? [{197}{93}{8}{138}{47}{46}{136}{46}]
??? [{3}{0}{235}{15}{144}{46}{128}{62}]
??? [{3}{0}{0}{117}{3}{233}{175}{0}]
??? [{233}{128}{0}{46}{128}{62}{3}{0}]
??? [{69}{117}{24}{46}{199}{6}{8}{0}]
??? [{190}{151}{46}{199}{6}{4}{0}{202}]
??? [{151}{46}{199}{6}{6}{0}{200}{151}]
??? [{235}{97}{144}{46}{128}{62}{3}{0}]
??? [{197}{117}{24}{46}{199}{6}{8}{0}]
??? [{102}{147}{46}{199}{6}{4}{0}{114}]
??? [{147}{46}{199}{6}{6}{0}{112}{147}]
```

MakeGetM.PRG *continued*

```
??? [{235}{65}{144}{46}{128}{62}{3}{0}]
??? [{67}{117}{24}{46}{199}{6}{8}{0}]
??? [{204}{151}{46}{199}{6}{4}{0}{216}]
??? [{151}{46}{199}{6}{6}{0}{214}{151}]
??? [{235}{33}{144}{46}{128}{62}{3}{0}]
??? [{195}{117}{68}{46}{199}{6}{8}{0}]
??? [{116}{147}{46}{199}{6}{4}{0}{128}]
??? [{147}{46}{199}{6}{6}{0}{126}{147}]
??? [{235}{1}{144}{139}{236}{131}{197}{4}]
??? [{142}{94}{0}{46}{139}{30}{8}{0}]
??? [{138}{7}{60}{0}{116}{25}{46}{139}]
??? [{30}{6}{0}{138}{15}{46}{139}{30}]
??? [{4}{0}{138}{47}{38}{197}{29}{136}]
??? [{15}{38}{197}{93}{4}{136}{47}{203}]
SET PRINTER OFF
SET PRINTER TO
*-- EOP: GetMouse.prg
```

Procedure: MouseMod.PRG

```
PROCEDURE MouseMod
  PARAMETER pc_mode
  PRIVATE lc_ver, ln_ver, ln_mode
  lc_ver = UPPER(VERSION(0))
  ln_ver = VAL(SUBSTR(lc_ver, AT("XX", lc_ver) + 2))
  ln_ver = ln_ver + IIF("RUN" $ lc_ver, 128, 0)
  ln_mode = IIF(UPPER(pc_mode) = "ON", 1, 0)
  LOAD MouseMod
  CALL MouseMod WITH CHR(ln_mode), CHR(ln_ver)
  RELEASE MODULE MouseMod
RETURN
```

Mouse Functions: IsMouse()

```
FUNCTION IsMouse
  *   IsMouse() will call Load and Call GetMouse.BIN to see if the
  *   mouse is active.  If the mouse is active, GetMouse.BIN will
  *   remain loaded.  If not, then IsMouse() will release it from memory.
  *   IsMouse() will also initialize the GetMouse.BIN file in memory.
  *   If you Release Module GetMouse.BIN, you should use IsMouse()
  *   to Load it again.  If the mouse is active, then IsMouse() will
  *    return .T., otherwise it will return .F.
  *-------------------------------------------------------------------------
  PRIVATE ll_result, lc_row, lc_col, lc_ver, ln_ver
  ll_result = .F.
  *-- Determine if this program is running RunTime or normal dBASE
  lc_ver = VERSION(0)
  ln_ver = VAL(SUBSTR(lc_ver, AT("xx",lc_ver) + 2, 2))
  ln_ver = ln_ver + IIF(SUBSTR(lc_ver, 10, 1) = "R", 128, 0)
  STORE CHR(255) TO lc_row, lc_col
  LOAD GetMouse
  ll_result = CALL("GetMouse", lc_row, lc_col, CHR(ln_ver)) < CHR(255)

  IF .NOT. ll_result
    RELEASE MODULE GetMouse
  ENDIF
RETURN(ll_result)
```

Mouse Functions: MRow() and MCol()

```
FUNCTION MRow
  *-- MRow() - Returns the row of the last mouse click
  PRIVATE lc_row, lc_col
  STORE CHR(255) TO lc_row, lc_col
  CALL GetMouse WITH lc_row, lc_col
RETURN(ASC(lc_row))
*-- EOF: MRow()


FUNCTION MCol
  *-- MCol() - Returns the column of the last mouse click
  PRIVATE lc_row, lc_col
  STORE CHR(255) TO lc_row, lc_col
  CALL GetMouse WITH lc_row, lc_col
RETURN(ASC(lc_col))
*-- EOF: MCol()
```

VendCont.Prg

```
PROCEDURE VendCont
  *   VendCont.PRG demonstrates how to simulate control keys on an
  *   EDIT screen.  The control key will perform the desired action
  *   when the user either presses the Enter key or clicks the
  *   mouse in the control key.  This is much like the IBM SAA CUA standard.
  *
  *   IsMouse()      - UDF that returns .T. if mouse is active
  *   GetMouse.Bin   - BIN file that returns the mouse row and column
  *   MRow()         - UDF that returns the mouse row
  *   MCol()         - UDF that returns the mouse column
  *-------------------------------------------------------------------
    gl_ismouse = IsMouse()        && Capture the mouse status
    IF SELECT("VENDORS") > 0      && If the vendor file is open already
      SELECT Vendors              && Put it in use
      SET FIELDS TO               && Clear any existing field list
    ELSE                          && Otherwise
      USE Vendors                 && Open the Vendors file
    ENDIF

    *-- The format file was initially designed with memory variables
    *-- but was changed to fields to make the control fields read-only
    *-- and still allow access to the field.  This is done with the
    *-- following SET FIELDS to commands:

    SET FIELDS TO ALL             && Pull in the Vendor fields
    SET FIELDS TO   PgUp = "<PgUp>", ;
                    PgDn = "<PgDn>", ;
                    Ctrl_End = "<Ctrl_End>", ;
                    Esc = "<Esc>"     && Assign the control fields
    SET FORMAT TO Vendor_M
    EDIT
    IF gl_ismouse                 && If the mouse is on
      RELEASE MODULE GetMouse     && Release the GetMouse BIN file
    ENDIF
RETURN
```

Function: ValidStat()

```
FUNCTION ValidStat
  *   ValidStat() is the validation handler for control fields.  It's
  *   job is to determine the action if the Enter key was used
  *   while in the field.  It's trick is not to do any thing if the
  *   mouse clicked past the field.  We use the cursor ROW() and COL()
  *   functions to do that.  That's because the cursor is not moved
  *   by the mouse click until it's in the destination field.
  *-------------------------------------------------------------------
  ll_ret = .T.
  ln_crow = ROW()
  ln_ccol = COL()
  lc_varread = VARREAD()
  ll_keyin = .F.
  ln_lastkey = LASTKEY()
  lc_keyout = ""

  DO CASE
    CASE lc_varread = "PGDN"
      IF ln_crow = 15 .AND. ln_ccol >= 20 .AND. ln_ccol <= 25
        ll_keyin = .T.
        lc_keyout = "{PgDn}"
      ENDIF
    CASE lc_varread = "PGUP"
      IF ln_crow = 15 .AND. ln_ccol >= 30 .AND. ln_ccol <= 35
        ll_keyin = .T.
        lc_keyout = "{PgUp}"
      ENDIF
    CASE lc_varread = "CTRL_END"
      IF ln_crow = 15 .AND. ln_ccol >= 40 .AND. ln_ccol <= 49
        ll_keyin = .T.
        lc_keyout = "{Ctrl-End}"
      ENDIF
    CASE lc_varread = "ESC"
      IF ln_crow = 15 .AND. ln_ccol >= 54 .AND. ln_ccol <= 58
        ll_keyin = .T.
        lc_keyout = " {27}"
        IF ln_lastkey = 13              && If action is needed on Enter
          ll_ret = .F.                  && Force the VALID failure.
        ENDIF
      ENDIF
    OTHERWISE
  ENDCASE
  IF ll_keyin .AND. ln_lastkey = 13     && If cursor in and Enter pressed
    KEYBOARD lc_keyout                  && KEYBOARD the action.
  ENDIF
RETURN (ll_ret)
```

Code continues on page 8

Function: WhenStat()

```
FUNCTION WhenStat
  *   WhenStat() checks to see if a mouse click was used to enter the
  *   control field.  If it was, then KEYBOARD the action key.  If
  *   the mouse clicked past the field or the keyboard was used to
  *   enter the field, nothing will happen.
  *-------------------------------------------------------------------

  IF gl_ismouse               && If the mouse is active

    ln_mrow = MRow()          && Capture the mouse row
    ln_mcol = MCol()          && and column positions
    lc_varread = VARREAD()
    ll_mclickin = .F.
    lc_keyout = ""

    DO CASE
      CASE lc_varread = "PGDN"
        IF ln_mrow = 15 .AND. ln_mcol >= 20 .AND. ln_mcol <= 25
          ll_mclickin = .T.
          lc_keyout = "{PgDn}"
        ENDIF

      CASE lc_varread = "PGUP"
        IF ln_mrow = 15 .AND. ln_mcol >= 30 .AND. ln_mcol <= 35
          ll_mclickin = .T.
          lc_keyout = "{PgUp}"
        ENDIF

      CASE lc_varread = "CTRL_END"
        IF ln_mrow = 15 .AND. ln_mcol >= 40 .AND. ln_mcol <= 49
          ll_mclickin = .T.
          lc_keyout = "{Ctrl-End}"
        ENDIF

      CASE lc_varread = "ESC"
        IF ln_mrow = 15 .AND. ln_mcol >= 54 .AND. ln_mcol <= 58
          ll_mclickin = .T.
          lc_keyout = "{27}"
        ENDIF

      OTHERWISE

    ENDCASE

    IF ll_mclickin            && If the click was on the field
      KEYBOARD lc_keyout      &&   Send the keystroke to EDIT
    ENDIF

  ENDIF                       && IF gl_ismouse

RETURN (.T.)
```

VENDOR_M.FMT *partial listing*

```
*-- @ SAY GETS Processing. ------------------------------------------------------
*--  Format Page: 1
@ 4,17 TO 16,61 DOUBLE
@ 5,19 SAY "VENDOR"
@ 5,30 GET Vendor PICTURE "9999"
@ 6,19 SAY "VENDORNAM"
@ 6,30 GET Vendornam PICTURE "XXXXXXXXXXXXXXXXXXXX"
@ 7,19 SAY "STREET"
@ 7,30 GET Street PICTURE "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
@ 8,19 SAY "CITY"
@ 8,30 GET City PICTURE "XXXXXXXXXXXXXXXXXX"
@ 9,19 SAY "STATEPROV"
@ 9,30 GET Stateprov PICTURE "XXXXXXXXXXXXXXXXXX"
@ 10,19 SAY "COUNTRY"
@ 10,30 GET Country PICTURE "XXXXXXXXXXXXXX"
@ 11,19 SAY "ZIPPOSTAL"
@ 11,30 GET Zippostal PICTURE "XXXXXXXXXX"
@ 12,19 SAY "PHONE"
@ 12,30 GET Phone PICTURE "XXXXXXXXXXXXXX"
@ 13,19 SAY "FAX"
@ 13,30 GET Fax PICTURE "XXXXXXXXXXXXXX"
@ 14,18 SAY "----------------------------------------"

*-- Control Fields.  These were initially defined on the forms surface
*-- as memory variables.  The code was then modified to remove the m->
*-- prefix.  This will then use the SET FIELDS TO values to make the
*-- fields read-only.
@ 15,20 GET Pgdn PICTURE "XXXXXX" VALID REQUIRED ValidStat() WHEN WhenStat()
@ 15,30 GET Pgup PICTURE "XXXXXX" VALID REQUIRED ValidStat() WHEN WhenStat()
@ 15,40 GET Ctrl_end PICTURE "XXXXXXXXXX" VALID REQUIRED ValidStat() WHEN WhenStat()
@ 15,54 GET Esc PICTURE "XXXXX" VALID REQUIRED ValidStat() WHEN WhenStat()

*-- Format file exit code -------------------------------------------------------

*-- SET STATUS was ON when you went into the Forms Designer.
IF lc_status = "OFF"  && Entered form with status off
   SET STATUS OFF     && Turn STATUS "OFF" on the way out
ENDIF
IF .NOT. ll_cursor
  SET CURSOR OFF
ENDIF

RELEASE lc_fields,lc_status
*-- EOP: VENDOR_M.FMT
```