REMOTE and LOCAL DATABASE ORCHESTRATION

August M. Turano, Ph.D.
MED-CHEK, A DAMON LABORATORY
4900 Perry Highway
Pittsburgh, PA 15229
(412) 931-1281

ABSTRACT

The ability to create a high availability system using a common P.C. programming language, coupled with a separate distributed database on a PC-LAN, is discussed. Use of the M-language, M and DOS operating systems, and interaction with P.C. client programs thru a network is described and explained. The utilization of P.C.'s as transaction agents allows a client/server interaction to develop. P.C. non-MUMPS software provides the user with windowing, event driven programs, mouse control, and graphics without the need for expensive terminals, and does not degrade a mainframe hosts' performance. A description of system architecture, software methodology, networking, cost and time factors are given, when applying this method to a business application.

KEYWORDS

M, distributed database, cooperative processing, client/server, high availability systems, journalling

INTRODUCTION

Databases represent key software technology that drives data acquisition and retrieval in the information age. Databases exist for virtually all types of hardware and software environments, from the smallest P.C. to the largest mainframes. Until recently, databases have been separate and distinct units that might, but usually did not have to, interact with front end application programs. Distributed database software is still not a popular technology, although activity in this area is growing. This article will demonstrate prototype applications where local P.C.

databases are utilized by GUI (Graphical User Interfaces) front end programs that use both local X-base databases and a large distributed M database on a remote machine, to work in concert to perform various application tasks. Development of sophisticated modern interfaces that can be designed by screen oriented drag and drop drawing programs, is particularly appealing to those system analysts and designers that must reengineer older applications or create new applications that compete against first class modern interfaces.

MODERN INTERFACES

The traditional roll and scroll interfaces are a thing of the past. Users demand color, pop-up help windows, and disappearing and reappearing screen control elements. These elements must react appropriately to the most varied and broad spectrum screen input. Large computers that offer graphical capability, such as X-Windows[1] and Motif, have been used, but require special terminals and can be very CPU intensive. Microsoft Windows[2] has become extremely popular and many users are very comfortable in this type of application environment where a consistent user interface is displayed. The cost of P.C. MIPS (Million Instructions Per Second) is roughly a factor of ten (10) or more, less expensive than mainframe MIPS. Therefore, if a P.C. can be utilized to validate, perform screen data capture, and obey user directives, such as mouse clicks, it makes sense to use them. Great cost savings can be realized by employing cooperative processing utilizing both P.C.'s and larger machines connected by networks. Providing better functionality at a reduced cost is a worthy goal for system designers. P.C. hardware is inexpensive, as is most P.C. software; this allows for experimentation and development. In comparison, operating systems, application interfaces, programming languages, statistical packages, graphics packages, etc. cost hundreds of dollars for the P.C., while the same functionality on a mini or mainframe computer could run into the thousands or tens of thousands of dollars.

Event driven computing has now become the defacto standard for most commercial applications. This programming paradigm is not difficult to learn, it just takes a while to adjust to it. Traditional programming usually begins with declarations, then input, computation, and finally output. This activity is usually performed in a top-down manner. In the event driven environment, the user drives the application by clicking on a screen control element, or an action or command button. The user decides the program path, and the application designer merely enforces the rules with which given activities may occur. Some controls may be changed from active to inactive or a new question may appear on a pop-up window in response to input from another control item. A control item might be a command button (FILE/QUIT), a text box, a radio button (buttons that are mutually exclusive eg. sex: male or female), or a list box. Activities that can occur depend on the type of control. Each control object has properties associated with it. The programming environment should limit what operations can be performed on each control. The old style of one line at the bottom of the standard text mode screen for a warning and/or error message is gone. Creating user friendly software for the front end of an application, such as a data entry transaction, can aid in areas where employee turnover is usually high: Reengineering critical applications can be very important to keeping business applications running successfully. Again, the cost of the software and hardware makes tools and experimentation in this area of computing both exciting and worthwhile.

COMBINING LOCAL P.C. X-BASE TECHNOLOGY WITH A REMOTE M DATABASE

Defined more traditionally as client/server computing, the P.C. and a remote network or large mainframe can work together to accomplish a given transaction. The P.C. can be thought of as the front end client responding to the user and making a request to the remote database server. The function of validating input and responding to mouse clicks or commands when the transaction is ready to be filed is a primary function. Next, forwarding the accumulated information to the host machine or network to store the data in a separate repository is highly desirable and more secure. The remote database system can be connected via Ethernet or even RS-232 lines depending on the speed and the amount of communication that needs to be performed between the P.C. client and the host server. In the popular vernacular, the operating environment between the P.C. and the host is utilizing DDE (Dynamic Data Exchange). "DDE is an open language-independent, message based protocol that lets applications exchange data or commands in any mutually agreed on format. The basic concept is that of conversation between client and server, with the client being the initiator"[3]. It is the message, not the medium, that is important in moving data between independent computer systems.

The client software performs a variety of tasks, such as forms presentation, data acquisition and validation. It executes the logic associated with a particular transaction in relation to the application. Data captured by the presentation layer can also be used for data manipulation, queries, etc. Using the local P.C. for validation and program execution minimizes the need for constant data requests on a network; data is requested only on an as needed basis. Sorts and searches can be executed locally, even if the data must be retrieved from the host server. Report generators, statistical and graphics packages can be used to produce output, either directly from the database or indirectly from secondary files that might have been created from a search or sort function. The P.C., which is usually a single user entity, provides very little security for data; message protocols and passwords are necessary to implement proper user authentication. System maintenance is also difficult. If data is stored on various local P.C. databases, it is difficult to insure that proper backups are performed, or that the local database file structure is intact, or that database maintenance is routinely performed. At best it is extremely difficult to manage important data in this manner, so a central data repository makes more sense, and yields a higher confidence level in the application system. This data repository is the remote database server. The database server will handle data storage and backups, provide access and security mechanisms and provide data via database "seeks and gets" using the correct respective indexes to that data. The function of the front end and database server can be summarized as follows[4] (see Figure 1):

GUI Front End	**Database Back End The Database Server**
- forms presentation	- security/access
- data capture	- backups
- data validation	- archival activities
- application logic	- indexes
- report tools	- data retrieval
- menus	- data storage
- data manipulation	

Front End/Data Server Functions

Figure 1

In coupling the P.C. programs to the remote database, speed of display and accuracy of the data is critical. There should be no noticeable delay in an application if lookups are being done either locally or remotely. Fast disks for data inquires are essential in either case. Access time using the network data will predominately take longer than a local database query. For this reason local databases should be used whenever and wherever possible.

It is advantageous when trying new technology to use resources which keep the learning curve low. The X-Base database standard for local database construction is a common very familiar format, which is taught in almost every introductory database course. X-Base is extremely popular and powerful when one considers the abundance of software utilities and libraries from third parties that are available for just about any kind of data manipulation that might be encountered. X-Base provides several basic data types such as,

CHARACTER	- alphanumeric text
MEMO	- longer text files
DATE	- a valid date field
NUMERIC	- numbers where width and decimal places are specified
LOGICAL	- yes or no

In dBase data is stored in files that are comprised of individual data fields. A programmer selects a file with the appropriate index, then can seek or position the file to access any of the data. The dBase program code accesses individual fields much like a global reference with data in various piece positions, then manipulation of each element can occur.

In X-Base form, quick database lookups can be performed using indexed files, where B-tree type file construction provides fast access to specific data. Fields can be combined to form combination indexes (client number + patient ID) where the need for more involved lookups is a factor. Clipper[5], as well as dBase[6] provides an easy-to-learn programming language to access any database file that is created in that environment. The advantage of using a product like Clipper is that it provides a complete system that is portable; the end user does not have to own Clipper. Clipper produces executable files that are C-extensible (can call C routines directly) and can be distributed royalty free. Thus, a large site with many P.C.'s can use the application software for just the development cost, no additional royalties or licensing is needed. Clipper and/or dBase can provide a good database environment, but for reasons discussed earlier, database administration and system functions are difficult to manage. A combination of both local and remote databases to be used at will is optimal. This will minimize data traffic on the network and provide the advantages of the P.C. programming environment combined with the security and database administration associated with a larger remote database. An application designer knows what files are static such as religion, race, states, etc. and these files can be kept locally on the P.C. Large pools of information such as patient names, ID's, daily transactions, prescriptions, bank account transactions, etc. would be kept remotely. It was with this design that the first prototype applications involving patient queries and a patient registration filing application was created.

Clipper was used to design several local databases, but yet the P.C has access to all patient data that was stored on the remote M database. The remote database in our clinical environment consists of six (6) gigabytes of data distributed on four (4) 486/33 machines running MSM-MUMPS 3.0.12[7]. The patient registration application necessitated the creation of several Clipper utility programs that would allow recreation of SET $PIECE and normal $PIECE extraction functions. A set of library routines was written to allow Clipper application code to make direct M calls in a noncryptic and straightforward way. This library works by preparing data and acquiring results via Clipper in a function called MCMD, whose syntax is simply MCMD ("mumps command here"). DEC (Digital Electronics Corporation) provides a DOS product DSM DDP_DOS[8] to provide access to using DSM-DDP protocol. This TSR product provides access using Ethernet directly from the DOS environment. A resident TSR assembles and disassembles packets for the Ethernet network. Each M command has a specific format that must be constructed before the interrupt call that activates the TSR can be made. This syntax, while effective, was tedious to code in application software. A clean way to allow a M programmer access to this functionality was developed, which provided an M syntax that was familiar. Support for SET, KILL, LOCK, $ORDER, $GET, $QUERY, volume set selection, and network open and close utilities provided by the TSR were adapted to a more conventional syntax. In this way, both Clipper syntax and M syntax could be hybrid to design the application.

An example of the Clipper code for performing an M function, such as $DATA follows. Calls to the

Grumpfish library[9] add sparkle to this application using special screen techniques and color. To illustrate the Clipper to M interface, consider the following Clipper code used to obtain $DATA(^PA(value)) from the M database. In Clipper, we would build the full M reference as a string and pass it on to M as follows:

```
GLOBREF:="^PA("+CHR(34)+value+CHR(34)+")"
CMDSTR:="$D("+GLOBREF+")"
RES:=MCMD(CMDSTR)
```

The result, RES, would contain the value returned by $DATA. Note that in Clipper, the "+" is the concatenation operator when operating on strings. See Appendix A for additional examples. The actual performance of the system was quite good. It delivered over 125 database reads per second (approximately .008 seconds per read). These numbers are, of course, dependent on the user load, network traffic, speed of the disks and the CPU processing the requests.

VISUAL BASIC, CLIPPER and M

Clipper, although extremely powerful and easy to use, still lacks in mouse support and ease in overall GUI creation. For these reasons the research continued for a better way to create friendly, attractive user interfaces. Visual Basic has been getting rave reviews for both its DOS and Windows versions. Visual Basic allows the creation of a form, drawn by pointing and dropping items onto the window. All items on the form are given a control name and depending on their type, specific properties are associated with them. For example, a text box can have width, border style, color descriptions for foreground and background, a tab stop number, activity for when the control is activated or deactivated and more. Controls can be grouped together or created separately. A complete development environment with menu construction, mouse support, and ease of coupling the form with specific program code are provided. The language is easy to learn and use, it is Basic with much greater power and control than ever before. Libraries for doing a variety of functions are provided as part of the Professional edition[10]. Visual Basic puts the fun back into programming. Programs can be created for either the Windows or DOS environment, so that a nice looking, colorful interface can be designed for whatever platform is deemed appropriate. The event driven programming paradigm is a little hard to get used to at first, but just takes a day or two to achieve a satisfactory comfort level. Drawing the interface that the user wants is what GUI design is all about and Visual Basic provides that with great ease. In Visual Basic, timers can be set, graphics can be used, complete I/O capabilities are available and EXE files are the final output. The biggest drawback to Visual Basic is the ISAM (Indexed Sequential Access Method) file structure. It is awkward, not an industry standard, and is limited to 128 Megabytes per file. For this reason a hybrid of Visual Basic coupled with a Clipper Database file structure was chosen as the development path. A third party vendor, Sequiter Software[11] provided a set of library routines that allowed access to all dBase functions, such as seek, goto record, and get field data. Thus, it was possible to combine the ease of programming afforded by Visual Basic with the standard dBase format and plethora of access tools and report generators available to the dBase programmer. Visual Basic could now be used just as Clipper was, in the initial experiments to develop P.C. client applications. These applications use the same remote access TSR software that was used in the first Clipper applications. The only changes necessary were internal to the library routines to handle details concerning mixed language memory allocation and string passing inconsistencies.

CONCLUSION

Redesigning the user interfaces of currently operating application code is a common and very important task. Giving applications a face lift benefits both the users and the system designers. If this task is performed correctly, using cooperative processing techniques can save precious mainframe or host database CPU cycles and improve overall system performance. Using the P.C. to perform basic I/O and validation can significantly off load a host processor. The ease of learning and the power of P.C. languages, such as Clipper (X-Base) and Visual Basic, make the reengineering of many applications an exciting activity. The use of royalty free third party libraries allows both consistency and solid software construction affording both modularity and reusability of code. Redesigning the user interface is always worthwhile particularly for older applications, after all the user thinks the interface is the program.

ACKNOWLEDGEMENT

The author gratefully acknowledges the programming skills of YeYi Wang, who created vital parts of the interface library.

ENDNOTES

[1] X-Windows - ANSI Standard Protocol, Committee. X3H3.6, also *MUMPS MDC document* X11/SC11/TG2/91-2.

[2] *Microsoft Windows 3.1©.* Microsoft Corporation, 1990-1992.

[3] Kevin Kornfeld and Kevin Gilhooly. "OOPS via DDE". *Byte.* June 1992, pp. 145-154.

[4] Tom Duesher. "Selling the Database Server". *Reseller Management.* July 1991, pp. 54-61.

[5] *Clipper - CA & Associates©.* Nantucket Corporation, 1984-1990.

[6] *dBase IV - Aston-Tate, A Borland Company©.* Borland International, Inc. 1988-1992.

[7] *Micronetics Mumps©.* Micronetics Design Corporation, 1992

[8] DSM-DP-DOS, Software product, post #'s. BI-PB8XA-BK and BI-PBDAA-BK.

[9] Greg Lief. *Grumpfish Library©.* Grumpfish, Inc., 1988-1991.

[10] *Microsoft Visual Basic Professional Edition, Programming System for MS-DOS©.* Microsoft Corporation, 1992.

[11] *CodeBasic, Database Management©.* Sequiter Software, Inc., 1988-1992.

APPENDIX A Clipper access of MUMPS database

```
/*  acn.prg      program that uses DEC-DDPDOS and CA-CLIPPER to access a remote MUMPS database for lookups as
well as data deposits  */

/*   Copyright 1993 A. Turano Ph.D.    */

// Necessary initialization
// The following are modules that are contained in CDDP.LIB

     EXTERNAL MCMD
     EXTERNAL OPENDDP
     EXTERNAL CLOSEDDP

     UCI="LAB"          // select user access  account
     VOL="LPB"           // select volume set to access
     RES := " "        // Set RES (result) to character,
     ACN := "      "   // (accession number of specimen) to character

     LOOP := 500         // how many times do you want to go thru the loop
                          // accessing remote data

// Set screen up

     SAVE_DRAPE("temp.scr")  // fancy save for DOS screen - restore on exit
     SET COLOR TO  "W+/B,W/R"   //set some screen color parameters
     SET DECIMALS TO 4          //set number of decimals for computations
     SET SCOREBOARD OFF          //don't show the bottom screen activity monitor
     CLEAR SCREEN

     // call grumpfish exploding box function

     msg="Med Chek Labs - A Damon Laboratory"
     CLRSCR(5)     'fancy screen wipe
     @23,30 SAY "Hit a key to continue...."

     FALLGUY(20,23,msg,100)  // grumpfish library calls
     RAINBOW(msg)            // grumpfish library calls

// Open Channel to MUMPS system
     CLRSCR(8)

     ExpBox(1,1,22,76,1,20,"W+/BG+","Accessing MUMPS through Clipper")
     SET COLOR TO "W+/BG+"
     OK := OPENDDP(UCI,VOL)   //open the DDP channel
     IF OK = -1
        IMPBOX(20)        //make an imploding box
        CLS
        @ 5,5 SAY "Bad UCI or VOLUME name!"
        RETURN
    ELSEIF OK = -2
        IMPBOX(20)
        CLS
        @ 5,5 SAY "DDP unable to start.."
        RETURN
    ENDIF

// Get ACN from user
    @ 2,5 SAY "Enter starting accession number:" GET ACN PICTURE "!9999999"
    READ
    setcolor("B/BG+")
    @ 2,5 say "MUMPS $O through Patient File, "+str(LOOP)+" Records"
    setcolor("W+/BG+")

// Build MUMPS command
    GLOBREF := "^PA("+CHR(34)+ACN+CHR(34)+")"    //construct the global reference
    CMDSTR := "$D("+GLOBREF+")"    //make the command string to be performed
// Look at the MUMPS system and see it that ACN exists
    RES = MCMD(CMDSTR)  //execute a MUMPS command-remote $D
    qout("res=",RES)    //this is the result
    res:= VAL(res)  // everything from MUMPs is character MAKE numeric
    IF res = 0
        IMPBOX(20)    // grumpfish function
        CLS
        @ 5,5 SAY "Accession number not found!"
    ELSEIF res=10
        IMPBOX(20)
        CLS
        @5,5 SAY "Accession number not defined but has descendents."
    ELSEIF res=1
        IMPBOX(20)
        CLS
        @5,5 SAY "Accession number found and has NO descendents."
    ELSEIF res=11
        IMPBOX(20)
        CLS
        @5,5 SAY "Accession number found with descendents."
    ENDIF
    ? "Press any key to continue..."
    INKEY(0)    //wait for user to continue
// Found the accession #, so get the info
    CMDSTR := "$G("+GLOBREF+")"    //construct another command string
    RES = MCMD(CMDSTR)  //go visit MUMPS
    TARRAY := {}  // TARRAY is an array to hold the pieces of data
                  //dynamic array allocation--just like MUMPS!
    DATA := EXPIECE(RES,"^",19,TARRAY)    //Extract the 19th piece of RES
                                          //TARRAY is just in case you want
                                          //more.
```

```
// Display that patient's name+sex+age

    @ 3,5 CLEAR TO 21,75
    rollup(padr(TARRAY[14],35)+padr(TARRAY[16],10)+TARRAY[17])

// Display next LOOP patient names and time it

    STARTTIME := TIME()
    FOR I=1 TO LOOP-1      // DO SOME $O FUNCTIONS

        // Get the next ACN # - Perform the $O function

        CMDSTR="$O("+GLOBREF+")"

        RES = MCMD(CMDSTR)  // go visit MUMPS

        // Get data for that ACN #

GLOBREF="^PA("+CHR(34)+RES+CHR(34)+")"  //global reference
        CMDSTR := "$G("+GLOBREF+")"

        RES = MCMD(CMDSTR)  //go visit MUMPS

        // Pick out the patient's name
        TARRAY := {}
        DATA = EXPIECE(RES,"^",19,TARRAY)
        // Write out PIECES 14=Name+PIECE 16=Sex+PIECE 17=DOB
        rollup(padr(TARRAY[14],35)+padr(TARRAY[16],10)+TARRAY[17])
    NEXT
    ENDTIME := TIME()

// Show the results and times

    @ 2,5 CLEAR TO 21,75
    @ 4,5 say "Starting Time: " + STARTTIME
    @ 5,5 say "Ending Time: " + ENDTIME
    SECS := ComputeTime(STARTTIME, ENDTIME)
    @ 7,5 say "Seconds per read, piece, and display: " + STR(SECS/LOOP)
    @ 8,5 SAY "# Reads, pieces, displays per second: " + STR(LOOP/SECS)
    @ 17,5 SAY "Press a key to CONTINUE..."
    INKEY(0)

// Display that patient's name
    clrscr(2)
    setcolor("B/BG+")
    @ 2,5 SAY "MUMPS $Q through Patient File, "+str(LOOP)+" Records"
    setcolor("W+/BG+")

// Display next LOOP patient names and time it
```

```
    STARTTIME := TIME()
    FOR I=1 TO LOOP        // DO SOME $O FUNCTIONS

        // Get the next ACN #
        CMDSTR="$Q("+GLOBREF+")"
        RES = MCMD(CMDSTR)     //go visit MUMPS

        // Get data for that ACN #
        GLOBREF:=RES
        CMDSTR := "$Q("+GLOBREF+")"

        RES = MCMD(CMDSTR)     // go visit MUMPS

        rollup(globref+"=")
        cmdstr="$G("+globref+")"
        RES = MCMD(cmdstr)
        rollup(res)

    NEXT

    ENDTIME := TIME()

    // Show the results

    @ 2,5 CLEAR TO 21,75
    @ 4,5 say "Starting Time: " + STARTTIME
    @ 5,5 say "Ending Time: " + ENDTIME
    SECS := ComputeTime(STARTTIME, ENDTIME)
    @ 7,5 say "Seconds per read, piece, and display: " + STR(SECS/LOOP)
    @ 8,5 SAY "# Reads, pieces, displays per second: " + STR(LOOP/SECS)
    @ 17,5 SAY "Press a key to exit..."
    INKEY(0)



// Measure only READ time
    @ 2,5 clear to 21,75
    setcolor("B/BG+")
    @ 7,5 SAY "Measure of RAW SPEED doing "+str(LOOP)+" $Qs:"
    setcolor("W+/BG+")
    SET CURSOR OFF
    STARTTIME := TIME()
    FOR I=1 TO LOOP        // DO SOME $O FUNCTIONS
        CMDSTR="$Q("+GLOBREF+")"
        RES = MCMD(CMDSTR)
        GLOBREF:=RES
        CMDSTR := "$G("+GLOBREF+")"
        RES = MCMD(CMDSTR)
    NEXT
    ENDTIME := TIME()
    SET CURSOR ON
    SECS := ComputeTime(STARTTIME, ENDTIME)
    @ 9,5 SAY "Seconds per READ: " + STR(SECS/LOOP)
    @ 10,5 SAY "READS per second: " + STR(LOOP/SECS)

// Close up
    @ 15,5 SAY "Press a key to exit..."
    INKEY(0)
    ImpBox(20)     // Close up the on-screen box
    PULL_DRAPE("TEMP.SCR",30)     //GRUMPFISH FUNCTION
RETURN

// ****************************************************************

//Mimic the $Piece function with some extras

     FUNCTION EXPIECE(mstring,delim,n,narray)
     local xstr,i   && xstr=is working string, i=loop counter
                    && n is the piece you want, narray =logical return
                    && entire array of pieces
     IF N>256
        return "* Error piece number too large*"
     ENDIF
     xstr=mstring+delim
     FOR i=1 TO N
        pos=AT(delim,xstr)  && postion of delimiter
         IF pos=0
          EXIT  && exit the loop
         ENDIF
        pie=SUBSTR(xstr,1,pos-1)
        IF narray!=NIL
         AADD(narray,pie)  && Build an array of pieces
        ENDIF
        IF i=n
          return pie
        ENDIF
        xstr=SUBSTR(xstr,pos+1,len(xstr)-pos)
     NEXT
     RETURN NIL  && not found

/******************************************************************/
//Mimic the set $piece function

     FUNCTION SETPIECE(mstring,delim,N,mvalue)
     && RELEASE parray  && clean up the array before starting...
     IF N>256
        return "* Error piece number too large*"
     ENDIF
```

```
   xstr=mstring+delim
   i=0 && counts the number of delimiters
   DO WHILE AT(delim,xstr)>0
      i=i+1
      pos=AT(delim,xstr)  && postion of delimiter
      pie=SUBSTR(xstr,1,pos-1)
      AADD(parray,pie)  && Build an array of pieces
      xstr=SUBSTR(xstr,pos+1,len(xstr)-pos)  && shorten string by 1 delim
   ENDDO
   IF i<n      && This code will add null elements to the needed depth

      FOR J=1 TO N-i
       AADD(parray,"")
      NEXT

   ENDIF       && delimiter
       parray[N]=mvalue  &&set the appropriate element
       mystring=""
      FOR I=1 TO LEN(PARRAY)  && construct the changed delimited string
       mystring=mystring+parray[i]+delim
      NEXT
      mystring=SUBSTR(mystring,1,LEN(mystring)-1)
   RETURN mystring


function ComputeTime(STARTTIME, ENDTIME)
local MINS, SECS
    MINS := VAL(SUBSTR(ENDTIME, 4, 2)) - VAL(SUBSTR(STARTTIME, 4,2))
    SECS := VAL(SUBSTR(ENDTIME, 7, 2)) - VAL(SUBSTR(STARTTIME, 7,2))
    IF SECS < 0
        SECS := 60 + SECS
    ENDIF
return(SECS+60*MINS)

/* *********************FUNCTION  rollup  ****************** */
STATIC FUNCTION rollup(mtxt)
        SCROLL(2,5,20,70,1) // TOP LEFT, BOTTOM RIGHT , # OF LINES TO SCROLL
        @20,5 SAY mtxt
        RETURN NIL
```