PDF pour AC_18215 scan Memory

Le présent fichier PDF **Memory Management** concerne le document

ExperLisp,
For the MacIntosh

Il s'agit d'extraits d'un Manuel de 220 pages pour un logiciel créé en 1985-1986 par la société Californienne **ExperTelligence, Inc**.
Le directeur de cette société est Denison Bollay (voir 3ème page de couverture) et l'Auteur du manuel est Dean A. Ritz
A l'intérieur de ce Manuel, deux disquettes de 3'' pour la version 1.5 du logiciel ExperLisp, un langage de programmation LISP.

Cet ouvrage est disponible sous le numéro **AC 18215** dans l'inventaire de la collection de l'ACONIT, Association pour un conservatoire de l'informatique et de la télématique, Grenoble.

Auparavant il appartenait aux collections de la Bibliothèque Municipale de Grenoble, d'où la référence MA 2001 007 0003 qui apparaît sur la page de couverture.

Le présent PDF fournit un scan d'extraits de cet ouvrage, avec copie de :

-la page de couverture du Manuel,
-les 2 pages 189-190 décrivant le contenu des deux disquettes et leur mise en œuvre,
-les pages 201 à 206 du « Memory Management »,
-les pages 207 à 210 pour l'accès aux « serial ports ».

Fiche rédigée le 24/6/2020 par Jean Ricodeau.

ExperLisp™ for the Macintosh™

by ExperTelligence™

LIS

Files on Your ExperLisp Disk

Your ExperLisp disk contains three items: ExperLisp, ExperLisp Folder, and Sample Programs.

- **ExperLisp** manages the ExperLisp application. This is the icon that can open the application.

- **ExperLisp Folder** contains the three very important documents: COMPILER, LispENV, and ▪lispinit.

- **Sample Programs** folder contains ExperLisp documents. These are text only files containing ExperLisp source code for some awe inspiring sample programs.

As mentioned, the ExperLisp folder contains:

- **COMPILER** contains the ExperLisp compiler and a bulk of the ExperLisp functions. The information in this document is indexed by LispENV. The compiler creates different LispENV documents if it detects that it is running on a different machine, or if the LispENV has been damaged due to an ungraceful exit. ExperLisp will first look for the COMPILER on the disk volume which contains ExperLisp, and then on the disk volume which contains the system. These are the only disk volumes that are checked.

- **LispENV** contains an index of the COMPILER. This index changes from one size macintosh to another. Therefore, there are times when ExperLisp will rebuild the LispENV. LispENV is always created on the disk volume which contains the ExperLisp application.

☞ Erratic behavior can sometimes be attributed to a damaged LispENV, therefore it is recommended that you dispose of the LispENV and let ExperLisp create a new one if you are experiencing erratic behavior. This is done by dragging the LispENV icon into the trash can, and then selecting **Empty Trash** from the **Special** menu.

- **▪lispinit** is the name of an ExperLisp document whose contents are automatically compiled during bootup. Do not recompile it. Some system initialization code is placed there. You may add to the end of this

document if you wish to have them compiled every time you open the application.

Additionally, the Sample Programs folder contains the ExperLisp document **Touretzky**. This file contains source code that, when compiled, modifies ExperLisp to more closely resemble the Lisp in the book *LISP - A Gentle Introduction to Symbolic Computation.* This book is supplied as the tutorial text for those learning Lisp.

Copy Protection

This ExperLisp package includes two (count'm 2) copy protected disks. You can, though, run ExperLisp from a hard disk drive. Simply drag all the files from the master disk onto a volume of the hard disk drive. ExperLisp will ask to see the master disk each time the application is opened. As of yet, programs which place copy protected programs onto a hard disk cannot remove the necessity for validation by the master disk and may cause a "flakey" copy to be placed onto the hard disk.

Memory Management

Memory management is automatically handled by the system. But should you wish to manipulate blocks of memory on your own you can use the following information.

The Macintosh offers two types of memory blocks: relocatable (handles), and non-relocatable (pointers). ExperLisp promotes the use of relocatable memory blocks.

Using Handles

Handles point at relocatable blocks of memory via one level of indirection. The actual contents of the handle's address is an address which points at the current starting address of the relocatable block. The handle's address never changes, but the contents of the handle changes as the Macintosh memory manager moves the relocatable blocks. See your copy of *Inside Macintosh* for more information.

Using relocatable blocks is recommended. Fragmenting the heap by using non-relocatable blocks is not socially acceptable behavior.

```
(SETQ  MYHANDLE  (NEWHANDLE  100))
 ->   ADDRESS  #680111FC
```

This allocates a relocatable block in memory of 100 bytes and returns a handle. Before accessing a relocatable block of memory it is important to lock it in place using HLOCK. This will insure that your block of memory doesn't move underneath you. After accessing the block of memory be sure to HUNLOCK it so that the memory manager can do its job and relocate it as needed.

☞ It is not always necessary to HLOCK your memory block. You only need to lock it if you are not doing other things with memory between the time you get a pointer to the block, and the time you actually access the block using that pointer. It is better to lock the block if you are unsure about this.

This locks the block in memory.

```
(HLOCK  MYHANDLE)
  ->   0
```

This binds the variable POINT to an address which is offset 0 bytes from the beginning of the MYHANDLE and with an offset of 0 bytes. By changing the 0 to other values, other locations of the block can be accessed. DEREF of a handle returns a pointer to the beginning of the relocatable block. MAKELOC returns a new pointer that is offset *N* (in this case 0) bytes into the block. *N* can be either positive or negative number.

```
(SETQ  POINT  (MAKELOC (DEREF MYHANDLE)  0))
  ->   ADDRESS  #6801108C
```

STOW stuffs the address of the Lisp object 6 into a long starting at the POINT.

```
(STOW 6 POINT)
  ->   6
```

FETCH is retrieving the Lisp object whose address is stored in a long at POINT.

```
(FETCH  POINT)
  ->   6
```

STOW and FETCH manipulates longs, STOWWORD and FETCHWORD manipulate words, and STOWBYTE and FETCHBYTE manipulate bytes.

DISPOSHANDLE releases the memory allocated by the block NEWHANDLE and returns it to the free memory pool. The variable we used is now set to a useless value. This is done to prevent the accidental use of a disposed handle.

```
(HUNLOCK  MYHANDLE)
  ->   0
(DISPOSHANDLE  MYHANDLE)
  ->   0
(SETQ  MYHANDLE  NIL)
  ->   NIL
```

☞ System performance will deteriorate (if it performs at all) if you leave locked handles around fragmenting the application heap - or- if you try using a disposed handle. THIS IS A VERY IMPORTANT POINT!!

Normally, neither handles nor their contents are garbage collected. It is possible, though, to create memory blocks whose contents (other Lisp objects) are garbage collected. It is not possible to create handles which are garbage collected.

Handles are given type bits of $68. Lisp objects pointed to by handles are not garbage collected. Lisp objects pointed to by environments are garbage collected. This is a trick which gets the garbage collector to respect the Lisp objects pointed to by your homemade relocatable memory blocks.

☞ Notice that HLOCK and HUNLOCK are missing from the steps that follow. They are not necessary for simple things like this. Again, use HLOCK and HUNLOCK if you are unsure about this.

(1) Grab your block of memory.

```
(SETQ STORAGE (NEWHANDLE 48))
  ->   ADDRESS #68011ED8
```

(2) Coerce the handle to type environment using the global system variable DTP_ENV.

```
(SETQ STORAGE (COERCETYPE DTP_ENV STORAGE))
   ->   FUNCTION #5C011ED8
```

(3) Stow NIL into the first 8 bytes. Because the address of a Lisp object takes 4 bytes, it is necessary to STOW NIL only twice. Once at the beginning of the block, and again with an offset of 4 bytes from the beginning. This is necessary for our ExperLisp garbage collector.

```
(STOW  NIL  (MAKELOC (DEREF STORAGE) 0))
   ->   NIL
(STOW  NIL  (MAKELOC (DEREF STORAGE) 4))
   ->   NIL
```

(4) Now store a unique Lisp object that has not yet been entered into the system. This will enable us to test our protection from the dreaded garbage collector. Use something unique, like your name on a list.

```
(STOW  '(VINNY SHINBLIND) (MAKELOC (DEREF STORAGE) 8))
   ->   (VINNY SHINBLIND)
```

(5) Perform a garbage collect and then check the Lisp object and see if it was swept away by the garbage collector.

```
(GC)
   ->   NIL
(FETCH (MAKELOC (DEREF STORAGE) 8))
   ->   (VINNY SHINBLIND)
```

☞ The functions which operate on handles operate only on those of type $68. Therefore it is necessary to coerce your handle to type $68 before before using any of the following functions: DISPOSHANDLE , NEWHANDLE, SETHANDLESIZE, GETHANDLESIZE, HLOCK, HUNLOCK, HPURGE, and HNOPURGE. These functions operate exclusively on handles.

STUFF-LIST-IN-BLOK stores the contents of a list in a relocatable memory block. It is not necessarily more memory efficient, it is just a further example of safely using relocatable memory blocks.

```
(DEFUN STUFF-LIST-IN-BLOK (L &AUX BLOK POINT L-LENGTH)
  (SETQ L-LENGTH (LENGTH L)
        BLOK (NEWHANDLE (+ 8 (* L-LENGTH 4)))
        BLOK (COERCETYPE DTP_ENV BLOK)
        POINT (DEREF BLOK))
  (STOW NIL (MAKELOC POINT 0))
  (STOW NIL (MAKELOC POINT 4))
  (DOTIMES (INDEX L-LENGTH)
           (STOW (CAR L)
           (MAKELOC POINT (+ 8 (* INDEX 4))))
           (SETQ L (CDR L)))
  BLOK)
```

GET-LIST-FROM-BLOK returns a new list comprised of the Lisp objects stored in the supplied memory block.

```
(DEFUN GET-LIST-FROM-BLOK (BLOK &AUX POINT SIZE RESULT)
  (SETQ SIZE (GETHANDLESIZE (COERCETYPE $68 BLOK))
        POINT (MAKELOC (DEREF BLOK) 0))
;EACH INDEX IS OFFSET BY + 4 BYTES. SKIP THE FIRST 2 SLOTS
  (DOTIMES (SLOT (- (LSH SIZE -2) 2))
           (PUSH
             (FETCH (MAKELOC POINT (+ 8 (* SLOT 4))))
            RESULT))
  (NREVERSE RESULT))
```

DISPOSIT is useful for disposing of handles and their associated memory.

☞ This can be used to dispose of bunnies and arrays. It is recommended you do so because neither bunnies nor arrays are garbage collected.

```
(DEFMACRO DISPOSIT (HAND)
   `(PROGN
      (DISPOSHANDLE (COERCETYPE $68 ,HAND))
      (SETQ ,HAND NIL)))

(SETQ MYLIST (STUFF-LIST-IN-BLOK
  '(HAVE A (TUNA COLADA))))
   -> function #5C011F3C
(GET-LIST-FROM-BLOK MYLIST)
   -> (HAVE A (TUNA COLADA))
(GC)
   -> NIL
(GET-LIST-FROM-BLOK MYLIST)
   -> (HAVE A (TUNA COLADA))
(DISPOSIT MYLIST)
   -> nil
```

Garbage Collection

CONS space, STRING space, and NUMBER space are garbage collected. STRING space and NUMBER space grow on demand (string space is limited to 32K), though they cannot shrink. CONS space is fixed at boot time, taking into consideration the available memory {up to 4 megabytes}. String space, unlike the other data spaces, is not automatically garbage collected.

The function GC invokes a garbage collection. It does not invoke a garbage collection of string space. Use the function STRGC to invoke a garbage collection of string space. String space is only garbage collected when an explicit STRGC is executed.

☞ Please note that arrays objects and bunnies are not garbage collected. Use the DISPOSIT macro described earlier to clean up after yourself.

☞ Also be sure to dispose of your region definitions with DISPOSERGN, and your polygons with KILLPOLY. It is the only humane thing you can do.

Tell the Driver

ExperLisp offers access to the Macintosh serial ports. This is done through the file system using the functions OPEN_READ and OPEN_WRITE.

`(OPEN_READ ".Ain")` returns an output stream to the modem port.
`(OPEN_READ ".Bin")` returns an output stream to the printer port.
`(OPEN_WRITE ".Ain")` returns an input stream to the modem port.
`(OPEN_WRITE ".Bin")` returns an input stream to the printer port.

Data can be read from a serial port using either FTYI, or READ-CHAR. Data can be send out a serial port using, FTYO, PRINT, PRINC, or PRIN1. FTYI and FTYO are the fastest functions for accessing the serial port.

Disclaimer: Some of the example programs require knowledge of ExperLisp and Macintosh memory management. Incorrect uses of memory management techniques can result in serious system errors. Good luck!

The example programs below show how to read in a form from a serial port. The data is read in character by character and stored into a memory block (see the section of Memory Management for more information). When the reading is done (either the ENDCHR was received or the memory block became full) the handle for the block, along with a beginning and ending offset, is passed to the reader. Here it is read and the result is returned.

```
(DEFUN READ-ONE-LINE (PORT &AUX STREAM BUFF RESULT)
   (SETQ PORT (IF (EQ 'MODEM PORT) ".Ain" ".Bin")
         BUFF (NEWHANDLE 1000)
         STREAM (OPEN_READ PORT)
         RESULT (READ-FROM-BUFF BUFF 0
                    (PUT-IN-BUFF BUFF STREAM 13)))
   (CLOSE STREAM)
   RESULT)
```

```
(DEFUN READ-FROM-BUFF (BUFF BEGIN END &AUX STORE)
   (SETQ STORE (CONS BUFF (CONS BEGIN END)))
   (SETQ *STDSTREAM STORE)
   (WHILE (< (CADR *STDSTREAM) (CDDR *STDSTREAM))
          (SETQ ALIST (READTOP NIL))
          (SETQ *STDSTREAM STORE))
   ALIST)

(DEFUN PUT-IN-BUFF (ABUFFER ASTREAM ENDCHAR
                    &AUX (OFSET 0) BUFPT ACHR
                    (BUFFSIZE (GETHANDLESIZE ABUFFER)))
   (HLOCK ABUFFER)
   (SETQ BUFPT (MAKELOC (DEREF ABUFFER) 0))
   (WHILE (AND (< OFSET BUFFSIZE)
               (≠ (SETQ ACHR (FTYI ASTREAM)) ENDCHAR))
       (STOWBYTE ACHR (MAKELOC BUFPT OFSET))
       (SETQ OFSET (+ OFSET 1)))
   (HUNLOCK ABUFFER )
   OFSET)
```

Another useful serial port function is SERIALP. SERIALP returns T if characters are waiting for input on the printer port of the Macintosh. Convert the hex number from $2D8 to $2D0 to check the modem port instead. It assumes that a stream for the port has already been created.

```
(DEFUN SERIALP (&AUX NUMINBUF NUMREAD)
   (SETQ NUMINBUF (FETCHWORD (MAKELOC (DEREF $2D8) $2C)))
         NUMREAD (FETCHWORD (MAKELOC (DEREF $2D8) $2E)))
   (NOT (= NUMINBUF NUMREAD)))
```

Using the above example functions as models, we can construct a series of functions that load and compile a text file.

EVAL-FROM-BUFF takes a handle, a begin offset, and an end offset. It passes the contents of the buffer to the reader and compiler. The 'ª' is printed by depressing the "Option" and "9" keys simultaneously.

```
(DEFUN EVAL-FROM-BUFF (BUFF BEGIN END &AUX STORE)
   (SETQ STORE (CONS BUFF (CONS BEGIN END)))
   (SETQ ªSTDSTREAM STORE)
   (WHILE (< (CADR ªSTDSTREAM) (CDDR ªSTDSTREAM))
          (EVAL (READTOP NIL))
          (SETQ ªSTDSTREAM STORE)))
```

LOAD takes a filename (string) and loads the file. The file length is limited to 32765 characters. The compilation of the memory block is wrapped with a CATCH just in case an error is thrown - there is no good reason to leave a 32765 byte block floating around. It is faster to LOAD several smaller files than to create and LOAD a single huge file.

```
(DEFUN LOAD (FILENAME &AUX STREAM RESULT
                           BLOK ALIST (FLAG T))
   (IF (SETQ STREAM (OPEN_READ (GETPREP FILENAME)))
       (PROGN
          (CATCH NIL
             (PRINT "loading...")
             (SETQ BLOK (NEWHANDLE 32765))
             (EVAL-FROM-BUFF BLOK
                             0
                             (LOAD-IN-BUFF BLOK STREAM))
             (SETQ FLAG NIL))  ;closes the CATCH
          (IF FLAG
              (PROGN
                (DISPOSHANDLE BLOK)
                (CLOSE STREAM)
                (THROW NIL NIL))
              (PROGN
                (DISPOSHANDLE BLOK)
                (CLOSE STREAM))))))
```

LOAD-IN-BUFF stuffs the contents of a file into the memory block ABUFFER until the end of the file is reached. It returns the number of characters stored in the buffer. It is closely modelled after PUT-IN-BUFF

```
(DEFUN LOAD-IN-BUFF (ABUFFER ASTREAM
                     &AUX (OFFSET 0) BUFPT ACHR)
   (HLOCK ABUFFER)
   (SETQ BUFPT (MAKELOC (DEREF ABUFFER) 0))
   (WHILE (NOT (END_OF_FILE ASTREAM))
          (SETQ ACHR (FTYI ASTREAM))
          (STOWBYTE ACHR (MAKELOC BUFPT OFFSET))
          (SETQ OFFSET (+ OFFSET 1)))
   (HUNLOCK ABUFFER)
   OFFSET)
```