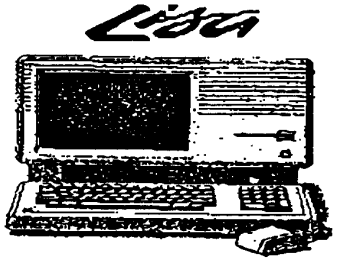
Date: November 7, 1980

To: Lisa Software

cc: Couch, Smith

From: Tom Malloy



(1)

Subject: Interface Specification for the standard storage manager

Several weeks ago we discussed the advantages of a unified intra-segment storage management strategy. At that time I proposed we adopt the Lotus storage manager, or some variation thereof, as the standard. This met with some approval. Attached is an intefface specification for the storage manager we discussed. Please read and comment copiously on content and style.

It seems unavoidable to have two memory managers; this intra-segment manager and the OS' real-memory/segment manager. While two is not as good as one it is certainly much better than one per application. Between these two pieces of software it would be nice (and I think it is very possible) to satisfy most/all of our memory management needs. If we have almost satisfied your needs, but not quite, lets get together and talk - maybe we can enhance one or both of the memory managers to avoid the need for another one.

Thanks - Tom

DTC 4/2001

The Lisa's intra-segment memory manager, which appeared 1st in the LisaWrite program (principal programmer was Tom Malloy of the Xerox Bravo word processor fame) was later adopted as the Apple Macintosh computer's system-wide memory manager (ca. 1984) by Bud Tribble and Andy Hertzfeld.

Preliminary Interface Specification - UnitHz ②

I. Introduction

UnitHz performs intra-segment memory management for Lotus, LisaCalc, the Window Manager (Lily) and portions of the Operating System and Pascal run-time. UnitHz provides a layered implementation of "heap-like" storage regions, called zones. The most basic layer provides allocation and deallocation of non-relocatable blocks of storage within the zone. This is, in the parlance of Pascal, "a Pascal heap with NEW and DISPOSE". At the second level, code is added to provide relocatable storage block management: a compacting heap. Since this relocation ability is outside the domain of knowledge of both the hardware and the program development environment, certain conventions must be obeyed by the programmer when using this storage mechanism. Failure to follow the conventions will usually produce catastrophic side effects. This potential drop in reliability should be weighed against the added flexibility when choosing a storage management methodology. Finally, a third level of code implements a primitive, object oriented virtual memory. Lotus is the only known client of this interface. Storage blocks managed by UnitHz have a maximum size of 32k bytes. The zones themselves may be arbitrarily large.

DTC 4/2001: See figure 1, p. 6

II. Dependencies

UnitHz imports UnitStd to provide a set of very common type declarations not pre-defined by Pascal. For instance the type, TP, is a pointer to undescriminated storage (i.e. compatible with any pointer). Similarly there are standard type declarations for arrays of bytes, arrays of characters and other low level primitives. UnitStd also defines primitive operations on these types, such as the maximum and minimum of two integers (CMax and CMax).

III. Key objects

The key exported objects of UnitHz are the handles on the storage blocks of various types. In particular:

p — undescriminated pointer. The handle on a non-relocatable block of storage is a p.

h — a pointer to a p. This double indirect pointer is the handle on a relocatable block of storage. The h is invariant with respect to compaction of the zone. The pointer, h^, is not. The conventions concerning the use of the compacting allocator deal with the circumstances under which these handles can be dereferenced. The dereferencing rule, in its simplist form states that all dereferenced handles become invalid upon the execution of any procedure in the UnitHz interface. A more practical corollary states that any procedure call of unknown effect may invalidate the handles. This includes all external procedures. In practice, Lotus implementors typically consider dereferenced handles invalid after any procedure call, internal or external.

The handle, h, points at a single, unique pointer to the

(3) block itself. The block, in turn, contains enough information to locate this pointer when the compactor wishes to relocate it. The pointer must be in a static location. In particular, it cannot be embedded in a relocatable block.

n name. A 32-bit (mostly?) uninterpreted handle on a swappable object. Access to swappable objects is via a procedural interf which maps the name through a hash table which identifies all of the currently resident, swappable objects. If the objec is is not resident, a user definable "trap" routine is invoked to read it into the cache. The particulars of this interface will remain vague at least until we design the procedure variable scheme.

IV. Secondary objects

UnitHz manages a collection of storage zones (ahz's). A storage zone is a contiguous block of memory. It is expected that most zones will be the complete contents of hardware data segments, although this is not a requirement. A storage zone is divided into a header, describing the contents of the zone, followed by a sequence of storage blocks (abk's). A zone has a type, tyhz, which determines the kind of blocks the zone might contain. We currently imagine four types of blocks:

1) Free blocks
2) Non-relocatable blocks
3) Relocatable blocks
4) Named blocks

It does not seem to make sense to have all of these in a single zone. For instance, it has not been decided whether allowing the mixture of relocatable and non-relocatable blocks in the same zone would be a feature or headache, although it is technically feasible. Let's say for discussion's sake that there are two types of zones:

tyhzNrel	non-relocatable blocks only
tyhzRel	relocatable and swappable blocks only

Each block has a header followed by the user's storage. The header contains a 2-bit block type field, tybk, a 14-bit count of words, cw, and some tybk depe information. Figure 1 illustrates the layout of a zone.

The type dependent information is:

1) Free blocks are linked together on a doubly linked list used by the allocation routines.

2) Non-relocatable blocks have no type dependent information and hence the smallest per block overhead (16-bits)

3) Relocatable blocks have a 16 bit "locator" of the single pointer to the block. This single pointer, which is pointed AT by the handle, h, can be in one of two general areas. Most of these pointer will be in a pool of pointers at the beginning of the zone, in which case the locator is an offset from hz to the pointer. When the zone is initialized the user may supply another base address, pBase. This is provided to allow the user to place block pointers in his global data area. Any call to allocate a block must supply a handle which points at a location within 32k bytes of pBase or hz.

4) Named blocks contain the 32-bit name which identifies the block plus some miscellaneous information about the object. For instance, there is a small LRU timer used by the software to choose a block to remove from the cache. (4)

V. Initialization, reconfiguaration and destruction of objects

HzInit(pFst, pLim, ipPoolMac, logIpnLim, pBase) : hz

The contiguous block of storage IN [pFst..pLim) is turned into a zone. If ipPoolMac > 0 then a pool of pointers is allocated at the beginning of the zone. These pointers will point at relocatable blocks, therefore the address of each pointer is potentially an h-type handle. A hash table large enough to hold 2^^logIpnLim pointers is allocated as a relocatable block in the zone. A handle on the zone is returned. We can imagine dynamically increasing or decreasing ipPoolMac and ipnLim, but the current implementation does not include this feature.

A zone which contains no user data can be abandoned at any time by the user without notifying UnitHz.

VI. Operation

IMPLEMENTATION NOTE:

In order to allow for multiple zones each routine in the implementation must take the zone handle, hz, as an argument. This may be viewed, with some justification, as a high price to pay. A subtle implementation decision was made in order to minimize this overhead in the future. Namely, the zone pointer is always the first parameter to the procedure. If and when the Pascal run-time conventions are changed to pass procedure parameters in registe this could have the very pleasant effect of the zone handle being placed in an address register when an interface procedure is called and remaining there for the duration of its execution. This convention may be of use to other programmers implementing "object oriented" interfaces.

AllocateBk(hz, hDst, cb, tybk)

Allocates a block of type tybk with cb bytes of user storage in zone hz. A pointer to the user's portion of the block is placed in hDst^.

HAllocate(hz, cb) : hRslt

Allocates a relocatable block with cb bytes of user storage in zone hz. A pointer to the block is allocated from the pool and its address returned as the result.

FreeBk(hz, hDst, tybk)

Frees the block refered to by hDst.

ChangeSizeH(hz, hDst, cbNew)

Changes the size of a relocatable block refered to by hDst. Bad news if it is not a relocatable block.

PMapN(hz, n, fLock) : p

Maps the name, n, through the hash table of resident named objects. If it is found a pointer to the user portion of that block is returned. Otherwise, <hand wave> the user-defined "trap" mechanism is invoked to read it into the zone. fLock is a flag. If true the object is locked into physical memory until unlocked by another call on PMapN with the same n and fLock = FALSE. Without this extra feature it would be technically impossible to "dereference" two named objects simultaneously, since the second call might invalidate the first pointer.

SetFDirty(hz, n, fDirty)

Sets the dirty flag for the object named by n. When the object is swapped out it will be updated on the disk.

http://msdn.microsoft.com/library/techart/hungarotat.htm
http://www.rjh.org.uk/PAW/m1102.htm

DTC 4/2001 A little bit of history ...
Tom Malloy follows a programming convention called "Hungarian Style" that originated at Xerox PARC under Charles Simonyi, a former resident of Hungary. This style prefixes variables with letters specifying the type of variable. This style was used in Xerox programs such as the Bravo WYSIWYG word processor on the Xerox Alto (ca. 1975-79) computer. Apple's Lisa programmers sometimes followed this convention.

A HEAP ZONE

⑥

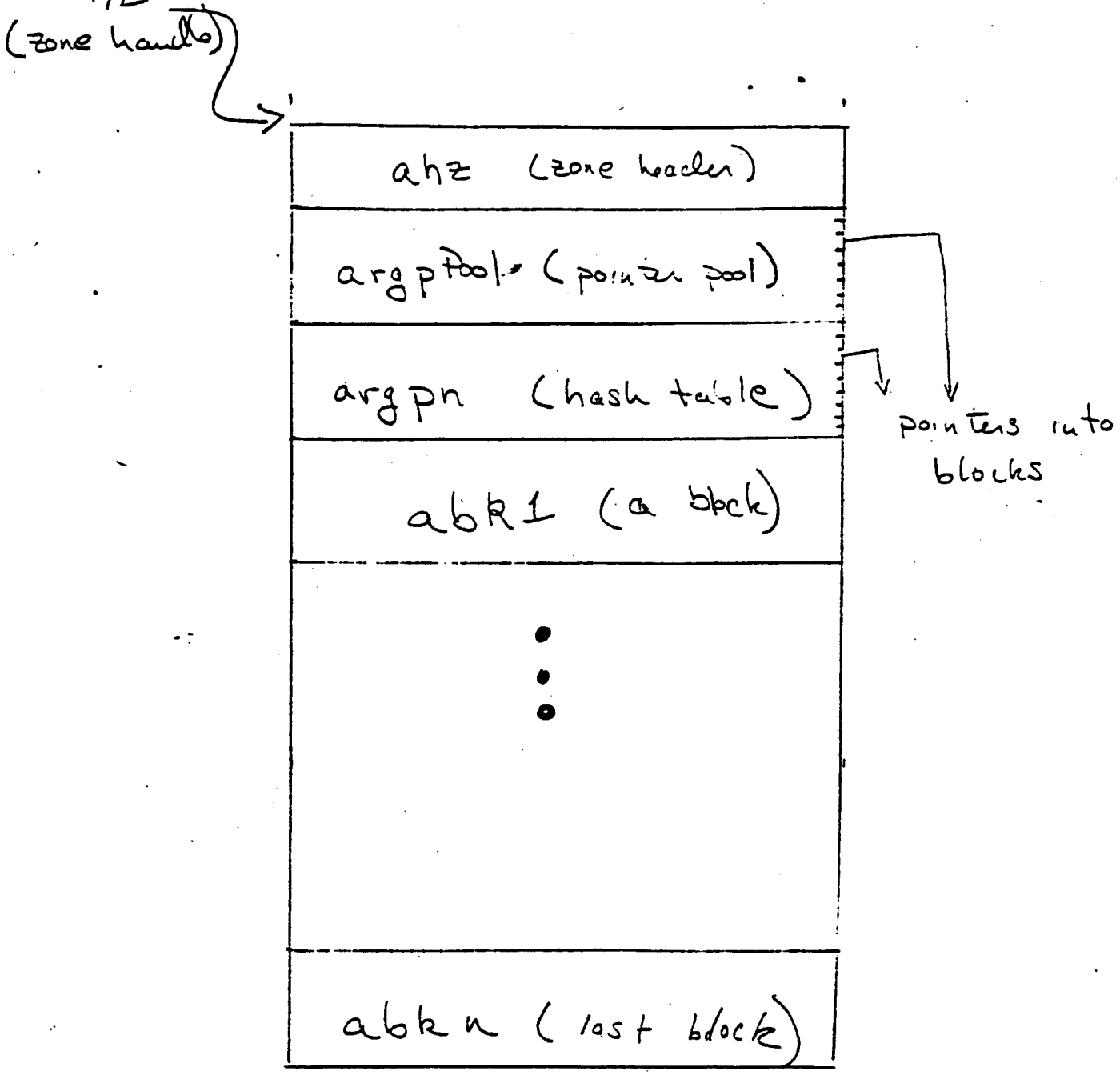


FIGURE 1.

o The End o