Xerox Corporation
3333 Coyote Hill
Palo Alto, California 94304
415 494-4000

November 20, 1981

Paul McCullough
Tektronix
Mail Stop 92-805
Beaverton, Oregon 97077

XEROX

Dear Paul:

Enclosed is the final release of the Smalltalk-80 system. In addition to this letter, you should have:

A magnetic tape containing the Smalltalk-80 system image,

An updated memo, Specifics not covered in the book,

Minutes of the Smalltalk-80 Implementors' Conference.

The virtual machine specification has not been changed for this release, except to clarify two issues. Therefore, your implementation that ran the last release should run this release with at most two minor changes.

The first issue is an omission in the process scheduling primitives. When a Process has been placed on a list (either a Semaphore or one of the ProcessorScheduler's quiescent process lists), it needs to maintain a back pointer to that list. The back pointer goes in the field with index 3 (i.e., the fourth field). The **addLastLink:toList:** routine should set the back pointer of the first argument to be the second argument and the **removeFirstLinkOfList:** routine should set the back pointer of the single argument to **nilPointer**.

The second issue is an unclear specification of the form of the 32-bit clock values used by **primitiveTickWordsInto**, **primitiveTimeWordsInto** and **primitiveSignalAtTick**. These 32-bit values are stored in the first four bytes of a byte-indexable object. The Smalltalk code for the last release assumed that the values were stored with the high order 8-bits in the byte indexed by 1 and the low order 8-bits in the byte indexed by 4. However, since this is the reverse of the convention for LargeIntegers, we have decided to make the specification that the high order 8-bits are in the byte indexed 4 and the low order 8-bits are in the byte indexed 1.

The major change between the last image and this one is the inclusion of a user interface implemented totally in the Model-View-Controller framework. Documentation of how to use this interface will be mailed to you shortly. This final image also includes many more comments than the last and a more coordinated categorization of messages in classes.

XEROX

Since this release still contains much potential for improvement (as well as some outright bugs), communication among us will continue to be desirable. We will still be happy to answer questions over the phone, but we would like to have bug reports and suggestions for improvements in writing. Since we are no longer a focus of distribution, please send copies of these reports to the other implementors. We will only distribute our own suggestions. We will offer the following suggestion as an example.

> The speed of text editing can be improved by a primitive implementation of the method that moves characters from one String to another. Currently this is done in Smalltalk by SequencableCollection's method for replaceFrom:to:with:startingAt:. We suggest that a primitive that accomplishes this for byte-indexable collections be written. For compatability's sake, we suggest that everyone use the primitive index 137. A method including this primitive index should be added to String and ByteArray. The failure code can be either a copy of the superclass code or can relay the message to super (as in SmallInteger +).

We are still looking foreward to a set of papers on the implementation process and will look for first drafts of your contributions around the end of February.

Thank you for your assistance in debugging the release process for Smalltalk-80. All of us at Xerox hope that you feel adequately compensated.

Sincerely,

[illegible]

David Robson
Software Concepts Group

Inter-Mittent Memorandum

To	Smalltalk-80 Implementors	Date	November 18, 1981
From	Glenn Krasner	Location	XEROX PARC
Subject	Smalltalk-80 Specifics not covered in the book	Organization	SCG

Scope

The Smalltalk book contains information necessary to implement Smalltalk, including the list of bytecodes and their implementations, the list of primitives and their implementations, and a description of storage management. The book, however, does not contain specifics for any given Smalltalk-80 virtual memory image, and in particular the image accompanying this memo. This memo is intended to supply those specifics.

Contained below are a description of the file formats for the tape, plus a list of those objects and classes known to the interpreters.

What is on the Tape

The Smalltalk-80 Virtual Memory Image, and associated files, are written on a 9-track, 1600 bpi ~~phase-encoded~~ magnetic tape. The tape consists of binary files in 'continuous stream' mode, with 512 byte records, and an eof mark after each file. The files (in order of appearance) are:

1) Virtual Memory Image (copy 1)	977 records,
2) Virtual Memory Image (copy 2)	977 records,
3) Sources file (in the image, called 'st80jul16.sources')	1992 records,
4) Changes file (in the image, called 'st80jul16.changes')	1 record,
5) Trace file 1 of simulator	55 records,
6) Trace file 2 of simulator	32 records,
7) Trace file 3 of simulator	49 records,
8) List of object pointers for classes	27 records,
9) List of object pointers for methods	316 records,

Order of Bytes

All bytes are considered 8-bits, all words are 16-bits. Words in the file are stored in the order of more significant byte followed by less significant byte. We realize that some implementations would prefer to have words stored low byte followed by high byte. We think that the transformation of the image that would work for most such machines is to swap the bytes of all fields accessed as words, and to not swap the fields accessed as bytes.

For word-type objects: swap every field.
For CompiledMethods: swap Length, Class, Header and Literal fields only.
For all other byte-type objects: swap Length and Class fields only.

THE FILES

Virtual Memory Image File

There are two copies of the Virtual Memory Image file. The Virtual Memory Image consists of length information, followed by the data representing objects (object space), followed by the data representing the object table. The first word (stored as most significant byte first) contains the high-order 16 bits of the length of the object space, and the second contains the low-order 16 bits of this length. The third word of the file is the high-order 16 bits of the length of the object table, followed by the low-order 16 bits. The next 252 words are set to 0. By convention, an image file is defined to be in interchange format if the fifth word (ninth and tenth byte) is zero.

For this image, the first ten bytes are:
0, 3, 130_8, 27_8, 0, 0, 166_8, 222_8, 0, 0.

Following this (starting at the 257th word) is the object space. The first word encountered is the first word of the object whose object pointer (oop) is 2 and whose object space address is 0 (20 bit address). The next words are the fields of the other objects, stored consecutively, up to the length of the object space. Following the object space are enough 0's to start the object table at a page (256-word) boundary.

The next word is the first word of the object table entry for the object with oop (object pointer) = 0. (Which, of course is an invalid oop, but the object table entry exists anyway.) This is followed by the rest of the words of the object table. The last word of the object table is the last word of the file.

Note:
--The length and object space portions of the file are padded with 0's to the end of a page, but the object table is not.
--The object table may contain unused entries. These have the 'freeEntry' bit set, but all other bits in both words are 0. Implementors will have to link these themselves, if desired.
--The object table entries either point to objects in the object space or are free entries, and the object space contains only objects; there is no free memory in the object space and no entry in the object table for free memory blocks.
--The object table assumes that objects are stored contiguously starting at address 0 in a 20-bit address space. There is no distinction for "segment" boundaries. Suitable address translation as necessary is up to the implmentors.

Sources and Changes Files

The third file of the tape is a copy of the system sources. Print it. This file consists of the definition of each class in the system, followed by the code for that class's methods. (There is a form feed character between each class, ASCII 12). The format of this file may be understood by reading the source code for the nextChunk method in class ReadWriteStream and the getSource method in class CompiledMethod; it is the same format as that used in fileIn/fileOut.

Each CompiledMethod in the image contains a pointer to its source code, encoded in the last three bytes of that method. The two msb's of the first of these bytes determine the file on which the source is stored (00=Smalltalk80.sources', 01=Smalltalk80.changes', 10=unused, 11=unused). The six lsb's of this byte with the two following bytes make up a 22-bit pointer specifying where in that file the source code begins. The source code for that method is terminated by $! (imbedded $!s are doubled).

Note:
The sources file on this tape is quite long, about 350 pages when printed; and the changes file has only one expression on it.

Trace Files

The fifth, sixth, and seventh files on the tape contain three traces of the Smalltalk-80 interpreter executing the first bytecodes in this Smalltalk-80 virtual image. These were made by running the formal specification of the interpreter written in Smalltalk-80 itself. The three traces show decreasing levels of detail over increasing durations.

- The first trace shows all memory references, allocations, bytecodes, message transmissions, returns and primitive invocations for the first 112 bytecodes executed.

- The second trace shows only the bytecodes, message transmissions, returns and primitives for the first 405 bytecodes.

- The third trace shows message transmissions, primitives and returns for the first 1981 bytecodes. The lines of this trace are indented according to the level of method invocation (i.e., the depth of the context stack).

The format of each type of entry is given below. All numbers are shown in decimal.

Memory Reference (only in first trace)

Pointer Fetch

object-pointer pointer: *fieldIndex* = *field contents*
(e.g., 20656 pointer: 20 = 1617)

Byte Fetch

object-pointer byte: *byteIndex* = *byte contents*
(e.g., 3872 byte: 46 = 208)

Word Fetch

object-pointer word: *fieldIndex* = *field contents*
(e.g., 18168 word: 0 = 5)

Pointer Store

object-pointer pointer: *fieldIndex* ← *field contents*
(e.g., 20654 pointer: 1 ← 15)

Allocation

allocating oop: *object-pointer*
(e.g., allocating oop: 20654)

Bytecodes (in first and second traces)

Bytecode <*bytecode-index*> *bytecode-description*
(e.g., Bytecode <16> Push Temporary Variable 0)

Message Transmission (in all traces)

[cycle=*bytecode cycle*] *receiver-description selector-string argument-descriptions*
The bytecode cycle is the number of bytecodes that have been executed. The receiver and argument descriptions will show the class of the appropriate object except in the case of SmallIntegers, Strings, true, false and nil which print more nicely.

```
(e.g.,      [cycle=408] aLargePositiveInteger digitAt: 3
            [cycle=75]  40 digitMultiply:neg: 808 false)
```

Primitive Invocations (in all traces)

Primitive *#primitive-index*
(e.g., Primitive #70)

Returns (in all traces)

↑ (method / block) of *returned value description*
(e.g., ↑ (method) of aLargePositiveInteger
↑ (block) of 64)

Object Pointers Lists

As an aid to debugging, the eighth and ninth files are a list of the oops of all classes and methods in the system.

Objects Known to Interpreters

There is a set of objects that must be known by a Smalltalk-80 interpreter. The oops of these objects are usually used as constants to interpreters, but could be located in special tables. These special oops/objects are (those marked * are not necessarily needed by interpreters, but are included in this table as debugging aids):

2 - the object nil
4 - the object false
6 - the object true
010 08h - an Association whose value field is Processor
012 0Ah - *Symbol classVariable USTable, the table of Symbols
014 0Ch - class SmallInteger
016 0Eh - class String
020 10h - class Array
022 12h - *an Association whose value field is the SystemDictionary, Smalltalk
024 14h - class Float
026 16h - class MethodContext
030 18h - class BlockContext
032 1Ah - class Point
034 1Ch - class LargePositiveInteger
036 1Eh - *class DisplayBitmap
040 20h - class Message
042 22h - class CompiledMethod
044 24h - * #unusedOop18
046 26h - class Semaphore
050 28h - class Character
052 2Ah - symbol #doesNotUnderstand:
054 2Ch - symbol #cannotReturn:
056 2Eh - *symbol #monitor:
060 30h - SystemDictionary classVariable SpecialSelectors, the array of selectors for bytecodes 0260-0317
062 32h - Character classVariable CharacterTable, table of Characters
064 34h - symbol #mustBeBoolean

First Ever Smalltalk-80 Implementors' Conference
September 24-25, 1981

Minutes

(These minutes are intended to jar your memories, rather than to be a pretty record of what went on, hence their form.)

Introduced everyone
Went over goals/agenda

Technical Area -- Termination of Review Process

Discussed "September 30" as date on which last image would be frozen
Tape would come a couple of weeks later to allow for documentation
Strong desire to make few/no changes to VMachine from June 30
Strong desire to make this be clean, even at the expense of slipping schedule
Book (vol. 1) will go to publisher on Dec 31; (by definition); review will be done at this time
Should image have changed (desire not to); reviewers will receive new image, which will be the license image (if any)

Image release process bugs/changes

Interchange Format:
Discussed format for images with words other than 16 bits; decided since it is not specified what is needed, we will stick with the one we have.
Discussed whether we would release a small image, with various things clamped out; decided we would not, but might provide guidelines somewhere in the system for paring it down.
Discussed snapshot format as distinct from interchange format; decided this was ok, but required primitives
primitive a) snapshot to snapshot image; required
primitive b) snapshot to interchange format; optional
plus defined starting Smalltalk-80 as resuming snapshot format
To allow an interpreter to discover whether a file was interchange format or not, interchange format was amended to insist that seventh word be 0 for interchange format, non-0 otherwise

System bugs/changes

String copy:
Discussed adding primitives for String copy, and for String compare, because of measured effects on Dolphin.
Discussed using BitBlt instead.
Decided probably this was only one of several places that, depending on the implementation, would need to be sped up; therefore, it would be up to each implementation.

Process primitives:
Discussed whether to make them to be stack rather than queue; did not resolve

CursorLink:
Whether true/false at startup, also discussed the state of the system in general at snapshot resumption/startup. Decided it should be further specified.

Display Size:
Discussed minimal/interchange (startup) display size; did not resolve, but probably should be 640x480 or less and allowable size should/could be determined by a primitive

Larger OT and Object Spaces:

Discussed their ramifications, besides interchange format, also large/small integers, and how this relates to system specification; no resolution

Mouse Bounding:
Whether the cursor could leave the display, and if so by how much; discussed importance to graphics world; did not resolve

BeCursor:
It does not affect the position. What is returned should be specified.

BeDisplay:
Discussed beDisplay especially as it affects remote displays. Noted that having a remote display is a form of caching, and should have the appropriate treatment; in particular, BitMaps are currently accessible only through BitBlt and at:/at:put:, so those primitives must know about the cache.

Initialization of Methods:
Methods should be initialized through the primitive, but the system did not reflect this on June 30. Use of 0 for initial bits is bad for literal frame, 2 (nil) would not hurt.

Perform with arguments:
Could possibly go off the end of the stack in its current specification. Could activate without pushing on current stack first, but this has problems. Compiler could force a large context when perform:withArguments: was sent (although this would only reduce not eliminate probability of failure)

SomeInstance/nextInstance Primitives:
Fail code should be specified.

Reference Counting:
Discussed ramifications of reference counting, such as storing nil when popping the stack, what happens with perform:withArgumentss:, and the process primitives. Decided code in book would be too cluttered to include it all, but people should be careful.

Instantiation of >1 object at a time, as in the messageNotUnderstood code, could cause problems, because it involves storing references to one object in another that may not yet officially "exist" (because it has no references to it yet); again one must be careful

Discussed whether implementors could count on not reference counting various things, such as small integers, and the canned oops. Decided that it was ok to ignore small intergers, probably fine to ignore nil, true, and false, but not really good practice to ignore the others, but let the overflow bit work, and optimize that if necessary.

The @ Primitive:
Specification should be changed to fail for non-SmallIntegers rather than non-Numbers.

**Quo:, rem:, div:, mod:, /, //, \\, **
Discussed that the world has at least two consistent views of integral division and remainder that differ in the way they deal with negative remainders, and that often people implement division from one view and remainder from the other. To satisfy everyone, all possibilities are included in the Smalltalk-80 system.

0 vs. 0.0 in Float code:
Some code would be more optimal if it used 0.0 rather than 0 because of coercion.

Source Code Management:
Discussed current strategy as space-efficient, but dirty.

Discussed that the real solution is to include source code as Smalltalk objects in a virtual memory system, rather than trying to use an external thing (files).

Discussed saving temp names and comments only, rather than all of the code and having a smarter decompiler

Semaphores and Errors:

Discussed out of memory/oops semaphores; decided not to require it.

Discussed the general use of the semaphore mechanism as standard way to signal errors, implementation-specific

Discussed one of these semaphores perhaps being for invalid bytecode, which should never happen, of course

Discussed out of memory condition, whether one could recover in time, and what would be involved. (e.g. soft and hard storage limits).

Discussed that it would be nice to have a convenient facility for adding handlers for various other machine exceptions

System Optimization:

Discussed that new system will have different profiles, should be optimized here as much as reasonable, then up to implementors

Primitive Failure Reporting:

Discussed giving more information to Smalltalk code on why primitive failed; decided this was in the realm of multiple-object returns from methods, and therefore a language research question.

Known/future bugs:

Discussed compatibility vs. extensions, centralized control vs. decentralized, future vs. present

Technical Area--Future Collaborations

Virtual Memory as First Instance

Outlined LOOM design, discussed its transfer (if any) as example of future collaboration

Other stuff

Discussed what benchmarks to use to compare (if desired) implementations or to measure progress of one implementation; collection to be included in a Smalltalk-80 class definition; some attatched. Benchmarks seem to come in "macro" and "micro" styles: e.g., speed of executing 3+4 as micro, and speed to browse as macro. Both would be good

Applications

Areas for major extension, such as multiple superclasses

Non-Technical Areas

Motivations for implementing

Prototyping, analysis of system

Sustaining interest (assumed)

Collection of papers based on experiences

Software: Practice and Experience suggested, desired by some as "real journal, so can justify writing to management"

Our own paperback book, makes it easier to write

Adele volunteered to begin organizing it

First drafts due around February, to give enough time to reflect, but still fresh; final drafts to publisher in Spring

Topics include: "The Xerox Story of Smalltalk-80", "what I did", "what I would have liked to have known that I know now", "what I should have done", and many more

Discussed including (if ok with authors) PASCAL versions of interpreters; transliteration of book vs. transformed for optimization, both would be of interest

Topic could be "how to write a Smalltalk-80 program"/"what is a program in Smalltalk-80"

Future Newsletter

Allan Shiffman from Fairchild wants to make Smalltalk magazine or newsletter
Contributions could include technical issues with solutions sought
Could be forum for challenges in applications/design/language ideas/...

Future Implementors'/Users' Groups

Too early for Users'
Maybe in Spring (as part of the process of finishing the book/collection of articles)
Tek people volunteered Portland

Smalltalk-80 Licenses/Licensees

Adele outlined proposed Xerox licenses for Smalltalk-80; should be available 1st quarter 1982, when book goes to publisher; licensees get pre-publications book copies
Licensees should be involved in future implementors' meetings asap

Other Topics

Writing an article for managers; for *WSJ* or *Forbes*; discussing issues of how Smalltalk aids (does not aid) "productivity" in design/prototype/programming/...
Discussed blocks; recursion and its problems, sharing temps, syntax of args
Mouse buttons and their use; will be three (three states) in interface
Discussed typing as method of browsing