```
to number x y ()
to class x y ()
to vector x y ()
to atom x y ()
to string x y ()
to false x y ()
to float x y ()
"(TITLE USER DO SIZE CODE SELF AREC GLOB MESS RETN CLAS
 length eval or and mod chars
 ".,/;:-[]~|↑!#$%) ←={}*+?<>)
to printing (CODE 38). printing 0.
```

'DONT EDIT ABOVE HERE!--These classes and atoms mentioned early to guarantee addresses for machine code.

On NOVAs, printing 0 turns off listing, printing 1 turns it back on again.

BOOTSTRAPPING MAGIC.

HEREWITH A SOMEWHAT WHIMSICAL ANNOTATED VERSION OF SYSDEFS. ANNOTATIONS ARE IN ITALICS. WHILE IT IS HOPED THAT THIS WILL PROVIDE SOME ELUCIDATION OF THE CODE ESCAPES, OBSCURITIES WILL NO DOUBT PERSIST . THE ANNOTATIONS ARE INTENDED TO BE BUT DIMLY LIGHTED MARKERS ON THE ROAD TO TRUE ILLUMINATION.'

```
to print (%.?() CODE 6)
```

';x.Print its address in octal.
Printing goes to the same place as CODE 20.
This is used primarily for bootstrapping.
All system classes will print themselves.'

```
to read (CODE 2)
```

'Read keyboard input into a vector. This is almost identical in function to the SMALLTALK read routine, except that DOIT is signalled by <CR> at zero-th parenthesis level, and single-quote strings are ignored. It is only available in Nova versions.'

```
to fillin x (↑x. CODE 16 x)
```

'Patches the keyboard input to be read from the file specified. This is only for Novas, and the filename can~t have an extension. fillin"foo will read from FOO. and revert to the keyboard upon EOF.'

```
to STOP (CODE 8)
```

'Writes the entire SMALLTALK workspace out to the file VMEM (on Novas only). Thus when SMALLTALK is next started on that Nova, it will begin with that workspace. It is good to keep a backup copy of VMEM for this reason.'

```
to # (↑#)
```

'Returns a REFERENCE to its argument~s binding.'

```
print #number. print #atom. print #vector. print #string.
print #false. print #class. print #float.
```

'MESSAGE HANDLING'

to ↑ (CODE 18)

```
'     to ↑ name

%" ?(↑"name nil ?(!name←caller message quotefetch)
                 (!caller message quotefetch)

Fetch the next thing in the message stream unevaluated
and bind it to the name if one is there.

%# ?(↑"name nil ?(!name←caller message referencefetch)
                 (!caller message referencefetch)

Fetch the reference to next thing in the message stream
and bind it to the name if one is there.

      (↑"name nil ?(!name← caller message evalfetch)
           !caller message evalfetch)

Fetch the next thing in the message stream evaluated
and bind it to the name if one is there.'
```

to % (CODE 17)

```
'↑"token. token←caller.message.code[caller.message.pc]?
      (caller.message.pc←caller.message.pc+1. !true) !false.
That is, if a match for the token is found in the message, then
gobble it up and return true, else return false.'
```

to ! (CODE 13)

```
'↑x. then do a return, and apply x to any further message.
Note that in (... !x←3. "y←y-2), the assignment to y will never
happen, since ! causes a return.'
```

to " (CODE 9)

```
'!↑". That is, get the next thing in the message
stream unevalled and active return it (which
causes it to be applied to the message).'
```

to isnew (CODE 5)

```
'null instance?("instance←allocate permsize.
      instance[0]←class. !true)
!false).'
```

'CONTROL CLASSES'

to repeat token (:#token. CODE 1)

```
'repeat (token eval) Not a true apply to "eval,
and therefore token MUST be a vector.'
```

to done x (%with?(↑x. CODE 25) CODE 25)

```
'done causes a pop out of the nearest enclosing repeat , for, or do.
"done with val" will cause the repeat to have value val'
```

to again (CODE 6)

```
'repeat ("active←active caller. eq active.
class #repeat?(done)). That is, redo the most
recent repeat, for, or do loop.'
```

```
to if exp (:exp?(%then?(:exp. %else?(:". exp)exp)error "(no then!))
        %then?(:".  %else?(:exp) false)error "(no then!))

    'The ALGOL "if ... then ... else ..."'

to for token step stop var start exp (
    :"var. (%=?(:start.)"start←1.) (%←?(:start))
    (%to?(:stop.)"stop←start.)
    (%by?(:step.)"step←1.)
    %do. :#exp. CODE 24)

    'An Algol-like "for". Note the default values if
    "s","to","by",etc., are omitted.
    CODE 24 means --repeat(exp eval).
    This implies "done" and "again" will work,
    which is correct.'

to do N exp (:N. :#exp. for N to N do (exp eval.))

    '"N" is made available and contains the current
    iteration number.'
```

'INITIALIZING SYSTEM CLASSES'

```
    'Here are the main kludges which remain from
    the time when we really didn't understand classes
    very well, but wanted a working SMALLTALK.
    PUT and GET are two of the principle actions of class
    class. The new verson of SMALLTALK will have
    class as a class with these actions intensional.'

to PUT x y z (:#x. :y. :z. CODE 12)

    'The first argument MUST be an atom which is bound
    to a class table. The third argument is installed
    in the value side of that table corresponding to the
    name (atom) which was the second argument.'

to GET x y (:#x. :y. CODE 28)

    'If "x" is a class table then the binding of
    the atom in "y" will be fetched.'

to leech field bits : ptr (
    isnew?(:ptr)
    %[? (:field. %]. "bits←(%&?(1)0) CODE 27))

    'Lets you subscript any instance
    a[0] gives you the class, a[1] gives the first field, etc.
    a[2] gives you the pointer; a[2]& returns the BITS in an integer
    a[2]←foo will dereference count previous contents,
    but a[2]&←foo will not.'

PUT USER "TITLE "USER

PUT atom "DO "(CODE 29

    '%←?(:x. !x -- Lookup SELF and replace its value by x.)
    %#?(! -- Lookup the binding of SELF)
    %=?(!SELF=!)
    %chars?(! -- printname of SELF (a string))'

    %is?(%atom?() %??(!"atom) :". !false)
    %print?(disp←SELF chars) )
```

```
'Done this way (PUT used rather than using "to") because
we wanted to know where the system classes are.
Hence the initial "to atom x y ( )" , for example,
in "Bootstrapping Magic" followed by the
behavior here.'

'THE SMALLTALK EVALUATOR!!!!!!!!!'

    to ev (repeat (cr read eval print.)). "cr←6

'Later in the boot there will be "to cr(disp←13)",
which will actually do carriage returns when
a display frame has been instanced.'

PUT false "DO "(CODE 11

    '%?? (:".)
    %or? (!:)
    %and? (:.)
    %<? (:.)
    %=? (:.)
    %>? (:.)'

    %is?(%false?(!true) %??(!"false) :".)
    %print?("false print) )

PUT vector "DO "(CODE 3 ?(!Substr SELF x GLOB MESS)

    'isnew?(Allocate vector of length :.
        Fill vector with nils.)
    %[?(:x. %].
        (%←?(:y. !y -- store y into xth element. )
        ! xth element) )
    %length?(! length of string or vector)
    %eval?("pc←0. repeat
        (null SELF["pc←pc+1]?(done)
        "val←SELF[pc] eval)
        !val) sort of...'

    %is?(%vector?() %??(!"vector) :". !false)
    %print?(disp←40. for x to SELF length
        (disp←32. SELF[x] print). disp←41) )

PUT string "DO "(CODE 3 ?(!Substr SELF x GLOB MESS)

    'isnew?(Allocate string of length :.
        Fill string with 0377s.)
    %[?(:x. %].
        (%←?(:y. !y -- store y into xth element. )
        ! xth element) )
    %length?(! length of string or vector)'

    %is?(%string?(·) %??(!"string) :". !false)
    %print?(disp←39. disp←SELF. disp←39)
    %=?(:y is string?(SELF length=y length?(
        for x to SELF length (SELF[x]=y[x]?() !false)) !false)
        error "(string not found))
    %+?(:y is string?("x←SELF[1 to SELF length+y length].
        !x[SELF length+1 to x length]←y[1 to y length])
        error "(string not found))
    %eval?(! (vectorize SELF) eval)
    %fill?("x←1. repeat(
        16=SELF[x]←disp←kbd?(done)
        "x←x+(SELF[x]=8?(x=1?(6)-1)1)
        x>SELF length?(done))) )
```

```
to Substr op byte s lb ub s2 lb2 ub2 (
      !#s. !lb. !#GLOB. !#MESS.
      !ub.(%]?() error "(missing right bracket))
      "byte ← "lb2 ← "ub2 ← 1.
      %find? ("op ← (%first?(1) %last?(2) 1) + (%non?(2) 0). !byte. CODE 46)
      %←? (%all? (!byte. "op←0. CODE 46)
            !#s2. "op←5.
            %[? (!lb2. %to. !ub2. %]. CODE 46)
                  "ub2←30000. CODE 46)
      "op ← 6. "ub2 ← ub+1-lb.
      "s2 ← (s is string?(string ub2) vector ub2). CODE 46)

    'Substr takes care of copying, moving and searching
      within strings and vectors. It first gets its father (string/vector)
      and the lower bound, and then proceeds to fetch the rest of the
      message from above. Some examples:
            "(a b c d e)[2 to 3] -> (b c)
            "(a b c d e)[1 to 5] find "c -> 3
            "(a b c d e)[1 to 5] find "x -> 0'

to vectorize reader ("reader ← sequence !reader 0. lread)

PUT number "DO "(CODE 4

      '%+?(!val+:)
      %-?(!val-:)
      %*?(!val*:)
      %/?(!val/:)
      %<?(!val<:)
      %=?(!val=:
      %>?(!val>:)
      %&?(%+?(!val OR :)
            %-?(!val XOR :)
            %*?(!val AND :)
            %/?(!val LSHIFT :))'

      %is?(%number?() %??(!"number) !". !false)
      %print?(disp←Nprint SELF 10)
      %base?(!Nprint SELF !x) )

      'For floating point stuff see FLOAT'

to - (0-!)

      'An often used abbreviation.'

to Nprint r n l s !! pname (%knows?(ev):n. !r.) "l←18.
      (n<0?("s←1. "n←0-n) "s←0)
      repeat (pname["l←l-1] ← 48+n-r*"n←n/r. n=0?(done))
      (s=1?(pname["l←l-1]←45)) !pname[l to 17])
Nprint knows
"pname←string 17. done
```

'TURTLE COMMANDS'

```
to right (CODE 21)
to go (CODE 22)
to goto (CODE 36)
to TURTX x (!x. CODE 23)
to ink x (!x. CODE 32 x)
to penup (TURTX 0)
to pendn (TURTX 1)
to home (TURTX 2)
to up (TURTX 3)
```

```
to erase (TURTX 4)
to black (ink 0-1)
to white (ink 0)
to turtle z : pen ink dir x xf y yf dx dxf dy dyf (
      isnow?(0 CODE 39)
      %knows?(!(:".)eval)
      1 CODE 39 :z. 0 CODE 39 !z)
```

2{ '0: *Load SELF from THEturtle state.*
   1: *Load THEturtle state from SELF.*

UTILITIES'

```
to mem x y (:x. CODE 26)
      'to mem x y (:x. %←?(!core/mem x ←:)!core/mem x)
      mem loads integers from and stores them into real core.
      Tee hee...
      mem 0430 ← 0  ;set alto clock to zero
      mem 0430  ;read the clock
      for i to 16 (mem 0430+i ← cursor[i])  ;put new bits into cursor
      mem 0424 ← mem 0425 ← 0.  ;reset mouse x and y to 0.
      mem 0102 ← 0.  ;disconnect cursor from mouse
      mem 0426 ← x. mem 0427 ← y.  ;move the cursor
      mem 0100 ← 0177.  ;make DEL the interrupt char (instead of ESC).
      mem 0420.  ;get pointer to display control block
      mem 0177034.  ;reads the first of 4 keyboard input words.'

to trace (CODE 34)

      'For speed readers only. Causes a printout
      of <activation rec><pc><class><token> . A toggle
      -- trace sets it, trace unsets it.'

to mouse x (:x. CODE 35)

      ' x = 0-7  are a map on the mouse buttons.
      E.g. (4=mouse 4) comes back true if the top mouse
      button is depressed, (1=mouse 1)) comes back true
      if bottom mouse button depressed, (7=mouse 7))
      comes back true if all three mouse buttons depressed,
      etc. Mouse 8 returns the x coordinate of the
      mouse and mouse 9 returns the y coordinate.'

to core ((mem 59)-mem 58)

      'Returns the amount of space left in your Smalltalk.'

to nil x (#x)

      'nil is an "unbound pointer", which is used
      to fill vectors and tables.'

to null x (:x. 1 CODE 37)

      'Null returns true if its message is "nil",
      otherwise false.'

to error x (:x print. CODE 14)

      'Print class, caller and context (a la CODE 20) and then pop
      all the way up to the top level.'
```

```
to kbd (0 CODE 20)
```

'*Waits until a key is struck.*
Returns an ascii code when a key is struck on the keyboard.
Use to kbck (1 CODE 20) to return true if kbd has a character,
otherwise false.
Used in multiprocessing.'

```
to disp x i (
    %←?(ix is string?(for i to x length (TTY←x[i])) TTY←x)
    %clear.)
```

'*This disp is used for bootstrapping.*
Later in those definitions (READER) it will
be restored to an instance of "display frame."'

```
to TTY (0 CODE 20)
```

'*TTY←⟨integer⟩ will print an ascii on the Nova tty.*
On altos, TTY prints in little error window at bottom
of screen.'

```
to dsoff (mem 272←0)
```

'*Turns display off by storing 0 in display control block ptr.*
Speeds up Alto Smalltalk by factor of 2.'

```
to dson (mem 272 ← mem 62)
```

'*Turns display back on by refreshing display*
control block pointer.'

```
to apply x y (i#x. %to?(iy. CODE 16) CODE 16)
```

'*Causes its argument to be applied to the message stream of the*
caller, or, in the case of apply foo to ⟨vector⟩, to that vector.
Note that only the message is changed, and that the caller is
not bypassed in any global symbol lookup. Dan knows that
this really needs to also allow ⟨contexts⟩ and ⟨returns⟩ to be
specified.'

```
to cr (disp←13). to sp (disp←32)

"true←"true
"oval←"oval
to is ( %??(i"untyped)i". ifalse)
```

'*Those are used to handle messages to classes which*
can't answer question invoking "is", "eval", etc.'

```
to vecmod new end old posn ndel nins ins ("end←10000.
    iold. iposn. indel. iins.
    "nins←(ins is vector?(ins length-1) null ins?(0) 1).
    "new ← old[1 to old length+nins-ndel].
    (ins is vector?(new[posn to end] ← ins[1 to nins]) new[posn]←ins).
    new[posn+nins to end] ← old[posn+ndel to end].
    inew)
```

'*Vecmod makes a copy of old vector with ndel elements deleted*
beginning at posn. If ins is a vector, its elements are inserted
at the same place. It is the heart of edit.'

```
to addto func v w (i#func. iw. "v←GET func "DO. null v?(error "(no code))
    PUT func "DO vecmod v v length 0 w)
```

'*Addto should look to see if "(% .. ?(" is there.*'

'READING SMALLTALK'

```
to sequence : str ptr (
      %scan?(CODE 41?(!rpar))
      %pos?(%←?(!:ptr)!ptr)
      %fill?(str fill. "ptr←0)
      %is?(%sequence?() %??(!"sequence) :". !false)
      !snew?(:str. :ptr) )

      'Sequences are used to build the main read routine.
      CODE 41 advances pointer to the end of the next token in string,
      and returns the internal representation of that token.'

to read v s : i ownv lpar rpar subread growread (%knows?(ov)
      "v←ownv. "s←v length. !subread 1)
read knows
to subread i j t ("j←:i.
      repeat("t←roador scan.
            rpar=t?(v[j]←nil. done with v[i to j])
            lpar=t?("t←subread j. v[j]←t. s < "j←j+1?(growread))
            v[j]←t. s < "j←j+1?(growread) ))
to growread ("v←v[1 to "s←s+ (200>core?(50)s)] )
"ownv←vector 200.
"lpar←(sequence (string 1) fill 0) scan
( "rpar←(sequence (string 1) fill 0) scan
) done

to obset i input : vec size end (
      %←?(0=vec[1 to end] find first :input?
            (end<size?(vec["end←end+1]←input)error "(obset full)))
      %delete?(0="i←vec[1 to end] find first :input?()
            vec[i to end]←vec[i+1 to end+1]. "end←end-1)
      %print?(SELF map "(vec[i] print. sp))
      %map?(:input. for i to end (input oval))
      %is?(%obset?()%??(!"obset) :". !false)
      !snew?("end←0. "vec←vector !size+1)
      )
```

'SHOWING SMALLTALK'

```
      'A more grandiose "show" follows after class dispframe
      (see READER) is defined.'

to show func t (
      :#func. "t←GET func "DO.
      null t ? (!"(no code)) pshow t 0.)
to tabin n i (disp←13. for i to :n (disp←32))
to lpar (disp←40)
to rpar (disp←41)
to pshow ptr dont i (
      !ptr :dont.
      ptr length<3?(lpar. ptr[1] print. rpar)
      tabin dont. lpar.
      for i to ptr length-1
            (ptr [i] is vector ?(pshow ptr[i] dont+3.
                  i=ptr length-1?() ptr[i+1] is vector?() tabin dont-1)
            (i=1 ?() sp)
            ptr[i] print)
      rpar)
```

'NOVA FILE MAINTENANCE -- DP0:SYSDEFS.AN DP1:SYSDEFS.AN '

```
to sscan x (ix. ix) '*For auld langs syne.*'
printing 1. "version←"(version 6/28). STOP.
```

printing 0

'THE TRUTH ABOUT FILES

FILESMALL -- SMALLTALK file system 7/74

a file is found in a directory ("dirinst") by its file name ("fname"), and has a one "page", 512 character string ("sadr").

"f1 ← <directory> file <string> old finds an old file named <string> in <directory> or returns false if does not exist or a disk error occurs.

"f1 ← <directory> file <string> new creates a new file or returns false if it already exists. if neither old or new is specified, an existing file named <string> will be found or a new file created. if <directory> is not specified, the current default directory is used.

<directory> file <string> delete deletes a file from a directory and deallocates its pages. do not delete the system directory (SYSDIR.) or bittable (SYS.STAT.), or any directories you create.

<directory> file <string> load loads a previously "saved" SMALLTALK virtual memory, thereby destroying your current state.

<directory> file <string> save saves SMALLTALK virtual memory.

"leader" and "curadr" are the alto disk addresses of page 0 and the current page of the file, respectively. "bytec" is a character index into "sadr".

"dirty" is (should be) 1 if any label block integers ("nextp" thru "sn2") have been changed; -1 if "sadr" has been changed; 0 if the current page is clean. the user need not worry about this unless (s)he deals directly with the label or "sadr".

"status" is normally 0; -1 if eof occurred with the last "set"; a positive number (machine language pointer to offending disk command block (dcb)) signals a disk error -- passively noted at present.

the next 8 integers are the alto disk label block. "nextp" and "backp" are the forward and backward alto address pointers. "lnused" is currently unused. "numch" is number of characters on the current page, numch must be 512, except on the last page. "pagen" is the current page number. page numbers are non-negative integers, and the format demands that the difference in consecutive page numbers is 1. normal file access starts at page 1, although all files possess page 0 (the "leader" page). "version" numbers > 1 are not implemented. "sn1" and "sn2" are the unique 2-word serial number for the file. bits in "sn1" flag directories and random files.

the class function "ncheck" checks that file names are strings, contain no "illegal" characters, and terminate with a period .'

```
(to file : dirinst fname sadr leader curadr bytec dirty status nextp
      backp lnused numch pagen version sn1 sn2 : ncheck illegal (

    %←?  (17 CODE 50)

        'f1←<integer>, <string>, or <file> --
        ;x is string? (for i to x length (SELF←x[i]))
        x is file? (repeat (x eof? (done) SELF←x next))
        (numch<"bytec←bytec+1?
          (SELF set to write (pagen←bytec/512) bytec mod 512))
        sadr[bytec]←x &* 0377'
```

```
%next?      ((%into? (16)

                'fi next into <string> -- read a string
                for i to :x length(x[1]+SELF next).!x'

            %word? (%+? (7)

                'fi next word+<integer> -- write integer
                SELF + :x/256. SELF + x mod 256.
                (currently ignores word boundaries)'

                 6)

                'fi next word -- read an integer
                !(SELF next*256) + SELF next'

            %char. 0) CODE 50)

                'fi next or fi next char -- read a character
                (numch<"bytec+bytec+1?
                 (SELF set to read (pagen+bytec/512)
                  bytec mod 512? ( ) !0)) !sadr[bytec]'

%eof?       (10 CODE 50)

        'fi eof -- return false if end of file has not
        occurred. nextp=0? (bytec<numch? (!false))!false'

%flush?     (12 CODE 50)

        'fi flush -- dirty=0? ( ) write current page'

%set?       (%to. (%eof?(13)

                'fi set to eof -- set file pointer to end
                of file. SELF set to read 037777 0'

            %write?(5)

                'set to write <integer> <integer> -- set
                file pointer to :spage :schar. if current page
                is dirty and pagen not equal to spage, flush
                current page. read pages until pagen=spage.
                allocate new pages after eof if necessary
                (-1 512 is treated as start of next page,
                i.e. pagen+1 0). "bytec+schar'

            %read. 4) CODE 50)

                'same as "write" except stop at eof'

%skipnext? (18 CODE 50)

        'fi skipnext <integer> -- set character pointer
        relative to current position. (useful for skipping
        rather than reading, or for reading and backing up,
        but eof may not work if "bytec" points off the current
        page) "bytec + bytec + :offset'

%pages?     (20 CODE 50)

        'fi pages <integer> ... <integer> -- do "direct" disk
        i/o (not for the faint-hearted). :coreaddress.
        :diskaddress. :diskcommand. :startpage. :numberofpages.
        :coreincrement. if coreaddress=-1, copy "sadr" to a
        buffer before the i/o call, and copy it back to "sadr"
```

```
        after the call. diskaddress and diskcommand are the alto
        disk address and command.  startpage is relevant if label
        checking is performed.  numberofpages is the number of
        disk pages to process. coreincrement is usually 0 (for
        writing in same buffer) or 256 for using consecutive
        pages of core.  use label block from instance of "fi".
        perform i/o call.  return false if error occurs.'

    %is?  (%file? () %~? (!"file) :". !false)

    %print?     (disp ← fname)   'file prints its name'

    %rewind? (11 CODE 50)

        'fi rewind -- reposition to beginning of file
        SELF set 1 0'

    %clear?     ()

            'disp clear -- imitates dispframes for
            show in filout'

    %ovals?     (!(:") oval)

    %knows?     (ov)

    %close?     (dirinst evals(bitinst flush.filesopen) delete SELF.
        SELF flush. !"closed)

        'fi close or "fi←fi close (necessary to release
        instance) -- flush current page, remove instance from
        "filesopen" list of directory.'

    %shorten? (%to. 14 CODE 50)

        'fi shorten to <integer> <integer> -- shorten a file
        SELF set to read :spage :schar. "x←nextp. "nextp←0.
        "numch←schar. "dirty←1. deallocate x and successors'

    isnew?      (("fname←ncheck (:)? () error "(bad file name))
          (("dirinst←curdir) is directory? ()
           ("dirinst←directory evals(defdir))is directory?
            (directory evals(defdir)open) error"(illegal directory))

              'set directory instance for file. if curdir
              is not a directory (null global value because
              file was not called from the context of a
              directory instance), use the default directory'

          %delete? (15 CODE 50. !"deleted)

              'delete a file (see intro)'

          "sadr← string 512.
          (%new?(dirinst evals(filinst)is file?(3)19)
           %old?(2)1) CODE 50.

              'allocate string buffer. find an old file or
              add a new entry (see intro). directories have
              a special "sn1". machine code may return false'

          %load?      (8 CODE 50)
          %save?      (9 CODE 50.
                directory evals("curdir←0."defdir←
                     "dp0←directory dirname))
```

```
            'load returns via "save". virtual memory on
            file should have no active files or directories;
            dp0 is reinitialized upon load. how to reopen
            other files (e.g. DRIBBLE)?'

        dirinst evals(filesopen)←SELF

            'file puts itself into the filesopen list
            of its directory' ) ))
file knows
to ncheck i str (
    (:str is string? (0<str length<255? () !false) !false)
    for i to str length (str[i]<33? (!false)
     0<illegal[1 to 99] find str[i]? (!false))
    str[str length]=46? (!str) !str+". chars)

'check that file name is a string and of proper length and contains
no illegal characters. if name does not end with a period, append one'

("illegal←string 11) fill
[]().,;<>*#"
done


'a directory is found in a directory ("dirinst"), has a bittable file
("bitinst") for allocating new pages, a file of file entries ("filinst"
-- file names, disk addresses etc.), and a list of currently open files
("filesopen" which is an "obset"). the top level, "distinguished node"
of the directory structure is the system directory "dp0" (see "directory
knows" below if you also want "dp1"). dp0 knows the disk number ("dirinst")
and the true identity of the bittable. each file must ask its directory
for the bittable when page allocation is necessary, and the system
directory (via its local directory) for the disk number.

"di ← <directory> directory <string> old/new

currently, <directory> and old or new must be specified.

"dirname" is the system directory name and "bitname" is the bittable name.
"curdir" is a class variable bound to the last directory instance "opened",
and provides information "who called you" (i.e. CALLER) to a file or
directory. "defdir" is a default directory, initially set to dp0, which
is invoked when "curdir" fails to a directory, i.e. file was not called
in the context of a directory, but globally'

(to directory ch : dirinst bitinst filinst filesopen : dirname bitname
    curdir defdir (
    %file?      (SELF open. !apply file)

        'di file <string>... -- open directory. create file
        instance'

    %directory? (SELF open. !apply directory)

        'di directory <string>... -- open directory. create
        directory instance'

    %open?      ("curdir←SELF. filinst is file? ()
        (bitinst>0? ("bitinst←dirinst evals(bitinst).
         "filinst←file filinst new) "filinst←file filinst old.
         "bitinst←(dirinst is directory? (dirinst evals(bitinst))
          file bitname old))
         filinst evals(sn1)>-1? (error"(bad directory sn))
         dirinst is directory? (dirinst evals (filesopen)←SELF))

            'di open -- initialize directory file and
```

```
            'bittable instances. directory (except for "top
            level") puts itself into filesopen list of its
            directory'

    %flush? (filinst is file? (filesopen map "(vec[i] flush).
        "filesopen←obset 10. "filinst←filinst evals(fname).
        bitinst flush. "bitinst←-1))

            'di flush -- (normally not user called). if
            directory is open, flush it by flushing all files
            and directories in its filesopen list. store
            file name in "filinst", and "old" in bitinst so
            that it can be reopened'

    %close?  ((dirinst is directory? (dirinst evals(filesopen)
             delete SELF)) SELF flush. !"closed)

            'di close (e.g. dp0 close) or "di←di close
            (to release instance) --close a directory by
            deleting it from the filesopen list of its
            directory (except for the system directory).
            and flushing it'

    %list?      (SELF open. filinst rewind. disp clear.
        repeat (filinst eof? (cr. done)  0=1024 &*
        "ch←filinst next word? (ch<2? () filinst skipnext 2*ch-1)
        filinst skipnext 10. "ch←filinst next &+ 1.
        disp ← filinst next into string ch. sp) )

            'di list -- print the entry names contained in filinst'

    %ls? (%directory? () %~? (!"directory) :". !false)

    %print?     (disp←0133. filesopen print. disp←0135)

            'di or di print. --print the filesopen list'

    %use?       ("defdir←SELF)  'di use -- change the default directory'

    %evals?     (!(:") eval)

    %knows?     (ev)

    isnew?      ("filesopen←obset 10. "dirinst←curdir.
          dirname=:filinst? ("bitinst←-1. "curdir←SELF)
          "bitinst←(%new?(1) %old. -1). SELF open)

            'default directory will not work here unless
            "curdir" is "unset" upon exit. only thing
            special is that the system directory is not
            opened the first time through' ))
directory knows
("dirname←string 7) fill
SYSDIR.
("bitname←string 9) fill
SYS.STAT.

    'names for the system directory and bittable'

"curdir←0. "defdir←"dp0←directory dirname.

    'create the system directory instance (the initial default)
    on disk 0 in a "closed" state. to initialize a second disk:
    directory knows
    "curdir←1. "dp1←directory dirname.
    done'
```

```
done
"curdir←nil.
        'to prevent "curdir has no value, I was in
        file..." when default is desired'

'some "fillin-able" disk utilities

type.       types a file, 30 characters at a time, into "disp"
    e.g. type <file> or <string = file name>

dskstat.    prints the number of free/used pages on your disk
    e.g. dskstat

ftype.      types a file, 512 characters at a time (=1 disk page),
    into a display frame at the top of the screen. <space>
    increments the page number; <return> exits, any other
    character decrements the page number. the current page
    number is printed in "disp".
    e.g. ftype <file> or <string = file name>

booter.     boot a file (via software). alleviates holding down keys.
    an oft-used boot file should be installed, see below.
    e.g. boot <file> or <string = file name>
        booter <integer = disk address of page 1 of boot file>

install.    "installs" boot file at a designated disk location by
    copying page 1 of the boot there if the page is free. if
    the page is "owned" by another file, the serial number
    of that file is printed, and you must type <doit> to
    'overwrite; anything else to abort.
    e.g. install <file> or <string = file name> at
        <string = "bootkeys"> or <integer = disk address>'

to [ i x s (:x. "s←string 6. for i to 6(
    s[7-i]←48+x &* 7. "x ← x &/ -3). !s)
to bl i s n (:s. :n. for i=s to s+n-1 (([mem i)print.sp.))

'temporary debugging aids'

printing 1
```

printing 6.

'TEXT DISPLAY ROUTINES'

'Display frames are declared with five parameters. They are a left x, a width, a top y, a height, and a string. Hence -- "yourframe←dispframe 16 256 16 256 string 400. -- gets you an area on the upper left portion of the display that starts at x,y 16,16 and is 256 bits(raster units) wide and 256 bits high. The string (buf) serves as the text buffer, and is altered by ← and scrolling.

[N.B. X~s and width~s are driven to a multiple of 16 value in the machine code. X~s resolve to the floor (e.g. if x is 15 going in, it will be 0 on the display. Width~s resolve to the ceiling (e.g. if a width goes in as 249, it will come out on the display as 256.)This restriction may be relieved in the future.]

There are actually two entities associated with display frames--frames and windows. Currently both are given the same dimensions upon declaration (see isnew).

The four instance variables defining the window are "winx", "winwd", "winy", and "winht". The boundaries of this rectangle are intersected with the physical display. The window actually used by the machine language will reduce the size of the window, if necessary, to be confined by the physical display. Clipping and scrolling are done on the basis of window boundaries. If a character is in the window it will be displayed. If a string or character cause overflow of the bottom of the window, scrolling will occur.

The four instance variables defining the frame are "frmx", "frmwd", "frmy", and "frmht". This rectangle may be smaller or larger than its associated window as well as the physical display. Frame boundaries are the basis for word-wraparound. (Presently, if frmy+ frmht will cause overflow of the window bottom[winx+winht], frmht will get changed to a height consonant with the bottom of the window. This has been done to manage scrolling, but may get changed as we get a better handle on the meaning of frames and windows.).

"Buf" is the string buffer associated with any given instance of dispframe. This is the string that is picked on the way to microcode scan conversion. When scrolling occurs, the first line of characters, according to frame boundaries, is stripped out and the remainder of the buffer mapped back into itself. If a "←" message would overflow this buffer, then scrolling will occur until the input fits.

"Last" is a "buf" subscript, pointing to the current last character in the buffer. That is, the last character resulting from a "←".

"Lstln" also points into the buffer at the character that begins the last line of text in the frame. It is a starting point for scan conversion in the "←" call.

"Mark" is set by dread (see below) and points to the character in the buffer which represents the last

windows

winx winwd
winy winht

clipping

scroll ← , overflow string buffer
the window ⇒ (displayed)

frame

frmx frmwd
frmy frmht

frame < w > window
word wraparound w/r frame

string buffer (buf)
Last (last char)

LstRin (last line)

mark (last prior input)

prompt output by SMALLTALK; reading begins there. Mark is updated by scrolling, so that it tracks the characters. One could detect scrolling by watching mark.

"Charx" and "chary" reflect right x and top y of the character pointed to by "last".

The "reply" variable in the instance may be helpful in controlling things. When the reply is 0, it means everything should be OK. That is, there was intersection between the window and display and intersection between the window and the frame. When reply is 1, there was no intersection between the window and the display. A 2 reply means no intersection between window and frame. A 11 means that the frame height has been increased in order to accomodate the input. A 12 means the bottom of the window (i.e. window x + window height) has been overflowed --hence that scrolling took place. A 13 means that both 11 and 12 are true.'

```
(to dispframe input
    t winx winwd winy winht frmx frmwd frmy frmht buf
    last mark lstin charx chary reply editor
    t sub frame dread reread (
% ← ?(0 CODE 51)
```

'is. s is number ? (append this ascii char)
s is string ?(append string)
error.'

```
%param?(!(1".)eval)
```

'Allows access to instance variables. For example,
yourframe param ("winx←32)
will alter the value of window x in the instance of dispframe called "yourframe".'

```
%show?(4 CODE 51 3 CODE 51)
```

'Show clears the intersection of window and frame (see fclear, below) and displays buf from the beginning through last. A handy way to clean up a cluttered world.'

```
%fclear?(4 CODE 51)
```

'Fclear clears the intersection of the window and frame. Hence if the frame is defined as smaller than the window, only the frame area will be cleared. If the frame is defined as larger than the window, only the window area will be cleared, since that space is in fact your "window" on that frame.'

```
%wclear?(5 CODE 51)
```

'Wclear clears the intersection of a window and the physical display.'

```
%scroll?(2 CODE 51)
```

'Scroll removes the top line of text from the frame~s string buffer, and moves the text up one line.'

```
%clear?(1 CODE 51)
```

'Clear does an fclear and sets the "last" pointer into the string buffer to 0 and "lstin" to 1. It has the effect of cleaning out the string buffer as well as clearing the

frame area.'

%mfindc ?(7 CODE 51)

'Returns the number of the character corresponding to the x and y passed. Sample call --
"a←yourframe mfindc mouse 8 mouse 9.
A -1 return means x,y after end of string.
A -2 return means x,y not in frame.'

%mfindt ?(6 CODE 51)

'Like mfindc except returns a token number. Tokens in this context are taken to be separated by a blank or carriage return. Hopefully, it will soon be smarter about multiple spaces and/or cr-s. A sample call--
"variable←yourframe mfindt mouse 8 mouse 9.
-1 and -2 returns same as for mfindc.'

%read?(!dread)

'Makes a code vector out of keyboard input.
See dread below.'

%reread?(!reread !)

'Used by redo and fix. Goes back n(its argument), prompts and does a read from there.
See reread below. '

%sub?(sub !. SELF show)

'Evals its argument in a sub-window. Used by fix and shift-esc. See sub below.'

%knows?(ev)

'Whilst at the KEYBOARD, one can say "yourframe knows(DOIT)" and get a copy of the evaluator in the context of that instance of dispframe. Allows access to instance variables without going through the param path. '

%frame ? (apply frame)

'Draws a border of the given color around the frame. E.g.,
yourframe frame -1.'

```
!snow ? ("winx←!frmx. "winwd←!frmwd. "chary←"winy←!frmy.
    "winht←!frmht. !buf. "lstin←1.
    "mark←"last←"charx←"reply←0. frame -1) ))

dispframe knows
to dread reader t flag (
    disp←20. DRIBBLE flush. "flag←false. "mark←last.
    repeat (050>disp←"t←kbd?(
        t=010?(last<mark?(disp←buf[last+1])
        'Backspace only up to prompt.'
            buf[last+1]=047?("flag←flag is false))
            'Backspace out of string flips flag.'
        t=012?(flag?() done)
        'DOIT checks if in a string.'
        t=047?("flag←flag is false)
        'Flag is true if in a string'
        t=023?(sub "(ev). "last←last-1. disp show)
        'Shift-Esc make sub-eval.'
```

```
      ))
      disp←13. TTY←-1. "reador ← sequence buf mark. lread)
      'TTY ← -1 clears error window.'
to sub disp (
      "disp←dispframe winx+48 winwd-64 winy+14 winht-28 string 300.
      disp clear. (:)eval)

      'Opens a sub-frame, and evals its argument
      in that context.'

to frame a t ("a←turtle. 'save turt' ink t. penup goto winx-1 winy-1
      pendn goto "t←winx+winwd winy-1 goto t "t←winy+winht
      goto winx-1 t goto winx-1 winy-1. a 1 'restore turt')

      'Draws a line around the frame, one unit outside.'

to reread n i p reador (
      "p←mark. for i to n
            ("p←buf[1 to p-1] find last 20.
            p<1?(done))
      i<n+1?(error "(no code))
      "reador ← sequence buf p. lread)

      'Counts back n prompts (n is integer arg)
      and then does a read from there.'

done

to dclear (CODE 52)

      'This function takes four parameters -- x width y height,
      and clears the display rectangle thus defined. X will be
      taken to closest multiple of 16 [floor] unless already
      a multiple of 16. Also true for width, except
      taken to ceiling.'

to dmove (CODE 53)

      'This function takes six parameters -- source x width
      source y height destination x destination y. It takes the
      source rectangle (x and width mod 16-d as in dclear) and
      moves it to the destination x and y. Clipping will occur on
      display boundaries. The source will remain intact unless
      it overlaps with the destination, in which case the over-
      lapping portion of the destination wins.'

to dmovec (CODE 54)

      'Dmovec takes the same parameters as dmove, but
      in addition clears the non-intersecting source material.
      It is the general case of what happens on the display
      screen during a scroll, i.e. scrolling could be accomplished
      by saying  disp param (dmovec winx winwd winy+fontheight
      winht-fontheight winx winy).  A sample call --
      dmovec 0 256 0 256 256 256. This will move
      whatever is in the upper left hand corner of the
      display  to x,y  256,256 -- and then erase the
      source area. '

to redo (disp param ("last←mark-2). (disp reread :) eval. disp show.)

      'Causes re-evaluation of the input typed n prompts
      before this.  Setting last←mark-2 makes the redo
      statement and its prompt disappear with a disp show.'
```

```
to fix vec (disp param ("last-mark-2). "vec-disp reread t.
      (disp sub "(voced vec)) eval)

      '(Like redo, except that the previous input is given
      to the editor in a subwindow. When editing is done,
      the resulting code is evalled before returning.)

'Nova file maintenance.   DP0:DISPLAY.AN  DP1:DISPLAY.AN '
printing 1
```

```
printing 0.

to kbck (1 CODE 20)
      'Returns true if the keyboard has been hit'

to button n (!:n=mouse 7)
      'Returns true if that pattern is being held down'


to edit t (:#func.
      "t←GET func "DO.
      null t ? (!"(no code))
      PUT func "DO voced t.
      !"edited)

      'Edit picks up a code vector, makes sure it is not empty
      and calls voced to do the work.

      Voced can be used on any vector, and is used by FIX as well
      as EDIT.  It creates two new windows within the default DISP
      which exists when it is called.  One is used for a menu of
      commands, the other becomes the new default window DISP.
      The new default is passed to an intermediary, and the newly
      edited vector is returned.'

(to voced back nowdisp  menu x :t  menuwidth menulength menuvec
ed ed1 edpush edtarget gettwo getcommand wait44 waitfortokenin  (
      %knows?(ev)
      %evals?(!(:")eval)
      "back←false.
      disp fclear.
      disp param ("menu←dispframe winx+winwd-menuwidth menuwidth winy winht
                  string 70.
      mem 0425 ← winy + winht/2.
      for x to menulength do (menu←32. menu ← menuvec[x] chars.  menu←13)"
            "nowdisp ← dispframe winx winwd-menuwidth winy winht
                  string buf length - menu param(buf length))
      :x.
      "x ← ed nowdisp x.
      disp show.
      !x)  )


voced knows

"menuwidth ← 64.
"menuvec  ← "( Add Insert Replace Delete Move Up Push Enter Leave Exit)
"menulength ← 10.

to ed disp (:disp. apply ed1)
      'Ed is used to make DISP a new local.'

to ed1 ptr l n nrun command temp  t oldx oldy (
      "command ← 0.
      :ptr.
      repeat(
            "l←ptr length.
            back?(done with ptr)
            "oldx ← mouse 8.
            "oldy ← mouse 9.
            menu param (mem 0424 ← winx + winwd/2.
                  mem 0425 ← winy + winht/2)
            (command=126?(disp←126)
                  disp clear
                  for n to l-1
                        (disp←32.
```

Needed

```
                ptr[n] is vector?(disp←044)
                ptr[n] print))
        cr. disp←052.

        menuvec["command ← getcommand] print.
        mem 0424 ← oldx.
        mem 0425 ← oldy.

        "(
        ("ptr←vecmod ptr 1 0 read)
        ("ptr←vecmod ptr edtarget 0 read)
        (gettwo. "ptr←vecmod ptr n nrun read)
        (gettwo. "ptr←vecmod ptr n nrun nil)
        (gettwo. "temp ← ptr[n to n+nrun]
              temp[nrun + 1] ← nil.
              "i←edtarget.
              "ptr←vecmod ptr n nrun nil.
              (i>n ? ("i←i-nrun))
              "ptr←vecmod ptr i 0 temp)
        (ptr["n←edtarget] is vector ?
                ("ptr←vecmod ptr n 1 ptr[n])
                "command ← 126)
        (gettwo. edpush)
        ( ptr["n←edtarget] is vector ?
                (ptr[n]←ed1 ptr[n])
                "command ← 126)
        (done with ptr)
        ("back←true. done with ptr)
              ) [command] eval.
        )
    .)
    'The heart of ED1 is a vector, containing as its elements
    code vectors. The giant vector is indexed to get the particular
    piece of program, and it is sent the message EVAL. Note that
    the order of the segments in ED1 should match the order of the
    atom names in MENUVEC.'


to edpush ins ("ins←vector 2.
    ins[1]← ptr[n to n+nrun]. ins[1][nrun+1]←nil.
    "ptr←vecmod ptr n nrun ins)

to gettwo t1 n2 ("n←edtarget. "n2←edtarget.
        (n2<n ?("t1←n. "n←n2. "n2←t1))
        "nrun←1+n2-n)

to getcommand code( i(waitfortokenin menu 2*menulength)/2 )

to wait44 ( repeat (button 0 ? (repeat (
                button 7 ?(disp sub "(ev))
                button 4 ?(done)
                button 1 ? (done))
            done)
        )
    )

to waitfortokenin someframe max ind(
    :someframe.
    "max ← (:)+1.
    repeat (wait44.
        0<("ind←someframe mfindt mouse 8 mouse 9) < max ?
            (!ind)
        )
    )

to edtarget targ( "targ←waitfortokenin disp !.
```

```
    disp ← 056. ltarg-1)

done

printing 1.
```

```
printing 0

(to file i bytec status dirty leader curadr nextp backp inused
      numch pagen version sn1 sn2 sadr fname dirinst bitinst
      i dirname bitname DIRECT BTABLE nchock illegal (

      %←? (17 CODE 50)                                    ← write on file string
      %next?    ((%into? (16) %word? (%←? (7) 6) %char. 6) CODE 50)   next read from file
      %eof?     (10 CODE 50)                              eof check end of file
      %flush?   (12 CODE 50)                              flush
      %ovals?   (i(:") oval)
      %set?     (%to. (%eof?(13) %write?(5) %read. 4) CODE 50)   set
      %rewind? (11 CODE 50)                               rewind
      %is?  (%file? () %?? (!"file) :". !false)
      %print?   (disp ← fname)
      %skipnext? (18 CODE 50)                             skipnext
      %knows?   (ov)
      %close?.  (filesopen delete SELF. SELF flush. BTABLE flush. !"closed)   close / delete
      %shorten? (%to. 14 CODE 50)                         shorten
      %delete? (15 CODE 50. filesopen delete SELF. !"deleted)
      %clear?   ()
      %load?    (filesopen map "(vec[i] flush). 8 CODE 50)
      %save?    (filesopen delete SELF. filesopen map "(vec[i] flush).
                9 CODE 50. filesopen map "(vec[i] set 0 0).
                DRIBBLE set to eof. do 20 (DRIBBLE ←052))
      isnew?    (("fname←nchock :)? ((fname=dirname? ("dirinst←SELF)
            "dirinst←DIRECT. "bitinst←(fname=bitname? (SELF) BTABLE))
            "sadr←(%at? (:) string 512)
            (%new?(3) %old?(2) 1) CODE 50. filesopen←SELF. !SELF)
            error "(bad file name))  ))
file knows
("dirname←string 7) fill
SYSDIR.
("bitname←string 9) fill
SYS.STAT.
to nchock i str (
      !str is string? (0<str length<255?
      (for i to str length (str[i]<33? (!false)
      0<illegal[1 to 11] find str[i]? (!false))
      str[str length]=46? (!str) !str+". chars)!false) !false)     ← add
("illegal←string 11) fill                                  } not allowed in a file name
[]().,;<>*#"
done

to DIR ch (disp clear.                                              DIR all
      "ch ← (%all?("(DIRECT rewind)) "(DIRECT set to 4 262))
      file ovals (ch oval. repeat (DIRECT eof? (done)
      0=1024 &* "ch←DIRECT next word? (ch<2? () DIRECT skipnext 2*ch-1)
      DIRECT skipnext 10. "ch←DIRECT next &+ 1.
      disp ← DIRECT next into string ch. sp)) )
to load (file (:) old load)
to save  (file (:) save)
to type f t ("f←file (:) old? (filesopen delete f. "t←string 30.
            repeat(f eof?(done) disp←f next into t)))
to close dname (:dname. filesopen map
      "(vec[end+1-i] ovals (fname)=dname?(vec[end+1-i] close. done)))

printing 1
```

printing 0.

'BOOTSTRAPPING REVISITED'

```
to show fn a b i j k flags clsv clsm arocv arocm instv instm code (
      :#fn. null GET fn "DO?("(no code)) disp clear."to print sp
      "a←leech(#fn.)"b←vector 1. "b←leech b.  "clsm←"arocm←"instm←0.
      "clsv←vector 30 "arocv←vector 30. "instv←vector 30.
      for i=4 to 4+2*a[1]& by 2

      'Pull symbols out of class table'

            ("k←a[i]&.
            k=-1?(again).  "flags ← k&/-14. '0=class, 2=aroc, 3=inst'
            flags=0?(a[i]="DO?("code←a[i+1])
                  a[i]="TITLE?(a[i+1] print sp) a[i]="SIZE?()
                  clsv["clsm←clsm+1] ← a[i])
            b[2]& ← k&*03777.  "j←a[i+1]&.
            (flags=2?(arocv[j-6] ← b[2]. arocm<j-6?("arocm←j-6))
                  instv[j+1] ← b[2]. instm<j+1?("instm←j+1))
            )

      'Now make up input form.'

      for i to arocm (arocv[i] print. sp)
      (instm>0?("; print. sp. for i to instm (instv[i] print. sp)))
      (clsm>0?((instm=0?("; print)). "; print. sp.
            for i to clsm (clsv[i] print. sp)))
      pshow code 3. disp←10.)

'Keyboard fix to translate and dribble'

to kbd i ; translation (%knows?(ov))
kbd knows
      "translation ← string 0377.
      for i=001 to 0177(translation[i]←translation[0200+i] ← i)
      translation[0200]←translation[0233]← 040. 'ctl null and esc'
      to t i (translation[;i]←translation[0200+i]←;)
      t 0020 0010     'SHIFT BS'
      t 0021 0011     'SHIFT TAB'
      t 0037 0040     'SHIFT UP'
      t 0036 0040     'UP'
      t 0035 0040     'SHIFT DOWN'
      t 0034 0040     'DOWN'
      t 0030 0040     'SHIFT SPACE'
      t 0025 0015     'SHIFT RETURN'
      t 0027 0010     'SHIFT DEL->BS'
      t 0177 0010     'DEL->BS'
      t 0032 0040     'SHIFT INS'
      t 0031 0040     'INS'
      t 0022 0012     'SHIFT LF'
      "t←0
done

PUT kbd "DO "(DRIBBLE ← translation[TTY])
to DRIBBLE (%←?(;)%flush?()%set?(%to. %eof))

to fllout disp flist i t ((:disp is file? ()
      "disp←file disp? () error "(file error))
      dsoff (%add?(disp set to eof))
      (null iflist?(defs map "("t←vec[i] eval. show t. cr))
            for i to flist length-1 ("t←flist[i] eval. show t. cr))
      disp evals (disp shorten pagen bytoc). disp close. dson.) )

      'Fllout basically does a show in an environment
      where the display is replaced by a file.'
```

```
to fillin f reader t t  fseq bridgit ckend (%knows?(ov)
      (if is file? () "f←file f old? () error "(file does not exist))
      "reader←fseq f evals (sadr).
      repeat (dsoff read eval print. dson sp reader skip). f close dson.)
fillin knows
to fseq t t str ptr stop bridge str2 eof end (
      %scan?(ptr>stop?((bridge?("ptr←ptr-stop. "stop←end-75) bridgit)
                  "bridge←bridge is false. "t←str. "str←str2. "str2←t.
                  !SELF scan)
            CODE 41?(1="eof←eof+1?(!"done) eof=2?(!rpar) error "(file end)))
      %skip?("ptr←ptr+1)
      !new?(:str. "bridge←false. "end←512. "str2←string 150.
            "stop←end-75. "ptr←"eof←0. ckend))
to bridgit (str2[1 to end]←str[ptr+1 to end].
      "stop←end-ptr. f evals (nextp=0?(sadr[1]←0) f set to  -1 512. ckend)
      str2[stop+1 to end]←str[1 to end]. "ptr←0)
to ckend (f evals   'puts a null at eof'
      (numch<512?(sadr[numch+1]←0)) )
done
```

'Fillin basically does a read-eval-print loop, but its context causes read to use a file buffer ("reader←fseq . . .) instead of the usual keyboard buffer in a dispframe.'

```
to t (dsoff. fild evals (BTABLE is file?() "filesopen←obset 10.
            "DIRECT←file dirname old. "BTABLE←file bitname old.
            DIRECT evals ("bitinst←BTABLE))
      (fname=".?() "DRIBBLE←file fname chars.
      DRIBBLE set to eof. do 20 (DRIBBLE ← 052))
      "disp←dispframe 16 480 511 168 string 520. disp clear.
      "fool←#read. to read (!disp read)
      "defs←obset 100
      "fool←#to. to to x (CODE 19 defs←x. x)
      version print. dson)
```

't is called by start to set up a directory, a dribble fill (if not "."), a display frame, and defs, the list of list of newly defined functions. It then self-destructs to save space.'

```
to start fname (:"fname. t. "t←0. "Hello)
to read reader (disp←62. ("reader←sequence string 50 0) fill. !read)
'Nova file maintenance -- DP0:READER.AN  DP1:READER.AN '
printing 1. "printing←0.
PUT USER "DO "(cr read eval print.). 'Ignore error, type STOP <lf>...'
```

```
printing 0
```

'FLOATING POINT"

```
addto number "(%;?("y←one.
      repeat (%o?("y←y*ten) done) "x←float i.
      repeat (x<one?(done) "x←x/ten)
      !(float SELF)+x/y) )
PUT float "DO "(0 CODE 42
      %ipart?(1 CODE 42)
      %fpart?(2 CODE 42)
      %print?(SELF=zer?(disp←060)
            SELF<zer?(disp←055. Fprint zer-SELF)
            Fprint SELF)
      %e?(ix>0?(!SELF*Fprint evals
                  (for i to 6 (x<v3[i]?()!v2[i] e x-v3[i])))
            x<0?(!SELF/one e 0-x))
      )
"zer←float 0. "one←float 1. "ten←float 10.
to Fprint n i p q s ii fuz1 fuz2 v1 v2 v3 (%evals?((i")eval)
      in. "p←0. "n←n*fuz1. "s←1.
      repeat (for i=s to 6  'normalize to (1,10)'
            (n<ten?(n<v1[i]?("n←n*v2[i]. "p←p-v3[i]. done))
                  n<v2[i]?() "n←n/v2[i]. "p←p+v3[i]. done)
            i=7?(done) "s←i)
      ("q←p. "s←n*fuz2. p>8?("p←0) p<-3?("p←0) "q←0. "s←s e p
            p<0?(disp←073. "s←s e -p. do -p+1(disp←060)))
      for i to 9 (disp ← 060 + n ipart.
            "p←p-1. "n←ten*n fpart.
            p<0?(n<"s←ten*s?(done) p=-1?(disp←073)))
      q=0?() "e print. q print)
Fprint evals (ev)
"v1←vector 6. "v2←vector 6. "v3←vector 6.
v1[6]←one. v2[6]←ten. v3[6]←1.
for i=5 to 1 by -1 (v1[i]←ten/v2[i]←v2[i+1]*v2[i+1]
      v3[i]←2*v3[i+1]).
"fuz1←one+1;0e-9. "fuz2←8;0e-9.
done

'Nova file maintenance --- DP0:FLOATING.AN  DP1:FLOATING.AN '
printing 1
```

These 2 pages for Smalltalk repairmanic


```
INIT            MK     000013   .ARET 000155   QSTAR 001250   GBYT   003744
PAGE0           LL     000014   .INTN 000156   DWITE 001250   PBYT   003753
READ            CX     000015   U READ 000     QCLAS 001261   POCT   003767
EVAL            CY     000016   .READ 000157   QCOLN 001350   MSAVE  004000
SSCAN           SYSAV  000016   .RATM 000160   QMESS 001432   FREE   004004
CODE2           RP     000017   U RATOM 000    TRACE 001525   TOPLE  004010
UTIL            DWIDE  000040   .QWRI 000161   EVAL  001562   SINTB  004012
FUNCS           .LOAD  000070   .LMPC 000162   ARET  001577   STPRT  004015
COMM            .STOR  000071   .AMPC 000163   ERET  001637   SPRT   004020
CODE            TLMEM  000072   .APC  000164   EFIND 001654   ERR    004077
MEM             BHMEM  000073   .FIND 000165   QSLAS 001674   CRLF   004130
RCHAR           SELF   000074   .CACT 000166   QUEST 001774   STERR  004136
TURT            VALUE  000075   .LALO 000167   MESSX 001775   QERR2  004144
ALTO            DX     000076   .CRLF 000170   GLOBX 001776   QERR   004145
OPEN            RADIX  000077   .SPRT 000171   QSIZE 002104   QWRIT  004230
GTINS           ATTN   000100   .EVAL 000172   FETCH 002153   QERR1  004242
MOD16           MSMSK  000102   .FECH 000173   EVTK  002156   INDX   004253
CKPAR           TOKEN  000103   .PEEK 000174   QLEN  002157   DISP   004273
SETUP           EMPTY  000104   .LARG 000175   QAROW 002174   DOMUL  004311
MAZUR           SCLAS  000105   .ERET 000176   QAND  002203   DODIV  004322
LNOUT           ATCLS  000106   .INDX 000177   QAREC 002257   LINST  004340
ALENT           LSCLA  000107   .DISP 000200   QMOD  002264   LNST   004343
WCHAR           NCLAS  000110   .SARG 000201   BOUND 002310   SINST  004354
PSTRG           MASTE  000111   .SGET 000202   GNST  002342   INTN   004400
BCLMV           HOLD0  000112   .SPUT 000203   GINS  002344   ISIT   004440
CLREM           HOLD1  000113   .MUL  000204   GARG  002354   GET    004500
SETXY           .QUES  000114   .DIV  000205   QLESS 002360   GADDR  004576
SCAN            .QSIZ  000115   .LAL4 000206   SSCOD 002432   PUT    004577
UPDAT           .QLAR  000116   .REFI 000207   QRETN 002455   FIND   004630
NEWLN           .QPER  000117   .REFD 000210   QOR   002457   SGET   004700
INSEC           .QDO   000120   .SVAL 000211   SSCAN 002476   SPUT   004742
CKSTF           HOLD2  000121   .SVLI 000212   QRPAR 002644   CACT   004774
DOMOD           HOLD3  000122   .SSLF 000213   QCODE 002662   MXNUM  005013
CLRWF           HOLD4  000123   .LINS 000214   SMF   002702   SRETN  005041
DSPCH           HOLD5  000124   .SINS 000215   SMS   002731   SVAL   005113
SCROL           C2     000125   .ISIT 000216   PPW   002753   SVLI   005126
BKSPC           C3     000126   .FALS 000217   PNW   002755   IVAL   005137
SMFCO           C4     000127   .SSCA 000220   PTW   002756   U FTR  005
SMLOO           C5     000130   .SMF  000221   GPW   002777   U FLD  005
SMALL           RBMSK  000131   .GNB  000222   GNW   003001   REFD   005203
SMDIO           LBMSK  000132   .PNB  000223   GTW   003002   REFI   005233
SMFIL           RCMSK  000133   .GARG 000224   GPB   003014   LALO   005545
LAST            MXATM  000134   QLPAR 000240   GNB   003016   FPCLA  005610
NOVA            .IVAL  000135   QCHAR 000270   GTB   003017   LAL4   005614
WBOOT           .SRET  000136   QGLOB 000350   PPB   003037   CODES  005705
BOOT            .PUT   000137   QTITL 000513   PNB   003041   U FILIN 005
NMAX 061424     .GET   000140   QDO   000540   PTB   003042   U FPCOD 005
ZMAX 000236     .NEXT  000141   QMACH 000624   STRVE 003214   U SPLOT 005
CSZE            .GBYT  000142   QHYPH 000664   QMPER 003230   PRIN1  006002
EST             .PBYT  000143   QEQ   000764   QPER  003270   READ1  006006
SST             .PICK  000144   QEVAL 000774   QGRTR 003370   RPT1   006011
                .STIK  000145   PSTAD 001000   KYBD  003507   AGAIN  006017
DBITS           .ERR   000146   START 001000   APLY1 003531   DONE1  006021
DTABE           .QERR  000147   DWIDB 001000   FOR1  003555   NUM1   006065
FX     000005   .POCT  000150   STOP  001004   .FORX 003606   RFALS  006304
FW     000006   .MEMC  000151   QSELF 001011   QNOEV 003614   USERL  006345
FY     000007   .MEMO  000152   MKTOK 001013   LEECH 003653   USRX   006362
FH     000010   .TPUT  000153   QTO   001100   QPLUS 003654   QUOT1  006374
BF     000011   QLBRA  000154   QRBRA 001164   PICK  003730   ATOM1  006377
```

PUT1	006456	OPEN	012310	OUTX	013030	MSEY	020532	BOOT	061026		
EMPT1	006472	SVALL	012341	OUTWD	013031	MCHR	020533	PHIAD	062151		
MEM1	006550	SVACS	012342	OUTY	013032	MTOK	020534	MEND	130654		
GET1	006577	RSALL	012345	OUTHT	013033	MCHRS	020535	SINT1	177600		
FET1	006614	RSACS	012346	.LGWD	013036	MTOKS	020536	U	PDEC	177	
MAT1	006700	WIDTH	012351	.LSCN	013045	MSESW	020537	777			
TO1	006717	GETHT	012413	CURX	013062	GOTIT	020540				
NULL	007112	GETWD	012415	LNOUT	013064	GOTMS	020577				
APRET	007116	GTMES	012425	FHTGH	013206	STSCL	020700				
ISNEW	007131	.BF	012440	FWDTH	013207	SCROL	020706				
MKINS	007162	INSTB	012441	FSTRT	013210	BKSPC	021031				
LCL	007212	INSTC	012441	PCHAR	016417	FCODE	021057				
LPC	007215	WNTBL	012442	MAZX	016550	EOF	021333				
LMO	007220	WINX	012442	PSTRG	016554	DELET	021352				
LME	007223	WNWD	012443	CLEAR	016644	LOOKU	021402				
LGL	007226	WINY	012444	STCLR	016734	ELOOK	021577				
LRE	007231	WNHT	012445	CLRX	016736	ENTER	021605				
LIN	007234	FRTBL	012446	CLRWD	016737	DSKAD	022027				
LAO	007237	FRMX	012446	CLRY	016740	DSKSE	022062				
SCL	007242	FRWD	012447	CLRHT	016741	ALLOC	022104				
SPC	007245	FRMY	012450	INDEX	016754	SHORT	022142				
SMO	007250	FRHT	012451	STBCL	017004	DALLO	022166				
SME	007253	BUF	012452	STBLT	017005	LODM	022253				
SGL	007256	BFLN	012453	BMVCL	017031	SAVM	022256				
SRE	007261	LAST	012454	BLKMV	017032	SREW	022327				
STIN	007264	MARK	012455	CLREM	017223	SREOF	022330				
SAO	007267	LSLN	012456	STSXY	017346	SCLOS	022334				
LARG	007272	CHRX	012457	SETX	017350	SRBYT	022344				
SARG	007276	CHRY	012460	SETY	017352	SWBYT	022352				
SSELF	007312	RPLY	012461	SETXY	017361	CRR	022653				
AMPC	007320	WNBT	012462	CHR	017405	CRW	022654				
APC	007325	FRBT	012463	CONTB	017412	CCR	022655				
LMPC	007341	FRRT	012464	CTBO	017412	CCW	022656				
NEXT	007347	GTINS	012466	CTB1	017413	CWW	022657				
PEEK	007361	MOD16	012555	CTB2	017414	DSKB	022663				
LOADE	007406	RELTB	012574	SCAN	017415	DSKIO	022664				
STORE	007413	REALX	012574	HITSW	017452	DSKBU	023053				
MEMO	007425	RELWD	012575	WNOVR	017453	RICOP	023453				
MEMC	007447	REALY	012576	UPDAT	017460	IRCOP	023477				
VM2CR	007504	RELHT	012577	NEWLN	017545	RVCOP	023506				
CR2VM	007521	CKPAR	012601	INSEC	017610	VRCOP	023515				
ST2CO	007563	MZTBL	012662	LSTSP	017657	RFPTR	023531				
GCHAR	007612	MLBD	012663	LSTC	017660	SNST	023553				
RIGHT	010446	MRBD	012664	QFLG	017701	GFCH	023575				
GO	010600	LNTBL	012666	QPT1	017702	GFWD	023634				
DPLOT	010750	LLBD	012667	QPT2	017703	GFSTR	023644				
INK	011025	LRBD	012670	CRCHK	017705	PFCH	023672				
TTLC	011033	INTBL	012671	DOSPC	017757	PFWD	024017				
TSTAT	011051	INX	012671	CKSPC	017776	PFSTR	024026				
GOTO	011256	INWD	012672	MDDIM	020064	PATCH	024077				
ALMO	011512	INY	012673	MDINC	020070	MEMSI	024137				
XYBIT	011652	INHT	012674	MDFIX	020105	XYC1	024140				
KBCK	011674	OPNSW	012677	PCLER	020142	XYCO	024145				
KBINT	011716	SETUP	012701	WCLER	020143	MEMBI	024151				
ALTG	011733	.MGWD	012762	FCLER	020155	DSIZE	052400				
MOUSE	012045	.CRCK	012763	DSPCH	020176	TTGET	056151				
SIO1	012226	.DOSP	012764	BFCON	020204	TTPUT	056156				
BTMAP	012254	.CKSP	012765	MOOD	020357	PRING	056206				
DSPTB	012255	MAZUR	012773	STRMD	020440	PRTON	056213				