to class x y ()
to number x y :: nprint ()
to vector x y :: substr ()
to atom x y (CODE 29)
to string x y :: substr ()
to arec x y ()
to float x y :: fprint ()
to falseclass x y (isnew)
to isnew (CODE 5)
false+falseclass.
TITLE USER DO SIZE CODE SELF AREC GLOB MESS RETN CLAS
length eval or and mod chars error
"../:-[]?*S#()<>{}[]*<> go goto turn next contents end [A+C+K+N+Z]

'"h75.+f1.
DONT EDIT ABOVE HERE--These classes and atoms mentioned early to guarantee a
**ddresses for machine code.

to isnew (null instance=(instance-allocate permsize.
instance[0]+class. true)
false).

"h90.+f2.
BOOTSTRAPPING MAGIC.

"h75.+f1.
HEREWITH A SOMEWHAT WHIMSICAL ANNOTATED VERSION OF SYSDEFS. ANNOTATIONS A
**N ITALICS. WHILE IT IS HOPE THAT T
**HIS WILL PROVIDE SOME ELUCIDATION OF
**THE CODE ESCAPES, OBSCURITIES WILL NO
**DOUBT PERSIST. THE ANNOTATIONS ARE
**INTENDED TO BE BUT DIMLY LIGHTED MAR
**KERS ON THE ROAD TO TRUE ILLUMINATION
**.

"h60.+f0.'
to print (•
'"h75.+f1.
: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.
"h60.+f0.'
to read (CODE 2)
  'r75.+f1.
  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.
  +h60.+f0.'

'++90.+f2.
MESSAGE HANDLING
  +h60.+f0.'

to : (CODE 18)
  'r75.+f1.
  to : name

  \1\37\(\(\{\sname\ snil \Rightarrow (↑\sname\ +\ scram\ message\ quotefetch)
    (↑\scaller\ message\ quotefetch)

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

  \1\37\# \Rightarrow (\{\sname\ snil \Rightarrow (↑\sname\ +\ scram\ message\ referencefetch)
    (↑\scaller\ message\ referencefetch)

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

  (\{\sname\ snil \Rightarrow (↑\sname\ +\ scram\ message\ evalfetch)
    (↑\scaller\ message\ evalfetch)

  Fetch the next thing in the message stream evaluated
  and bind it to the name if one is there.
  +h60.+f0.'

to \ (CODE 17)
  'r75.+f1.
  \{\stoken\ . \stoken\ =\ scram\ . \stoken\ . \message\ . \code\ [\caller\ . \message\ . \pc\] \Rightarrow
    (caller\ . \message\ . \pc\ +\ scram\ . \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.
  +h60.+f0.'

to +K ( CODE 39 )

to +C ( CODE 36 )
to \(\uparrow\) (CODE 13)

'\+h75.\+f1.
\(\downarrow\)x. then do a return, and apply x to any further message. Note
that in (... \(\uparrow\)x+3. \(\uparrow\)y+y-2), the assignment to y will never
happen, since \(\uparrow\) causes a return.
\+h60.\+f0.'

to \(\uparrow\) (CODE 9)

'\+h75.\+f1.
\(\uparrow\):\(\uparrow\). That is, get the next thing in the message stream unvealed
and active return it (which causes it to be applied to the messag
**e).
\+h60.\+f0.'

to \# (:#)

'\+h75.\+f1.
Returns a REFERENCE to its argument's binding.
\+h60.\+f0.'

'\+h90.\+f2.
CONTROL CLASSES
\+h60.\+f0.'

to repeat token (:#token. CODE 1)

'\+h75.\+f1.
repeat (token eval) Not a true apply to eval, and therefore token
MUST be a vector.
\+h60.\+f0.'

to done x (.inspect:(x. CODE 25) CODE 25)

'\+h75.\+f1.
done causes a pop out of the nearest enclosing repeat, for, or do.
\+Odone with val\+O will cause the repeat to have value val
\+h60.\+f0.'

to again (CODE 6)

'\+h75.\+f1.
repeat (\(\uparrow\)active+active caller. eq active. class #repeat\(\downarrow\)done)).
That is, redo the most recent repeat, for, or do loop.
\+h60.\+f0.'

to if exp (:exp\(\rightarrow\)true\(\rightarrow\)false)\(\rightarrow\)error \(\rightarrow\)false)

'\+h75.\+f1.
The ALGOL \+Oif ... then ... else ... \+O
\+h60.\+f0.'
to for token step stop var start exp (  
  #:var. ( #=>;(start.)#:start+1).  
  ( #=>;(stop.)#:stop-start.)  
  ( #=>;(step.)#:step+1.)  
  do #:exp. CODE 24)  
  'h75.+f1.
An Algol-like +Ofor+O.  
Note the default values if +O+O, +Oto+O, +Oby+O, etc., are omitted.
CODE 24 means --repeat(exp eval).  This implies +Odone+O and +Oa
  **gain+O
will work, which is correct.
  'h60.+f0.'

to do token step stop var start exp (  
  #:step#:start+1. #:stop #:exp. CODE 24)

  'h90.+f2.
INITIALIZING SYSTEM CLASSES

  'h75.+f1.
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 version of SMALLTALK will have class as a class
with these actions intensional.
  'h60.+f0.'
to PUT x y z (::x. ::y. ::z. CODE 12)
  'h75.+f1.
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 tab
  **le
  **le
corresponding to the name (atom) which was the second argument.
  'h60.+f0.'
to GET x y (::x. ::y. CODE 28)
  'h75.+f1.
If +Ox+O is a class table then the binding of the atom in +Oy+O will
  **be
  **be
fetched.
  'h60.+f0.'
to leech field bits : ptr
  isnew := (ptr)
  CODE 27)
  'h75.+f1.
  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.
  'h60.+f0.'

PUT USER TITLE USER
PUT falseclass TITLE false

PUT atom DO (CODE 29
  'h75.+f1.
  (x. "x -- Lookup SELF and replace its value by x."
  eval (↑ -- Lookup the binding of SELF)
  = (↑SELF=:
  chars (↑ -- printname of SELF (a string))
  'h60.+f0.'

  (ISIT eval)
  print (disp SELF chars)
  )
  'h75.+f1.
  Done this way (PUT used rather than using ↑Oto↑O) because we
  wanted to know where the system classes are. Hence the initial
  ↑Oto atom x y (↑O , for example, in ↑OBootstrapping Magic↑O followed
  by the behavior here.
  'h60.+f0.'

to ev (repeat (or read eval print))

PUT falseclass DO (CODE 11
  'h60.+f1.
  (G...
  or (↑:)
  and (:.)
  (< (:.)
  => (:.)
  => (:.)
  'h60.+f0.'

  (false (↑true) ? (↑false :G:)
  print (false print))
PUT vector \texttt{GPDO G (CODE 3 \Rightarrow (\texttt{\#substr SELF x GLOB MESS})
\texttt{'}\texttt{\#h75.\#f1.}
\texttt{isnew\Rightarrow(Allocate vector of length \texttt{\#}.}
\texttt{Fill vector with nils.)}
\texttt{[\Rightarrow:(x. \texttt{\#}).}
\texttt{(\Rightarrow\Rightarrow:(y. \texttt{\#y -- store y into xth element.})}
\texttt{\texttt{\#xth element})})
\texttt{length\Rightarrow(\# length of string or vector)
\texttt{eval\Rightarrow(G^{pc<0. repeat}
\texttt{(null SELF[G^{pc+pc+1}]\Rightarrow(done)
\texttt{G^{eval=SELF[pc] eval})
\texttt{\#eval)} sort of...)
\texttt{\#h60.\#f0.').
\texttt{is\Rightarrow(ISIT eval)
\texttt{print\Rightarrow(disp=40. for x to SELF length}
\texttt{(disp-32. SELF[x] print). disp=41)
\texttt{map\Rightarrow:(y. for x to SELF length}
\texttt{(evapply SELF[x] to y))})

PUT string \texttt{GPDO G (CODE 3 \Rightarrow (\texttt{\#substr SELF x GLOB MESS})
\texttt{'}\texttt{\#h75.\#f1.}
\texttt{isnew\Rightarrow(Allocate string of length \#.)}
\texttt{Fill string with 0377s.)}
\texttt{[\Rightarrow:(x. \texttt{\#}).}
\texttt{(\Rightarrow\Rightarrow:(y. \texttt{\#y -- store y into xth element.})}
\texttt{\texttt{\#xth element})})
\texttt{length\Rightarrow(\# length of string or vector)+\#h60.\#f0.}

\texttt{is\Rightarrow(ISIT eval)
\texttt{print\Rightarrow(0 = G \texttt{x \leftarrow SELF[1 to 9999] find first 39}
\texttt{(disp=39. disp \leftarrow SELF. disp=39)
\texttt{SELF[1 to x-1] print. SELF[x+1 to SELF length] print)
\texttt{=}\Rightarrow:(y \texttt{is string\Rightarrow(SELF length=y length\Rightarrow(
\texttt{for x to SELF length (SELF[x]=y[x]\Rightarrow(\#false)) \#false)
\texttt{\#false)
\texttt{=}\Rightarrow:(y \texttt{is string\Rightarrow(G^{x\leftarrow SELF[1 to SELF length+y length].}
\texttt{\texttt{\#x[SELF length+1 to x length]-y[1 to y length]})
\texttt{error G^{(string not found))})}
### PUT number © DO © (CODE 4

' †h75. †f1.

© += (© val +:)

© -= (© val -:)

© *= (© val *:)

© /= (© val /:)

© < (© val <:)

© >= (© val =:)

© > (© val >:)

© [[ (© += (© val OR :)

© = (© val XOR :)

© * (© val AND :)

© / (© val LSHIFT :))

' †h60. †f0.'

© is (ISIT eval)

© print (SELF > 0 (nprint SELF)

SELF = 0 (disp + 060)

SELF = 0100000 (disp + base8 SELF)

disp = 025. nprint 0 - SELF )

' †h75. †f1.

For floating point stuff see FLOAT

' †h60. †f0.'

to - x ( : x * †U1)

' †h75. †f1.

An often used abbreviation, has to work for float as well.

' †h60. †f0.'

to base8 i x s ( : x. © s = string 7. for i to 7

(s[8 - i]) = 060 + x © 7. © x ← x © †U3. © s)

' †h75. †f1.

Returns a string containing the octal representation (unsigned) of it

**s

integer argument.

' †h60. †f0.'

© ISIT ← © © (© ? (© TITLE) †TITLE = : ©)

to nil x ( †#x)

' †h75. †f1.

nil is an †Ounbound pointer †O, which is used to fill vectors and tables.

' †h60. †f0.'
to null x (:x. 1 CODE 37)
  'h75.+f1.
  Null returns true if its message is  ④Onil①O, otherwise false.
  'h60.+f0.'

to eq x (CODE 15)
  'h75.+f1.
  (①x is-identical-to :) - compare 2 SMALLTALK pointers.
  'h60.+f0.'

'rh90.+f2.
UTILITIES
'h60.+f0.'

to mem x y (:x. CODE 26)
  'h75.+f1.
  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 0105 ← 0. --disconnect cursor from mouse
  mem 0426 ← x. mem 0427 ← y. --move the cursor
  mem 0107 ← 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
  mem 0177030. --reads the word with mouse and keyset bits.
  'h60.+f0.'

to mouse x (:x. CODE 35)
  'h75.+f1.
  x = 8 ? 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.
  'h60.+f0.'

to mx (①mouse 8)
to my (①mouse 9)
to core ("mem 077") mem 076
  "h75. f1.
  Returns the amount of space left in your Smalltalk.
  +h60. f0."

to kbd (0 CODE 20)
  "h75. f1.
  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.
  +h60. f0."

to disp x i (
  (x is string) (for i to x length (TTY x[i])) TTY x)
  clear() sub(x eval))
  "h75. f1.
This disp is used for bootstrapping. Later in these definitions
(READER) it will be restored to an instance of +Odisplay frame. +O
+h60. f0."

to TTY (0 CODE 20)
  "h75. f1.
TTY <integer> will print an ascii on the Nova tty. On altos, TTY
prints in little error window at bottom of screen.
+h60. f0."

to dsoff (mem 272 +0)
  "h75. f1.
Turns display off by storing 0 in display control block ptr. Speeds
up Alto Smalltalk by factor of 2.
+h60. f0."

to dsen (mem 0420 + 072)
  "h75. f1.
Turns display back on by refreshing display control block
pointer.
+h60. f0."

to apply x y (x: #y. x to (y. x in (:GLOB. CODE 10) CODE 10)
  in (:GLOB. CODE 10) CODE 10)
to evapply x y (x. x to (y. x in (:GLOB. CODE 10) CODE 10)
  in (:GLOB. CODE 10) CODE 10)
  "h75. f1.
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 unless the in-clause is used to
specify another context.
+h60. f0."
to cr (disp+13). to sp (disp+32)

true = true
val = eval
to is ( (? damages untitled): (true. ?false)
  'h75. f1.
  These are used to handle messages to classes which can't answer
  question invoking +Ois+O. +Oeval+O, etc.
  'h60. f0.'

to t nprint substr (ev). t
  'h75. f1.
  prevent -to- from making these global.
  'h60. f0.'

to nprint digit n (n=00) (digit + n mod 10. nprint + n/10. disp+060+digit)
PUT number nprint #nprint.
  'h75. f1.
  Prints (non-neg) integers in decimal with leading zeroes suppressed
  **d Q
  'h60. f0.'

to substr op byte s lb ub s2 lb2 ub2 (s: n: #s. :lb. :ub. :MESS. GLOB=ub. 'f1. tee hee= h60. f0.
  :
  ub. ( ) error (missing right bracket)
  byte += ub2 += 1.
  op = (op + (first+(1) last+(2) 1)
  + (non+(2) 0). byte. CODE 40)
  all = (byte. op+0. CODE 40)
  s2 = (lb2. to. ub2. ). CODE 40
  ub2+9999. CODE 40)
  op += 6. ub2 += ub+1-lb.
  s2 = (s is string (string ub2) vector ub2). CODE 40.
PUT string s substr #substr.
PUT vector substr #substr.
done
  'h75. f1.
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
See vecmod for more examples. String syntax is identical.
  'h60. f0.'
to vecmod new end old posn ndel nins ins (end+10000).
   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].
   ↑new
   'rh75.tf1.
   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.
   ↑rh60.tf0.'

to addto func v w (:#func. :w. v=GET func DO. null v=(error (no code))
   PUT func DO vecmod v v length 0 w)
   'rh75.tf1.
   Addto appends code to a class definition.
   ↑rh60.tf0.'

to q t i l str (  
   ↑1 ← :str length.
   t ← disp + kbd.
   (t = 10)
   (t ← disp + kbd)).
   str[↓i ← 1] ← t.
   repeat
   (i = l)(done)
   10 = str[↓i ← i + 1] ← disp ← kbd.(done)).
   ↑str)

to stream in : i s l(  
   CODE 22
   'rh75.tf1.
   CODE 22 is equivalent to...
   ≈^→
   (  
   (i = l)
   (s ← s[1 to l ← 2 * l]))
   ≈s[↓i ← i + 1] ← :)
   ≈next→
   (i = l)($0)
   ≈s[↓i ← i + 1])
   ≈contents→
   (↑s[1 to i])
   ↑rh60.+f0.'
<reset>
(G i ← 0)

isnew
(G s ←
(of :)
string 10).
G i ←
(from (:)
- 1)
0).
G i ←
(to (:)
s length))

is
(ISIT eval)
end
(∀ i = 1)
print
(
(i > 0 ⇒
(s[1 to i] print)).
disp ← 1.
l < i + 1⇒()
s[i + 1 to l] print))

to obset i input : vec size end (  
 add ((size=G vec+end+end+1⇒(G vec←vec[1 to G size+size+10]))
 vec[end]←:)
  ⇒(0=vec[1 to end] find first :input⇒
  (SELF add input))
  delete ((0=G vec[1 to end] find first :input⇒(↑false)
  vec[i to end]+vec[i+1 to end+1]. G end←end-1)
  unadd⇒(G input←vec[end]. vec[end]←nil.
  G end←end-1. ↑input)
  vec←(↑vec[1 to end])
  map((↓input. for i ← end to 1 by ↑U1 (input eval))
  print⇒(SELF map G (vec[i] print. sp))
  is⇒(ISIT eval)
  isnew⇒(G end←0. G vec←vector G size←4)
)
'fh90.+f2.
PRETTY-PRINT
'fh60.+f0.

'fh75.+f1.
This prints the code; classprint makes the header.
'fh60.+f0.'

to show func t (
  #:func. @t~GET func @DO.
  null t => (↑@t~(no code)) pshow t 0.)
to pshow ptr dent i t :: x tabin index (:ptr :dent.
  (ptr length>4→(tabin dent)) disp+40.
  for i to ptr length-1
    [t~t ← ptr[i].
     t is vector ⇒ (pshow t dent+3.
       i=ptr length-1⇒()   
       ➔x ← ptr[i+1]⇒()  
       x is vector⇒()     
       tabin dent)
     i=1 ⇒ (t print)
     0<@x~index @. , ↑S [ ] ⇒ t⇒
      (x=1⇒(t print. ptr[i+1] is vector⇒() tabin dent)
       t print)
     0=index @: ( @ # ↑ [ ] ⇒ [ ] ptr[i-1]⇒(disp+32. t print)
       t print)
    disp+41)
to t tabin index (ev)
t
to tabin n :: x (:n. disp+13. repeat
    (n > 6⇒
     (disp ← x[6].
      ➔n ← n - 6)
    done)
  disp ← x[n + 1])
(PUT tabin @x {string 0 32 q string 2 q string 3
    q string 4 q string 5 q string 6}).
  'leave these blanks'
PUT pshow @tabin #tabin.
to index op byte s lb ub s2 lb2 ub2 (  
  :s. :byte. @op ← @l~b~@s2~@lb2~@ub2~@-1. @ub~+9999. CODE 40)
  'fh75.+f1.
  A piece of substr which runs faster.
  'fh60.+f0.'
PUT pshow @index #index.
done
'+h90.+f2.
FLOATING POINT'+h60.+f0.
'
PUT float $\text{DO}$ (0 CODE 42
  $\text{ipart}$ (1 CODE 42)
  $\text{fpart}$ (2 CODE 42)
  $\text{ipow}$
  ($x = 0$ (↑1.0)
  $x = 1$
  $x > 1$
  (1 = x mod 2
   (↑SELF * (SELF * SELF)
    ipow x / 2)
   ↑(SELF * SELF)
    ipow x / 2)
  ↑1.0 / SELF ipow 0-x)
  $\text{epart}$
  (SELF $<$ x (↑0)
   SELF $<$ x * x (↑1)
   ↑
   ($\downarrow y + 2$ * SELF epart x * x)
   +
   (SELF / x ipow y)
   epart x)
  $\text{is}$ (ISIT $\text{eval}$
  $\text{print}$
  (SELF $=$ 0.0 (disp + 48. disp + 46. disp + 48)
   SELF $<$ 0.0
   (disp + 025.
    fprint + SELF)
   fprint SELF)
  )
  to t fprint (ev)
  to fprint n i p q s : : fuzz ( 'f1.Normalize to [1,10]+f0.'
  : n $<$ 1
   ($\downarrow p + (-10.0 / n)$
     epart 10.0)
   $\downarrow p + n$ epart 10.0)
   $\downarrow n$ + fuzz + n / 10.0 ipow p.
  'f1.Scientific or decimal+f0.'
   ($\downarrow q + p$.
    $\downarrow s + fuzz^2$
   p $>$ 6
   ($\downarrow p + 0$)
p < +U3>
(p ← 0)
q ← 0.
p < 0⇒
(disp ← 48. disp ← 46.
   for i ← p to +U2(disp ← 48))
   s ← s * 10.0 ipow p
'↑f1.Now print (s suppresses trailing zeros)↑f0.'
for i to 9
(disp ← 48 + n ipart.
   p ← p - 1.
   n ← 10.0 * n fpair.
   p < 0⇒
   (p = +U1⇒(disp ← 46))
   n < s ← 10.0 * s⇒(done)))
(p = +U1⇒(disp ← 48))
q = 0⇒()
disp ← 0145.
q print)
PUT fprint = fuzz 5.0 * 10.0 ipow +U9.
PUT float fprint #fprint.
done

'↑h90.↑f2.
TEXT DISPLAY ROUTINES
↑h60.↑f0.

↑h75.↑f1.
Display frames are declared with five parameters. They are a left x, a width,
  ** a top y, a height, and a string. He
  **nce --
     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.

There are actually two entities associated with display frames--frames and wi
  **ndows. Currently both are given the s
  **ame dimensions upon declaration (see
  **isnew).
The four instance variables defining the window are +Owind+O, +Owinwd+O, +Owi
**ny+O, and +Owinht+O. The boundaries
**of this rectangle are intersected wit
**h the physical display. The window a
**ctually used by the machine language
**will reduce the size of the window, i
*** necessary, to be confined by the ph
**ysical display. Clipping and scrolli
**ng are done on the basis of window bo
**undaries. If a character is in the w
**indow it will be displayed. If a str
**ing or character cause overflow of th
**e bottom of the window, scrolling wil
**l occur.

The four instance variables defining the frame are +Ofrmx+O, +Ofrmwd+O, +Ofrm
**y+O, and +Ofrmht+O. This rectangle
**may be smaller or larger than its ass
**ociated window as well as the physica
**l display. Frame boundaries are the
** basis for word-wraparound. (Present
**ly, if frm+y+ frmht will cause overflo
**w of the window bottom[winx+winht], f
**rmht will get changed to a height con
**sonant with the bottom of the window.
** This has been done to manage scroll
**ing, but may get changed as we get a
**better handle on the meaning of frame
**s and windows.).

+OBuf+O is the string buffer associated with any given instance of dispframe.
** This is the string that is picked o
**n the way to microcode scan conversio
**n. When scrolling occurs, the first
**line of characters, according to fram
**e boundaries, is stripped out and the
** remainder of the buffer mapped back
**into itself. If a +O→O message woul
**d overflow this buffer, then scrollin
**g will occur until the input fits.

+OLast+O is a +Obuf+O subscript, pointing to the current last character in th
**e buffer. That is, the last characte
**r resulting from a +O←+O.

+OLstln+O also points into the buffer at the character that begins the last l
**ine of text in the frame. It is a st
**arting point for scan conversion in t
**he +O←+O call.
+OMark+O is set by dread (see below) and points to the character in the buffer which represents the last prompt of output by SMALLTALK; reading begins there. Mark is updated by scrolling, so that it tracks the characters. One could detect scrolling by watching mark.

+OCharx+O and +Ochary+O reflect right x and top y of the character pointed to by +Olast+O.

The +Oreply+O variable in the instance may be helpful in controlling things. When the reply is 0, it means every thing should be OK. That is, there was no intersection between the window 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 3 reply means window height less than font height -- hence no room for scan conversion of even one line of text. A 4 means that the frame height has been increased in order to accomodate the input. A 5 means the bottom of the window (i.e. window x + window height) has been overflown -- hence that scrolling took place. A 6 means that both 4 and 5 are true.

+Ojustify+O is a toggle for right justifying the contents of a dispframe. The default is 0 and means no justification. Setting it to 1 causes justification on frame boundaries.

The +Ofont+O variable allows for the association of a font other than the default font with the display frame. To get a different font into core say

something ← file <fontfilename> intos string. Then you can say dispS (font something) or you can declare the font at the same time as the dispframe is declared as e.g.

yourframe ← dispframe 3 40 3 40 string 20 font something.

+th60.+f0.'
(to dispframe input
   : winx winwd winy winht frmx frmwd frmy frmht
   last mark lstln charx chary reply justify buf font editor
   : sub frame dread reread (}

  ↯ ↯(0 CODE 51)
  'h75.+f1.
  :s. s is number ⇒ (append this ascii char)
      s is string ⇒(append string)
      error.
  +h60.+f0.'

  ↯ :S⇒(↑(:S)↑eval)
  'h75.+f1.
  Allows access to instance variables. For example,
  yourframe↑S (↑frmwsumx<32)
  will alter the value of window x in the instance of dispframe
  called yourframe↑S.
  +h60.+f0.'

  ↯ show⇒(4 CODE 51 3 CODE 51)

  ↯ display⇒(SELF show. frame black)
  'h75.+f1.
  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.
  +h60.+f0.'

  ↯ hasmouse⇒(frmx<frmx+frmwd⇒(↑frmy<frmy+frmht)↑false)
  'h75.+f1.
  Tells you if the mouse is within a frame.
  +h60.+f0.'

  ↯ fclear⇒(4 CODE 51)
  'h75.+f1.
  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+O on that frame.
  +h60.+f0.'

      G↑last-0. G↑lstln-1. SELF+input. ↑charx-winx)
  'h75.+f1.
  For them as would rather do it themselves.
  +h60.+f0.'
wclear := 5 CODE 51
    '+h75.+f1.
    Wclear clears the intersection of a window and the physical
display.
    '+h60.+f0.'

scroll := 2 CODE 51
    '+h75.+f1.
    Scroll removes the top line of text from the frame's string buffer,
and moves the text up one line.
    '+h60.+f0.'

clear := 1 CODE 51
    '+h75.+f1.
    Clear does an fclear and sets the +Olast+ pointer into the string bu
**uffer
to 0 and +OlstIn+ to 1. It has the effect of cleaning out the string
buffer as well as clearing the frame area.
    '+h60.+f0.'

mfindc := 7 CODE 51
    '+h75.+f1.
    Find character.
    Takes two arguments -- x and y (typically msex and msey).
    Returns vector:
    vec[1] = subscript of char in string
    vec[2] = left x of char
    vec[3] = width of char
    vec[4] = topy of char
    If vec[1] is -1 x,y is after the end of the string.
    If vec[2] is -2 x,y is not in the window.
    Sample call:
    myvec+yourframe mfindc mouse 8 mouse 9.
    '+h60.+f0.'

mfindw := 8 CODE 51
    '+h75.+f1.
    Find word.
    Takes two arguments -- x and y (typically msex and msey).
    Returns vector:
    vec[1] = subscript of first char in word
    vec[2] = left x of word
    vec[3] = width of word
    vec[4] = topy of word
    If vec[1] is -1 x,y is after the end of the string.
    If vec[2] is -2 x,y is not in the window.
    Sample call:
    myvec+yourframe mfindw mouse 8 mouse 9.
    '+h60.+f0.'
<mfind>(6 CODE 51)
  'th75.tf1.
Find token.
 Takes two arguments -- x and y (typically msex and msey).
 Returns vector:
    vec[1] = token count, ala Smalltalk token Spaces and
carriage returns are considered as delimiters, but
multiple delimiters do not bump the count. Text
delimited by single quotes is counted as one
token, and embedded text (i.e. more than one
quote in sequence will not cause the token count
to be bumped (allows for embedding strings
within strings).
    vec[2] = left x of word
    vec[3] = width of word
    vec[4] = topy of word

If vec[1] is -1 x,y is after the end of the string or not in frame.
If vec[2] is -2 x,y is not in the window.
A sample call--
  Gvariable=yourframe mfindt mouse 8 mouse 9.
  +h60.+f0.'

<brread>(↑dread)
  'th75.tf1.
Makes a code vector out of keyboard input. See dread below.
  +h60.+f0.'

<reread>(↑reread :)
  'th75.tf1.
Used by redo and fix. Goes back n(its argument), prompts and does
a read from there. See reread below.
  +h60.+f0.'

<sub>(Ginput : sub :: SELF show. ↑input)
  'th75.tf1.
Evals its argument in a sub-window. Used by fix and shift-esc.
See sub below.
  +h60.+f0.'

<knows>(ev)
  'th75.tf1.
Whilst at the KEYBOARD, one can say
  +Oyourframe knows(DOIT)+O
and get a copy of the evaluator in the context of that instance of
disframe. Allows access to instance variables without going through
the +S path.
  +h60.+f0.'
frame ⇒ (apply frame)
  
  'h75.+f1.
  Draws a border of the given color around the frame. E.g.,
  yourframe frame -1.
  +h60.+f0.'

is ⇒ (ISIT eval)

isnew ⇒ (winx:.fwm. winwd:.fwmw. chary:.wny+.frmy.
  mark+.last-.chax+.reply+.justify-0. 
  (fnt:.(font)) (noframe:() frame black ))

dispframe knows

to dread t flag ( 
  disp-20. flag-false. mark-last.
  (null #DRIBBLE () DRIBBLE flush)
  repeat (050>disp-+t-kbd=(
    t=010⇒(last<mark⇒(disp-buf[last+1]))
    'h75.+f1.
    Backspace only up to prompt.
    +h60.+f0.'
    buf[last+1]=047⇒(flag=flag is false))
    'h75.+f1.
    Backspace out of string flips flag.
    +h60.+f0.'
    t=036⇒(flag=() done)
    'h75.+f1.
    DOIT checks if in a string.
    +h60.+f0.'
    t=047⇒(flag=flag is false)
    'h75.+f1.
    Flag is true if in a string
    +h60.+f0.'
    t=05⇒(sub (ev). last-last-1. disp show)
    'h75.+f1.
    Shift-Esc make sub-eval.
    +h60.+f0.'
    t=04⇒(disp-010. done print. disp-036. ↑(done))
)

disp-13. ↑read of stream of buf from mark+1 to last)

to sub disp ( 
  disp-dispframe winx+48 winwd-64 winy+14 winht-28 string 300.
  disp clear. (;)eval)
  'h75.+f1.
  Opens a sub-frame, and evals its argument in that context.
  +h60.+f0.'
to frame a (a = turtle at frm - 1 frm y - 1.
   at = Width + 2. at = Sink = \(0\) \(1\)
   do 2 (a turn 90 go frm wd + 2 turn 90 go frm ht + 2)
   'h75. + f1.
   Draws a double line around the frame.
   'h60. + f0.'

to reread n i p reader ((null \(n = (n + 1)\))
   n = mark. last - mark - 2. disp show.
   for i to n
      (n = buf [1 to p - 1] find last 20.
       p < 1 \(\text{done}\))
      i < n + 1 \(\text{error} \quad (\text{no code})\)
      \(\text{read of stream of buf from p + 1 to last} \)
      'h75. + f1.
      Counts back \(n\) prompts \((n\) is integer arg\) and then does a read from
      there. Also erases the line just typed.
      'h60. + f0.'

done
to dclear (CODE 52)
   'h75. + f1.
   This function takes five parameters --
   x width y height value, and dclear d the display rectangle thus
   defined to the dvalue d given. A 0 value, for example, puts all
   zeros into the rectangle.
   'h60. + f0.'

to dcomp (CODE 53)
   'h75. + f1.
   Just like dclear only complement rectangle.
   'h60. + f0.'

to dmove (CODE 54)
   'h75. + f1.
   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.
   'h60. + f0.'
to dmovec (CODE 55)
  '+h75.+f1.
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+S (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.
'+h60.+f0.'

to redo (^d(disp reread :) eval)
  '+h75.+f1.
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.
'+h60.+f0.'

to fix vec (^d(vec+disp reread :)
(disp sub (^d(veced vec)) eval)
  '+h75.+f1.
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.
'+h60.+f0.'

'+h90.+f2.
TURTLES
'+h60.+f0.'
to turtle var: pen ink width dir x xf y yf frame: f
  CODE 21 'πgo( (draw a line of length :)
  πturn( (turn right : (degrees))
  πgoto( (draw a line to : (x), (y)))
  π+S( : var, π+π( var + : )
    πvar eval)
  πpendn( πpen + 1. πSELF)
  πpenup( πpen + 0. πSELF)
  πblack( πink - 1. πSELF)
  πwhite( πink + 0. πSELF)
  πxor( πink - πU2. πSELF)
  πis( (ISIT eval)
  πhome( πx + frame +S (frmx+frmwd/2).
    πy + frame +S (frmy+frmht/2).
    πxf + πyf + 0. πdir +S 270. πSELF)
  πerase( (frame fclear. πSELF)
  πup( πdir +S 270. πSELF)
  isnew( πpen + πink + πwidth + 1.
    (πframe +S πframe + f)
    πat( (x, y). πxf + πyf + 0. πdir +S 270)
    SELF home)
  )

PUT turtle f dispframe 0 512 0 684 string 1 noframe.

.printf turtle.

'πh90. tf2.

THE TRUTH ABOUT FILES

+h75. tf1.

FILESMALL: *****Smalltalk file and directory definitions*****

also see <SMALLTALK> on Maxc for:
  install.sm, xfer, xplot

a file is found in a directory (πdirinstπO) by its file name (πOfnameπO), an
  **d has a one πpageπO, 512 character s
  **tring (πosadrπO). πOrvecπO is an opt
  **jonal vector of disk addresses used f
  **or random page access.

fi -
<directory> file <string> old -- finds an old file named <string> in <direct
  **ory> or returns false if does not exi
  **st or a disk error occurs.
`fi ←
(directory) file <string> new -- creates a new file or returns false if it a
  **lready 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 speci
**fied, the current default directory i
**s used.

(directory) file <string> delete -- deletes a file from a directory and deal
  llocates its pages. do not delete the
  **system directory (SYSDIR.) or bittabl
**e (SYS.STAT.), or any directories you
** create.

(directory) file <string> rename <string> -- renames file named by first stri
**ng in <directory> with second string.
** currently not implemented for direct
**ory files.

(directory) file <string> load -- loads a previously +Osaved+O Smalltalk virt
**ual memory, thereby destroying your c
**urrent state.

(directory) file <string> save -- saves Smalltalk virtual memory.

+Oleader+O and +Ocuradr+O are the alto disk addresses of page 0 and the curr
  nt page of the file, respectively.
**+Obytec+O is a character index into
**+Osadr+O.

+Odirty+O = 1 if any label block integers (+Onextp+O thru +Osn2+O) have been
**changed; = -1 if +Osadr+O has been ch
**anged; = 0 if the current page is cle
**an. the user need not worry about th
**is unless (s)he deals directly with t
**he label or +Osadr+O. it mighth be not
**ed here that multiple instances of th
**e same file do not know of each other
**s activities or +Osadr's.

+Ostatus+O is normally 0; -1 if end occurred with the last +Oset+O; a positiv
**e number (machine language pointer to
** offening disk command block (dcb))
**signals a disk error.
the next 8 integers are the alto disk label block. +Onextp+O and +Obackp+O a
**re the forward and backward alto addr
**ess pointers. +Olnused+O is currently
** unused. +Onumch+O is number of chara
**cters on the current page, numch must
** be 512, except on the last page. +Op
**agen+O is the current page number. pa
**ge numbers are non-negative integers,
** and the format demands that the diff
**erence in consecutive page numbers is
** 1. normal file access starts at page
** 1, although all files possess page 0
** (the +Oleader+O page). +Oversion+O n
**umbers > 1 are not implemented. +Osn1
**+O and +Osn2+O are the unique 2-word
**serial number for the file.

the class function +Oncheck+O checks that file names contain alphabetic or +O
**legal+O characters or digits, and end
** with a period.

+h60.+f0.'

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

&→ (17 CODE 50)
+h75.+f1.
fi<integer>, <string>, or <file> --
x is string⇒ (for i to x length (SELF+x[i]))
x is file⇒ (repeat (x end⇒ (done) SELF+x next))
(numch<bytec bytec+1⇒
(SELF set to write (pagen+bytec/512 bytec mod 512))
sadr[bytec]•x 0377
+h80.+f2.'
next ((word (7))
  'rh75.+f1.
  fi next word<integer> -- write integer.
  possibly increment pointer to word boundary.
  (0=bytec 1 (bytec-bytec+1)
   SELF :x/256. SELF = x mod 256.
   +h60.+f0.'

6)
  'rh75.+f1.
  fi next word -- read an integer
  (0=bytec 1 () bytec-bytec+1)
  ↑(SELF next*256) + SELF next
  +h60.+f0.'

into (16)
  'rh75.+f1.
  fi next into <string> -- read a string
  for i to :x length(x[i]=SELF next).↑x
  +h60.+f0.'

25) CODE 50)
  'rh75.+f1.
  fi next -- read a character
  (numch<>bytec-bytec+1)
  (SELF set to read (pagen+bytec/512)
   bytec mod 512 () ↑0)) ↑sadr[bytec]
  +h60.+f0.'

set to (end)(13)
  'rh75.+f1.
  fi set to end -- set file pointer to end of file.
  SELF set to read 037777 0
  +h60.+f0.'

write(5)
  'rh75.+f1.
  fi set to write <integer> <integer> -- set file
  pointer to :spage :schar. if current page is dirty,
  or ↑Oreset↑O, ↑Oset to end↑O or page change
  occurs, flush current page. read pages until
  pagen=spage. allocate new pages after end if
  necessary (~1 512 is treated as start of next
  page, i.e. pagen+1 0). bytec-schar
  +h60.+f0.'

read. 4) CODE 50)
  'rh75.+f1.
  same as ↑Owrite↑O except stop at end
  +h60.+f0.'
skipnext  (18 CODE 50)
 'h75. f1.
 fi skipnext <integer> -- set character pointer relative to
 current position. (useful for skipping rather than reading,
 or for reading and backing up, but $Oend$ may not work if
 $Obytec$ points off the current page) $bytec$ $bytec$ + :.
 +h60. f0.'

end  (10 CODE 50)
 'h75. f1.
 fi end -- return false if end of file has not occurred.
 nextp=0 (bytec<numch>(#false))#false
 +h60. f0.'

$S$  ($^$: EVAL)

flush  (12 CODE 50)
 'h75. f1.
 fi flush -- dirty=0 () write current page
 +h60. f0.'

writeseq  (22 CODE 50)
 'h75. f1.
 transfer words from memory to a file
 :adr. :count. for i=adr to adr+count-1
 (SELF next word ← mem i)
 +h60. f0.'

readseq  (21 CODE 50)
 'h75. f1.
 ...from a file to memory...(mem i ← SELF next word)
 +h60. f0.'

is  (ISIT EVAL)

remove  (dirinst forget SELF)
 'h75. f1.
 remove file from filesopen list of directory
 +h60. f0.'

close  (dirinst $S$ bitinst flush.
 SELF flush. SELF remove. $closed$
 'h75. f1.
 fi close or $fi$-fi close (if fi is global) -- flush bittable
 and current page, remove instance from filesopen list of
directory
 +h60. f0.'
shorten \( \rightarrow (\rightarrow \text{to.} \rightarrow \text{here}) \) (SELF shorten pagen bytec) 14 CODE 50
\[ ^*h75.+f1. \]
fi shorten to \(<\text{integer}>\ <\text{integer}>\ -- \text{shorten a file SELF} \text{set to read} :\text{spage} :\text{schar.} \ x\rightarrow \text{nextp.} \ \text{nextp}++0. \]
\( \neg \text{numch}\rightarrow \neg \text{schar.} \ \text{dirty}++1. \ \text{deallocate} x \text{and successors} \)
\[ ^*h60.+f0. \]

print \( \rightarrow (\text{disp} \rightarrow \text{fname}) \)
\[ ^*h75.+f1. \]
file prints its name
\[ ^*h60.+f0. \]

reset \( \rightarrow \) (11 CODE 50)
\[ ^*h75.+f1. \]
fi reset -- reposition to beginning of file
SELF set 1 0
\[ ^*h60.+f0. \]

intostring \( \rightarrow (\text{SELF} \text{set to end.}) \)
\( x\rightarrow \text{string bytec + 512 * pagen - 1.} \)
SELF reset.
\[ ^*\text{SELF} \text{next into} x \)

random \( \rightarrow \) (SELF set to end. \( \neg \text{rvec} \rightarrow \text{vector pagen.} \))
for x to rvec length ( \)
\[ \text{SELF set x 0.} \ \text{rvec}[x] \rightarrow \text{curadr}) \]
\[ ^*h75.+f1. \]
fi random -- initialize a random access vector to be used in fi set... new pages appended to the file will not be randomly accessed
\[ ^*h60.+f0. \]

pages \( \rightarrow \) (20 CODE 50)
\[ ^*h75.+f1. \]
fi: pages \(<\text{integer}>\ <\text{integer}>\ -- \text{out of the same great tradition as} \neg \text{Omem}+O \text{comes the power to do potentially catastrophic direct disk i/o} \text{(not for the faint-hearted).} \]
\( :\text{coreaddress.} :\text{diskaddress.} :\text{diskcommand.} :\text{startpage.} \)
\( :\text{numberofpages.} :\text{coreincrement.} \text{if} -1=\text{coreaddress,} \)
copy \( \neg \text{Osadr}+O \text{to a buffer before the i/o call. diskaddress} \)
\( (=1 \text{yields \neg Osadr}+O) \text{and diskcommand are the alto disk address and command. startpage} \text{is relevant if label checking} \)
\( \text{is performed. numberofpages is the number of disk pages} \text{to process. coreincrement is usually 0} \text{(for writing in same} \)
\[ \text{buffer) or 256 for using consecutive pages of core. use} \]
\[ \text{label block from instance of \neg Osadr}+O. \ \text{copy label block from instance. perform i/o call. copy \neg Osadr}+O \text{and label block} \]
\[ \text{into instance. if} -1=\text{coreaddress copy buffer to \neg \text{sadrr}.} \]
\[ ^*h60.+f0. \]
isnew: (G.fname ncheck: . fname is false)
  (error G.(bad file name) nil)
  (null G.dirinst->#curdir)
    (G.dirinst->directory +S defdir) is directory
    (dirinst open) error G.(illegal directory))
  'th75.+f1.
  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
  +h60.+f0.'

exists: (24 CODE 50. G.fname)
  'th75.+f1.
  return false if file name does not occur in the
directory
  +h60.+f0.'

delete: (15 CODE 50. G.deleted)
  'th75.+f1.
  delete a file (see intro)
  +h60.+f0.'

sadr + (G: using: :) string 512).
  'th75.+f1.
  set up file string buffer
  +h60.+f0.'

rename: (G: x ncheck: . x is false)
  (error G.(bad new name) nil)
  file x exists: (error G.(name already in use))
  2 CODE 50. G.fname + x. 23 CODE 50.
  SELF set 0 12. SELF + fname length.
  SELF + fname. SELF flush. G.fname)
  'th75.+f1.
  check that the new name is not already in use.
  lookup the original file and change its name in its
directory, and in its leader page
  +h60.+f0.'

load: (2 CODE 50. 8 CODE 50)
old (2)
sadr[14 to 13+fname length] ← fname.
new (dirinst ≠S filinst is file (3) 19)
1) CODE 60.
   'h75. ≠f1.
   find an old file or add a new entry (with its
   name as a BCPL string in its leader page. special
   handling for new directories). machine code may
   return false
   ≠h60. ≠f0.'

save (SELF set to write 256 0. SELF reset.
dp0 close. 9 CODE 50)
   'h75. ≠f1.
   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)
   ≠h60. ≠f0.'

intostring (↑SELF intostring)
dirinst remember SELF ))
   'h75. ≠f1.
   finally, file puts itself into the filesopen list of
   **its
   directory
   ≠h60. ≠f0.'

file ≠S(ev)
to ncheck str i x :: legal (≠str<:.
    (str is string ≠str length<255 ≠str false) ≠false)
   for i to str length
      ≠x+str[i].
      140 < x < 0173 (lowercase)
      057 < x < 072 (digit')
      0 < legal[1 to 6] find x (legal')
      0100 < x < 0133 (uppercase')
      ≠false)
    x=056 (≠str) ≠str+ (≠str chars)
   'h75. ≠f1.
check that the file name is a proper length string containing only lower/uppe
   **r case letters, digits, or legal char
   **acters. if name does not end with a p
   **eriod, append one.
   ≠h60. ≠f0.'
PUT ncheck $legal q string 6
+-().
done

'$.h75.$f1.
adirectory is found in a directory ($Odirst+O), has a bittable file ($Obit
 **inst+O) for allocating new pages, a f
 **ile of file entries ($Ofilinst+O -- f
 **ile names, disk addresses etc.), and
 **a list of currently open files ($Ofil
 **esopen+O which is an $Obset+O). the
 **top level, $Odistinguished node+O of
 **the directory structure is the system
 ** directory $Od0p0+O (see $Odirectory k
 **nows+O below if you also want $Od0p1+O
 **). dp0 knows the disk number ($Od0r
 **st+O) and the true identity of the bi
 **ttle. each file must ask its direct
 **ory for the bittable when page alloca
 **tion is necessary, and the system dir
 **ctory (via its local directory) for
 **the disk number.

$di +-
<directory> directory <string> old/new
currently, <directory> and old or new must be specified.

$Od0rname+O is the system directory name and $Ob0tname+O is the bittable name
 **. $Ocurdir+O is a class variable boun
 **d to the last directory instance $Oop
 **ened+O, and provides information $Owh
 **o called you+O (i.e. CALLER) to a fil
 **e or directory. $Odefdir+O is a defau
 **t directory, initially set to dp0, w
 **hich is invoked when $Ocurdir+O fails
 ** to be a directory, i.e. file was not
 ** called in the context of a directory
 **, but globally

'$.h0.$f0.'
(to directory name exp : dirinst bitinst filinst filesopen : dirname bitname
curdir defdir (}

$$\text{file}$$ (SELF open. $$\text{apply file}$$)
$$'h75.tf1.$$  
di file $$<\text{string}>$$... -- open directory. create file instance  
(see file intro)  
$$'h60.tf0.$$'  

$$\text{directory}$$ (SELF open. $$\text{apply directory}$$)
$$'h75.tf1.$$  
di directory $$<\text{string}>$$... -- open directory. create directory  
instance  
$$'h60.tf0.$$'  

$$\text{open}$$ (curdir+SELF. filinst is file ()
(bitinst)0 (bitinst + dirinst +S bitinst.  
filinst + file filinst new)
filinst + file filinst old.
(bitinst + (dirinst is directory-)  
(dirinst +S bitinst) file bitname old)).
dirinst is directory- (dirinst remember SELF))
$$'h75.tf1.$$  
di open -- (normally not user-called since access  
to the directory always reopens it) initialize  
directory file and bittable instances. directory  
(except for $$\text{top}$$ level+0) puts itself into  
filesopen list of its directory  
$$'h60.tf0.$$'  

$$\text{is}$$ (ISIT eval)

$$\text{print}$$ (disp-0133. filesopen print. disp-0135)
$$'h75.tf1.$$  
di or di print. --print the filesopen list  
$$'h60.tf0.$$'  

$$\text{map}$$ (SELF open. $$\text{exp-}$$: filinst reset.  
repeat (filinst end (cr. done)  
1024 > $$\text{name}$$- filinst next word  
(name < 2 ( filinst skipnext 2*name-1)  
filinst skipnext 10.  
(name filinst next into string filinst next.  
exp eval))
$$'h75.tf1.$$  
di map expression -- evaluate an expression for  
each file name  
$$'h60.tf0.$$'
list => (SELF map (disp-name. sp))
  'h75.+f1.
di list -- print the entry names contained in
filinst
  'h60.+f0.'

remember => (filesopen + :)

forget => (filesopen delete :)
  'h75.+f1.
...add or delete file instances in filesopen
  'h60.+f0.'

close => ((filinst is file) (filesopen map (vec[end] close).
  (dirinst is directory) (dirinst forget SELF)).
  filinst + filinst +S fname.
  bitinst flush. (bitinst + tU1)). (closed)
  'h75.+f1.
di close (e.g. dp0 close) or di di close (to
release instance) -- close a directory by closing all
files and directories in its filesopen list and
deleting it from the filesopen list of its directory.
this is currently one way to regain space by
closing unwanted file instances, and to change disk
packs
  'h60.+f0.'

use => (defdir + SELF)
  'h75.+f1.
di use -- change the default directory
  'h60.+f0.'

:+S => (:+(G) eval)

isnew => (filesopen + obset. dirinst + curdir.
  filinst + :.
  dirname = filinst => (bitinst + tU1. curdir + SELF)
  'h75.+f1.
store the directory file name in filinst and flag
old/new in bitinst. system directories are not
opened
  'h60.+f0.'

  bitinst + (new => (1) (old. tU1). SELF open)))
directory +$S(ev)
Gdirname + q string 7
SYSDIR.
Gtypename + q string 9
SYS.STAT.

'+h75.+f1.
names of the system directory and bitable
+h60.+f0.'

curdir+0. Gdefdir<dp0+directory dirname.

'+h75.+f1.
create the system directory instance (the initial default)
on disk 0 in a closed state. to initialize a second disk:

directory+$S (Gcurdir + 1. Gdp1 + directory dirname)
+h60.+f0.'

done

to error adr ptr arec class :: c shocode find sub (  
(0=Gadr+mem 0102->(Gknows,ev #) dson :ptr))

Garec=leech AREC.
disp sub G((0=aadr(ptr print)
mem 0102+0. disp-0377 mem adr.
  for adr+adr+1 to adr+(mem adr)U9 (  
    Gptr+mem adr.
    disp+ptr[0;]U8. disp+ptr[0377])

c r c ev))

error knows
to c class code cpc (  
  (GET class GTITLE) print. G: print.
  find arec[1]GET class GDO (shocode).
 )
to shocode i (  
  for i+1 to code length
    (i+cpc-5->(disp-056) i+cpc+5->(disp-056)
    sp. (i=cpc->(disp-031))
    code[i] is vector (G() print) code[i] print).
 )
to find adr vec vadr 1 (  
  '+h75.+f1.a tree search in vec for the address adr+h60.+f0.'
  Gadr+: G1=leech :vec.
  vec is vector false (↑false)
  Gvadr+(leech 1)[1]+1.
  (adr>vadr>>(adr+vadr+vec length+1) 
    (Gcpc + adr-vadr. G[1]=0. Gcode=vec. ↑true)))
  Gl+0. for 1 to vec length
    (vec[] is vector (find adr vec[1] (↑true)))
  ↑false)
to sub disp (disp + GET USER disp. (;) eval)
done

to kbck (1 CODE 20)
  'h75.+f1.
  Returns true if the keyboard has been hit.
  'h60.+f0.'

to button n (↑:n=mouse 7)
  'h75.+f1.
  Returns true if that pattern is being held down
  'h60.+f0.'

'x690.+f2.
THE SMALLTALK EDITOR ---
'x680.+f0.'

to edit func t (#func.
  t+GET func DO.
  null t ⇒ (↑ffl(no code))
  ↑ffl(title⇒ ((veced classprint func header) eval)
  PUT func DO veced t.
  ↑ffl(edited)
  'h75.+f1.
  Edit picks up a code vector, makes sure it is not empty and calls
  veced to edit the code body. If you say edit foo title, veced will
  edit the header as well, and the changed form will be evalled upon
  exit to redefine the function, title and all.

  Veced 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.
  'h60.+f0.'
(to vaced back newdisp menu x :: menuwidth menulen menustr
    ed edpush edtarget gettwo bugin getvec (knows (ev)
    back false.
    disp fclear.
    disp+ (menu dispframe winx winwd menuwidth
        winy (winht 139 winht 140) string 70.
        menu menustr.
        mem 0425 winy + 103.
        newdisp dispframe winx winwd menuwidth+2
        winy winht string buf length noframe)
    :x. x disp newdisp (ed x).
    disp show.
    $x) )

veced knows

menuwidth 64.
menustr string 0.
menulen 10.
do menulen (x-q string 9.

    menustr menustr+x[1 to x[1 to 9] find 13]).

Add
Insert
Replace
Delete
Move
Up
Push
Enter
Leave
Exit

# indisp disp (disp $ (eval)
  '+h75.?1.
  used to make DISP a new local.
  '+h60.?f0.'
to ed ptr l n nr
run command temp i nv n1 fnth hfnth (
  \command ← 0.
  \:ptr.
  \fnth ← mem ((mem 70)-2).
  \hfnth ← fnth/2.
  repeat( \l←ptr length.
    back⇒(done with ptr)
    mem 0424 ← menu ↑S (winx + winwd/2).
    menu show. disp clear
    \nv←0.
    for n to l-1
      (ptr[n] is vector⇒(disp←044. sp
        \nv←nv+1. \n1←n)
      ptr[n] print. disp←32)
    cr cr.
  \command ← (edcomp (bugin menu menuen) both).
  mem 0424 ← disp ↑S (winx + winwd/2).

  (\ptr←vecmod ptr l 0 read)
  (\ptr←vecmod ptr (edcomp edtarget both) 0 read)
  (gettwo. \ptr←vecmod ptr n nr
run read)
  (gettwo. \ptr←vecmod ptr n nr
run nil)
  (gettwo. \temp ← ptr[n to n+nr
run]
    temp[nr
run + 1] ← nil.
    \i←(edcomp edtarget both).
    \ptr←vecmod ptr n nr
run nil.
    (i>n ⇒ (\i←i-nr
run))
    \ptr←vecmod ptr i 0 temp)
  (getvec⇒(\ptr←vecmod ptr n 1 ptr[n]) again)
  (gettwo. edpush)
  (getvec⇒(ptr[n]+ed ptr[n]) again)
  (done with ptr)
  (\back←true. done with ptr)
  ) [command] eval.
)

'th75.tf1.

The heart of ED 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.

'th60.tf0.'
to edpush ins (ins+vector 2.
  ptr+vecmod ptr n nrun ins)

to gettwo t1 n2 (n+(edcomp edtarget top).
  n2+(edcomp edtarget bot).
  nrun ← 1+n2-n.
  nrun<1⇒(n=n2. nrun+2-nrun))

to bugin someframe max index(
  :someframe.
  max ← 1+:.
  repeat (button 0 ⇒ (repeat (button 7 ⇒(disp sub (ev))
    button 0 ⇒())
  done)
  index ←someframe mfindt mouse 8 mouse 9
  0<index[1]< max ⇒
    (↑index)
  '↑h75.↑f1.
  returns token index, if within range, else
  ↑h60.↑f0.'
  again
  ↑h75.↑f1.
  causes an exit out of this command by restarting ed's
  repeat
  ↑h60.↑f0.'
)

to edtarget (↑ bugin disp 1)

to getvec (nv=1⇒(n+n1. ↑true)
  ↑ptr[n+(edcomp edtarget both)] is vector)

to edcomp compvec y hth (:compvec.
  y+compvec[4].
  hth←(↑both↑(fnth)↑top↑(hfnt))
  ↑bot↑(y+y+hfnt. hfnt))

↑compvec[1]
)

done

'↑h90.↑f2.
BOOTSTRAPPING REVISITED
'↑h60.↑f0.'

to classprint fn a b i j k flags clsv clsm arecv arecm instv instm code (  
: #fn. code ← GET fn (DO. null code (no code))
            a ← leech #fn. b ← vector 1. clsm ← clsm + 1. arecm ← 0.  
k ← a[1] clsv ← vector k. arecv ← vector k. instv ← vector k.
            '↑h75.↑f1.
Pull symbols out of class table
'↑h60.↑f0.'

for i + 4 to 4 + 2 * k by 2
'↑h75.↑f1.

k is no. dbl entries - 1, here
'↑h60.↑f0.'

(  
        k ← (0 - 1) (again). flags ← k U 14.
            '↑h75.↑f1.
0 = class, 2 = arec, 3 = inst
'↑h60.↑f0.'

flags = 0 (0 = (DO TITLE SIZE) [1 to 3] find a[i] (  
            clsv ← clsm + 1 ← a[i])
   b[2] ← k 3777. j ← a[i + 1] (  
 )

            '↑h75.↑f1.
Now make up input form.
'↑h60.↑f0.'

        a ← vector 6 + arecm + instm + clsm.
    a[3 to 6 + arecm] ← arecv.
    (0 < instm + clsm ← a[j-1] ← a[j+1] ← instv.
              0 < clsm ← a[j-1] + clsv . a[j+1] + clsm + clsv))

↑header ← a[j+1] ← code. ↑a
for i to j (a[i] print. disp ← 32)
showpretty ← (pshow code 3) code print)


to show showpretty (showpretty ← true. showev (: )}
to showev shAtom shVal (:shAtom cr.
  (shAtom is atom⇒
    (shVal ← shAtom eval.
    (null GET shVal GDO⇒
      (print. shAtom print. G← print.
      (shVal is vector⇒ (print shVal)
        (null shVal⇒(nil print))
        shVal print. G. print)
      classprint shVal))
  shAtom print)
  disp<30.)
'th75.+f1.
*****Keyboard translation*****
'th60.+f0.'

to kbd (↑kmap[TTY])
Gkmap ← string 0377.
  for i=0200 to 0377(kmap[i] ← 0177) '↑f1.ILLEGAL GETS ←↑f0.'
  for i=001 to 0177(kmap[i] + i) '↑f1.+f1.REGULAR ASCII↓↑f0.'

  '↑f1.CHAR -- KEYBOARD↑f0.'
  '↑f1.+A -- ↑+A or ↑,
  **or ↑+SHF,↑f0.'
    '↑f1.+B -- ↑+B or any TOP BLANK
  ** KEY.↑f0.'
  kmap[0343]←kmap[0303]←kmap[0272]←03. '↑f1.+C -- ↑+C or ↑+SHF;↑f0.'
  kmap[0344]←kmap[0304]←04. '↑f1.+D "done" ↑f0.'
  kmap[0345]←kmap[0305]←kmap[023]←05. '↑f1.+E -- ↑+E or ↑+SHF ESC↑f0.'
  kmap[0346]←kmap[0306]←kmap[0262]←06. '↑f1.+F -- ↑+F or ↑+2↑f0.'
  kmap[0347]←kmap[0307]←kmap[0273]←07. '↑f1.+G -- ↑+G or ↑+↑↑↑f0.'
  kmap[035]←kmap[0315]←kmap[0275]←015. '↑f1.+L -- ↑+L or ↑+↑↑↑↑f0.'
  kmap[0357]←kmap[0317]←kmap[0242]←017. '↑f1.+O -- ↑+O or ↑+SHF↑f0.'
  kmap[0360]←kmap[0320]←kmap[0271]←020. '↑f1.+P -- ↑+P or ↑+9↑f0.'
  kmap[0361]←kmap[0321]←kmap[0261]←021. '↑f1.+Q -- ↑+Q or ↑+↑↑↑f0.'
  kmap[0362]←kmap[0322]←kmap[0300]←022. '↑f1.+R -- ↑+R or ↑+SHF2↑f0.'
  kmap[0363]←kmap[0323]←023. '↑f1.+S -- ↑+S↑f0.'
  kmap[0364]←kmap[0324]←024. '↑f1.+T -- ↑+T↑f0.'
  kmap[0366]←kmap[0326]←kmap[0265]←026. '↑f1.+V -- ↑+V or ↑+5↑f0.'
  kmap[0367]←kmap[0327]←kmap[0376]←027. '↑f1.+W -- ↑+W or ↑+SHF6↑f0.'
  kmap[0370]←kmap[0330]←kmap[0246]←030. '↑f1.+X -- ↑+X or ↑+SHF↑f0.'
  kmap[0371]←kmap[0331]←kmap[0277]←031. '↑f1.+Y -- ↑+Y or ↑+SHF/↑f0.'
kmap[0372]-kmap[0332]-kmap[0276]-kmap[0256]-032.
   'tf1.+Z --- ++Z or ++.
   ** or ++SHF.+f0.'
kmap[0333]-kmap[0264]-033. 'tf1.[- ++[ or ++4+f0.'
kmap[0334]-kmap[0267]-034. 'tf1.\ ++\ or ++7+f0.'
kmap[0335]-kmap[0375]-035. 'tf1.+] +++]f0.'
kmap[0337]-kmap[0336]-kmap[0222]-kmap[0212]-kmap[022]-kmap[012]-036.
   'tf1.++ -- LF or ++ + or ++SHF++f0.'
kmap[0247]-0174. 'tf1.\ -- SHF\ or ++?+f0.'
kmap[0257]-0176. 'tf1.> -- SHF6 or ++?f0.'
kmap[0263]-043. 'tf1.# -- SHF3 or ++3+f0.'
kmap[0270]-052. 'tf1.* -- SHF8 or ++8+f0.'
kmap[0220]-kmap[0210]-kmap[020]-010. 'tf1.ALL BS'S+f0.'
kmap[0225]-kmap[0215]-kmap[025]-015. 'tf1.ALL CR'S+f0.'
kmap[0240]-kmap[0230]-kmap[030]-040. 'tf1.ALL SP'S+f0.'

to filout disp flist i showpretty (showpretty <- pretty.
   dsoff (:disp is string⇒ (disp+file disp⇒ () error (file error))
   (add⇒(disp set to end))
   (null :flist⇒(defs map (showev vec[i]. cr))
   (list is atom⇒ (showev flist. (list+flist eval))
   for i to flist length-1 (showev flist[i]. cr)
   disp shorten to here. disp close. dson.)
   '+75.+f1.

Filout basically does a show in an environment where the display
is replaced by a file. filout pretty <file> or <string = file name> add
<vector> if +Opretty+O is used, the text representation is neater but
takes longer to generate. if +Oadd+O is used, function definitions ar
**e

appended to the file. if <vector> is not specified, +Odefs+O is
used.
'+60.+f0.'

to filin fi (dsoff
   (:fi is string⇒(fi < file fi old⇒())
   dson ↑false))
   repeat
   (fi end⇒(done)
   dsoff.
   cr (read of fi) eval print.
   dson).
   fi close.
)
   '+75.+f1.

Filin basically does a read-eval-print loop, but gets its input from
**a

file instead of a dispframe.
'+60.+f0.'
to type f t (((f is string⇒
  ⇒f + file f old⇒(f remove)
  ⇒false))
  ⇒t + string 30.
  repeat(f end⇒(done) disp + f next into t))

to t fool :: x (⇒knows⇒(ev)
  ⇒disp + dispframe 16 480 514 168 string 520.
  disp + version. ⇒defs + obset.
  ⇒fool + # to. to toAtm (CODE 19 defs + toAtm. toAtm)
  PUT USER ⇒do (cr read eval print). ⇒t + 0.)

⇒version + q string 26
SMALLTALK of April 3

to expand x (:: x. disp + S (⇒winy +⇒frmy + winy + x. frame black)
  disp show CODE 38)
  ⇒'h75 + t1.
  t is called to set up a display frame, and defs and then self-destruc
  ⇒*t1
  to save space. expand can be called to grab some storage from the
  display area to augment the SMALLTALK workspace. expand 200
  would take 200 lines off the top of the display and increase core by
  6400 words.

THE SMALLTALK READ ROUTINE (name changed to protect ev)
  ⇒'h60 + t0.'

(to junta scanner :: read1 tablscan rdbuf mknum rdstr rtb1 type
  letbit digitb sepbit atbits qtbit
  ⇒S ⇒(⇒ evalu)
  ⇒of⇒((⇒ scanner is string⇒
    ⇒(⇒ scanner + stream of scanner))
  ⇒scanner + tablscan scanner type.
  ⇒read1 rtb1)
  ⇒disp read))

junta + S (ev)

to read1 rbuf rdto flag (}
  ⇒rdto.
  ⇒rbuf + stream of vector 10.
  scanner read.
  ⇒rbuf contents)
to tablsan mask : source type seq isfil nxtchr

next

(CODE 14 next.
'CODE 14 is equivalent to...
:mask=0>(string 1. t[1]=nxtchr.
	nxtchr=source next. ▲atom t)

seq reset.
repeat
(0 = nxtchr=done).
0 = mask ▲type[nxtchr + 1]=(done).
seq = nxtchr.

tnxtchr = source next)

seq contents")

_skip=tnxtchr = source next

read=repeat
(rdaltb[nxtchr + 1]eval))

isnew=

(:source.
 :type.
 seq = stream.
(source is file=(isfil + 1))
SELF skip))

to rdnum sign base n fs(

sign = (nxtchr=025>(scanner skip. ▲U1)1).
base = (nxtchr=060>(8)10).
n = mknum scanner next digbit base.
flag = false.
056 = n = n =

(sign=next digbit

0=fs length=flag=true. ▲sign*n)

n = n + (mknum fs 10)/10.0 ipow fs length.
nxtchr=0145>(scanner skip. ▲n*(10.0 ipow rdnum)*sign)

▲= sign*n)


to mknum str base n i(

:str.
 :base.
 n = 0.0.
for i to str length

(=n = (n*base) + str[i]-060)

▲n)

to rdstr t (scanner skip.

t=scanner next qrbit.
 scanner skip.

nxtchr=047>(seq=047. ▲seq contents+rdstr)

▲t)
'rh75.⇒f1.INITIALIZATION OF READ TABLES
\+h60.⇒f0.'
\+rtbl \+ vector 256.
\+type \+ string 256.
\+sepbit \+ 2 \* \+letbit \+ 2 \* \+digbit \+ 2 \* \+qtbit \+ 1.
\+atbits \+ letbit \+ digbit
to scanner n v i j (\n  \n    .n .v. repeat (\n        i. \n            ( \to⇒(j. for k+i+1 to j+1 (\n                type[k]=n. rtbl[k]=v))\n                type[i+1]=n. rtbl[i+1]=v)\n            \+\+and⇒() done))
scanner 0 (rbuf⇒scanner next 0) 0 to 0377.
snanner letbit (rbuf⇒atom scanner next atbits) 0101 to 0132 and 0141 to 0172 **.
snanner digbit (rbuf⇒rdnum. flag⇒(rbuf⇒\+).) 060 to 071 and 025.
snanner sepbit (scanner next sepbit) 011 and 014 and 015 and 040.
snanner qtbit (rfbuf⇒ rdstr) 047.
snanner 0 (scanner skip. rbuf ⇒ (read1 rtbl1) eval) 020.
snanner 0 (scanner skip. rbuf ⇒ read1 rtbl1) 050.
snanner 0 (scanner skip. rbuf ⇒ nil. done) 051.
snanner 0 (rfbuf⇒ nil. done) 0 and 036.
for i to type length (type[i]⇒ type[i]\+qtbit)
done
\+read ⇒ \#junta.
\+rh75.⇒f1.⇒f1.allocates display over OS after setting up \+h60.⇒f0.'
to junta (PUT USER \+DO \+t. CODE 31)
'rh75.⇒f1.allocates display over OS after setting up \+h60.⇒f0.'