to class x y ()
to number x y :: nprint ()
to vector x y :: substr ()
to atom x y (CODE 20)
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
true, /::[]*#(->)={>()*<>}<> goto turn next contents end [] <=< <= min max)

'th75.tf1.
DONT EDIT ABOVE HERE! These classes and atoms mentioned early to guarantee addresses for mach
**ine code.

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

TO PRINT ALLDEFS ------

Get ALLDEFS into Bravo (with font ST8.AL, already FONT.0 for those
who have run any of the STBRAVO*.CM command files) and translate
onto another file (e.g. ALLDEFS.TR). Get ALLDEFS,EC from MAXC and
type it. You must have all the .EP files listed therein. Then you may type

GEARS/D ALLDEFS/G ALLDEFS.TR

'th90.tf2.
BOOTSTRAPPING MAGIC.

'th75.tf1.

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

'th60.tf0.'

to print (<..)
'th75.tf1.
..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.
'th60.tf0.'

to read (CODE 2)
'th75;II;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.
'th60.tf0.'
'th90.+f2.
MESSAGE HANDLING
'th60.+f0.'

to : (CODE 18)
'th75.+f1.
  to : name

∧G ⇒ (G 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.

∧& ⇒ (G 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.

(G 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.
'th60.+f0.'

to ∧ (CODE 17)
'th75.+f1.
 :G 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.
'th60.+f0.'

to ? ( CODE 39 )

to $ ( CODE 36 )

to ↑ (CODE 13)
'th75.+f1.
 :x. then do a return, and apply x to any further message. Note
that in (.. ↑x+3. G y+y-2), the assignment to y will never
happen, since $ causes a return.
'th60.+f0.'

to G (CODE 9)
'th75.+f1.
↑G. That is, get the next thing in the message stream unevalled
and active return it (which causes it to be applied to the message).
'th60.+f0.'

to #: (#)
'th75.+f1.
Returns a REFERENCE to its argument’s binding.
'th60.+f0.'

'th90.+f2.
CONTROL CLASSES
'th60.+f0.'

to repeat token (:#token. CODE 1)
'th75.+f1.
repeat (token eval) Not a true apply to eval, and therefore token
MUST be a vector.
'th60.+f0.'

to done x (<with> (x. CODE 25) CODE 25)
'th75.tf1.
done causes a pop out of the nearest enclosing repeat, for, or do.
"done with val" will cause the repeat to have value val
'th60.tf0.'

to again (CODE 6)
'th75.tf1.
repeat (Gactive-active caller. eq active. class #repeatG(done)).
That is, redo the most recent repeat, for, or do loop.
'th60.tf0.'

to if exp (:expG(\thenG(:exp. \elseG(\#. exp)exp)error G(no then))
\thenG(\#. \elseG(exp) false error G(no then))
'th75.tf1.
The ALGOL "if ... then ... else ..."
'th60.tf0.'

to for token step stop var start exp (G
\varG + s.
\startG = (\#(\#(1).
\stopG = (\#tokenG(\#(\#start).
\stepG = (\#byG(\#(1).
\doG :#{exp. CODE 24}
'th75.tf1.
An Algol-like "for".
Note the default values if "*","to","by", etc., are omitted.
CODE 24 means --repeat(exp eval). This implies "done" and "again"
will work, which is correct.
'th60 tf0.'

to do token step stop var start exp (G
\stepG + G\startG - 1. :stop. :\#exp. CODE 24)

'th90.tf2.
INITIALIZING SYSTEM CLASSES

'th75.tf1.
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 intentional.
'th60 tf0.'

to PUT x y z (:\#x. :\#y. :\#z. CODE 12)
'th75.tf1.
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
permitting an atomic name which was the second argument.
'th60 tf0.'

to GET x y (:\#x. :\#y. CODE 28)
'th75 tf1.
If "x" is a class table then the binding of the atom in "y" will be
fetched.
'th60 tf0.'

to leech field bits : ptr (CODE 27
isnewG(ptr))
'th75 tf1.
Lets you subscript any instance
a[0] gives 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.
'th60 tf0.'
PUT USER @TITLE @USER
PUT falseclass @TITLE @false

PUT atom $DO $(CODE 29
'fh75. tf1.
$⇒(x: ?x -- Lookup SELF and replace its value by x.)
$eval$(? -- Lookup the binding of SELF)
$⇒(SELF?:)
$chars$(? -- printname of SELF (a string))
'th60. tf0.'

$⇒(ISIT eval)
$⇒print%(disp=SELF chars )
'th75. tf1.
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.
'th60. tf0.'

to ev (repeat (or read eval print))

PUT falseclass $DO $(CODE 11
'th75. tf1.
$⇒ $(: $).
$⇒or $(: $).
$⇒and $(: $).
$⇒< $(: $).
$⇒⇒ $(: $).
$⇒$(: $).
'th60. tf0.'

$⇒(false$(true) ? $(false) $SELF)
$⇒print%(false print )

PUT vector $DO $(CODE 3 $substr SELF x GLOB MESS)
'th75. tf1.
isnew$(Allocate vector of length :.
(Fill vector with nils.)
$⇒[x: ix ]
 (?⇒(y: iy -- 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...
'th60. tf0.'

$⇒(ISIT eval)
$⇒+(y: y is vector $(x=SELF[1 to SELF length+y length],
 h x[SELF length+1 to x length]+y[1 to y length])
 error $(vector not found))
$⇒map$(y. for x to SELF length
 (evapply SELF[x] to y))
$⇒print%(disp=40. for x to SELF length
 (disp=32. SELF[x] print). disp=41)
)

PUT string $DO $(CODE 3 $substr SELF x GLOB MESS)
'th75. tf1.
isnew$(Allocate string of length :.
(Fill string with 0377s.)
$⇒[x: ix ]
 (?⇒(y: iy -- store y into xth element. )
 $ xth element )
$⇒length$(? length of string or vector) th60. tf0.
\$is\Rightarrow(ISIT eval)
\$=\Rightarrow(y \text{ is string}\Rightarrow(SELF length\Rightarrow(y length\Rightarrow(
\text{for } x \text{ to SELF length } (SELF[x]{}@x[/]{}@()) \Rightarrow falses)) \Rightarrow falses)
\Rightarrow falses)
\$+=\Rightarrow(y \text{ is string}\Rightarrow(\$x+SELF[1 \text{ to SELF length}+y \text{ length}]\Rightarrow
\text{falses})[SELF length+1 \text{ to } x \text{ length}]+y[1 \text{ to } y \text{ length}])
\text{error } \$=(\text{string not found}))
\$print\Rightarrow(0 = \$x+SELF[1 \text{ to 9999}] \text{ find first 39s}\Rightarrow
\text{(disp }+ 39, \text{ disp }+ \text{ SELF. disp }+ 39)
\text{SELF[1 to x+1] print. SELF[x+1 to SELF length] print})
)

PUT number $G^*DO G^*(CODE 4
1'@h75.\text{tf1.}
\$+=\Rightarrow(\$val+;)
\$-=\Rightarrow(\$val-;)
\$=*\Rightarrow(\$val*;)
\$/=\Rightarrow(\$val/;)
\$=\Rightarrow(\$val:)
\$>=\Rightarrow(\$val>;)
\$<\Rightarrow(\$val<;)
\$<=\Rightarrow(\$val<=;)
\$max\Rightarrow(\$x+; \text{ SELF}<x\Rightarrow(\$SELF) \Rightarrow x)
\$min\Rightarrow(\$x+; \text{ SELF}>x\Rightarrow(\$SELF) \Rightarrow x)
\$=\Rightarrow(\$val OR ;)
\$=\Rightarrow(\$val XOR ;)
\$=\Rightarrow(\$val AND ;)
\$/=\Rightarrow(\$val LSHIFT ;))
"h60.\text{tf0.}"

\$is\Rightarrow(ISIT eval)
\$print\Rightarrow(SELF>0\Rightarrow(nprint SELF)
\text{SELF}=0\Rightarrow(disp=060)
\text{SELF}=01000000\Rightarrow(disp=base8 SELF)
\text{disp}=025. \text{nprint 0-SELF) }
1'@h75.\text{tf1.}
\text{For floating point stuff see FLOAT
"h60.\text{tf0.}"

to - x (x*^1)
1'@h75.\text{tf1.}
\text{An often used abbreviation, has to work for float as well.}
"h60.\text{tf0.}"
to base8 i x s (x. $G^*S$-string 7. for i to 7
(\$[8-i] \Rightarrow 060 \times \text{ [7. } \$G^*x \times x \{3\}. \text{ } \$s)
1'@h75.\text{tf1.}
\text{Returns a string containing the octal representation (unsigned) of its integer argument.}
"h60.\text{tf0.}"

$G^*ISIT + G^*(\$?\Rightarrow(\$TITLE) \Rightarrow TITLE=\$).
to nil x (\#x)
1'@h75.\text{tf1.}
nil is an "unbound pointer", which is used to fill vectors and tables.
"h60.\text{tf0.}"
to null x (x. 1 CODE 37)
1'@h75.\text{tf1.}
\text{Null returns true if its message is "nil", otherwise false.}
"h60.\text{tf0.}"
to eq x (CODE 15)
to disp * x i (  
<\rightarrow>(x is string) for i to x length (TTY+x[i])) TTY+x)  
<cangle<clear>(x) <sub>(x eval))  
'th75.+f1.  
This disp is used for bootstrapping. Later in these definitions  
(READER) it will be restored to an instance of "display frame."  
'th60.+f0.'

to TTY (0 CODE 20)  
'th75.+f1.  
TTY+integer> will print an ascii on the Nova tty. On altos, TTY  
prints in little error window at bottom of screen.  
'th60.+f0.'

to dosff (mem 272+0)  
'th75.+f1.  
Turns display off by storing 0 in display control block ptr. Speeds  
up Alto Smalltalk by factor of 2.  
'th60.+f0.'

to dson (mem 0420 + 072)  
'th75.+f1.  
Turns display back on by refreshing display control block  
pointer.  
'th60.+f0.'

to apply x y (:@x, <!to:=y. <!in:=GLOB CODE 10 CODE 10)  
<!in:=GLOB CODE 10 CODE 10)  
'th75.+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.  
'th60.+f0.'

to cr (disp+13). to sp (disp+32)

'true+true  
eval+false  
to is ( ?x=?y untyped) ?false)  
'th75.+f1.  
These are used to handle messages to classes which can't answer  
questions invoking "is", "eval", etc.  
'th60.+f0.'

to t nprint substr (ev). t  
'th75.+f1.  
preserve to from making these global.  
'th60.+f0.'

to nprint digit n (:n=0-)  
<\rightarrow>(n mod 10. nprint n/10. disp+060+digit)  
PUT number <nprint #nprint.  
'th75.+f1.  
Prints (non-neg) integers in decimal with leading zeroes suppressed!  
'th60.+f0.'

to substr op byte s lb ub s2 lb2 ub2 (  
@s. @lb. @ub. MESS. GLOB+ub. 'tf1.tee hee+th60.+f0.  
.s. ub. (@s) error G\rightarrow>(missing right bracket))  
<\rightarrow>(byte + <\rightarrow>(lb2 + ub2 + 1.  
<hfind> (<\rightarrow>(op + (<\rightarrow>(first+1) <\rightarrow>(last+2) 1)  
+ (<\rightarrow>(non+0) 0). :byte. CODE 40)  
<\rightarrow> (<\rightarrow> (byte. G\rightarrow>op+0. CODE 40)  
@s+2. G\rightarrow>op+5.
to vecmod new end old posn ndel nins ins (Fend+10000.
Fnew = old[1 to old length+nins-ndel].
(ins is vector->(ins length-1) null ins>(0) 1).
new = vecmod(new[1 to old length+nins-ndel],
new[1 to nins] ins).
new[1 to ndel] = old[posn+ndel to end].

'th75. f1.
Vecmod makes a copy of old vector with ndel elements deleted
beginning at posn. Ins is a vector, its elements are inserted
at the same place. It is the heart of edit.
'th60. f0.'

to addto func v w (/:func. :w. Fv+GET func FDO. null v->(error F(no code))
PUT func FDO vecmod v v length 0 w)
'th75. f1.
Addto appends code to a class definition.
'th60. f0.'

to fill t i l str (  
Ft = :str length.
Ft = disp + kbd.
(t = 10->
(Ft = disp + kbd)).
str[Ft+1] + t.
repeat
(i = 1->done)
10 = str[Ft+1] + disp + kbd->(done)).

'tstr)

to stream in : i e l(  
CODE 22
'th75. f1.
CODE 22 is equivalent to...

<e->
(  
(i = 1->
(F<str[i + 1] + 2 * i]))
<str[i + 1] + i + 1]
<next>
(i = 1->($0)
<str[i + 1] + i + 1]
<contents>
($1 to i])
'th60. f0.'
<next>
($i + 0)
isnew=True
<empty
<str[i + 1] +
((оф():
    string 10).
    $i$ +
    (фото((0):
        - 1)
    0).
    $i$ +
    (фото((s length))
    ист (ISIT eval)
    конца
    ($i = 1$
    печать
    
    (i $>$ 0?
     (s[1 to i] print)).
    disp = 1.
    1 $<$ i + 1?
     (s[1 to i] print))
    to (set (фото+stream of vector 10. repeat(?
        set : (??set contents)
        set + :)) 'т1.uses stream to accumulate a vector*т0.'
    to indisp disp (дисп. ??eval) 'т1.redefines disp and evals a vector*т0.'
    to string of x (фото+..??indisp stream (x print. disp contents))
    'т9.т1.
    uses stream and indisp to give you the print-string of anything.
    т9.т0.'
    to obset i input : vec size end : each (:
        add=((size=фото+end+1>(фото+vec+vec[1 to фото+size+size+10]))
        vec[end] ;)
        $i$ = (0=vec[1 to end] find first :input> (SELF add input))
        delete=(0=vec[1 to end] find first :input> (фото+false)
        vec[i to end]+vec[i+1 to end+1]. фото+end-end-1)
        unadd=(фото+input+vec[end], vec[end] ; nil,
        фото+end-end-1. ??input)
        vec=(??vec[1 to end])
        map=(фото+input+..??repeat (end=фото+и=0. фото+input+..??repeat (done=фото+и=0. фото+input eval) фоfalse)
        печать (SELF map фото+(each print, sp))
        ист (ISIT eval)
        именова=(фото+end+0. фото+vec+vector фото+size+4))
    )
    'т9.т2.
    PRETTY-PRINT
    'т9.т0.
    'т9.т1.
    This prints the code; classprint makes the header.
    т9.т0.'
    to show fun т (:
        $i$ = GET fun фото+DO.
        null т = (фото+(no code)) pshow т 0.)
    to pshow ptr length i t : x tabin index (:ptr :dent.
    (ptr length)$\times$(tabin dent)) disp+40.
    for i to ptr length-1
        (фото+т = ptr[i].
        т is vector> (pshow т dent+3.
        i = ptr length-15())
        фото+т = фото+часть ptr[i+1] )()
x is vector\(\rightarrow()\)
tabin dent)
i=1\(\rightarrow(t\ print)\)
\(0<\mathfrak{G}\times\text{index }\mathfrak{G}^*(.,\ 's'\ [\ ]\ \rightarrow)\)
\((x=1\rightarrow(t\ print.\ ptr[i+1]\ is\ vector\(\rightarrow()\\)\ t\ print)\)
t print)
\(0:\text{index }\mathfrak{G}(\mathfrak{G}^\#\ \uparrow[\ \mathfrak{G}\Rightarrow\Box]\ ptr[i-1]\rightarrow(disp+32.\ t\ print)\ t\ print)\)
\(\text{disp+41}\)
to t each tabin index (ev)
t
to each (\(\uparrow\text{vec[i]}\))  'shorthand for mapping with obsets'
PUT obset \(\mathfrak{G}\) each #each.
to tabin n :: x (:n. disp+13. repeat
\(n > 6\rightarrow\)
\(\text{disp + x}[6],\)
\(\mathfrak{G}^*n + n - 6)\)
done)
disp + x[n + 1])
(PUT tabin \(\mathfrak{G}\times\) (string 0 32 fill string 2 fill string 3
fill string 4 fill string 5 fill string 6)).
'leave these blanks'
PUT pshow \(\mathfrak{G}\) tabin #tabin.
to index op byte s lb ub s2 lb2 ub2 (\(\mathfrak{G}\)
\(\mathfrak{G}^*\text{byte},\mathfrak{G}^*\text{op+}\mathfrak{G}^*\text{lb+}\mathfrak{G}^*\text{s2+}\mathfrak{G}^*\text{lb2+}\mathfrak{G}^*\text{ub2+1, }\mathfrak{G}^*\text{ub=9999. CODE 40}\)
'\(\mathfrak{G}\)h75.tf1.\)
A piece of substr which runs faster.
'\(\mathfrak{G}\)h60.tf0.'
PUT pshow \(\mathfrak{G}\) index #index.
done
'\(\mathfrak{G}\)h90.tf2.
FLOATING POINT+\(\mathfrak{G}\)h60.tf0.
'
PUT float \(\mathfrak{G}\) DO \(\mathfrak{G}\)(0 CODE 42 'this does + - * / < > \leq \geq '
\(\mathfrak{G}\)ipart\(\rightarrow(1\ CODE\ 42)\)
\(\mathfrak{G}\)fpart\(\rightarrow(2\ CODE\ 42)\)
\(\mathfrak{G}\)ipow\(\rightarrow\)
\((x = 0\rightarrow(\mathfrak{G}\)
x = 1\rightarrow(\)
x > 1\rightarrow
\((1 \times x \mod 2)\rightarrow\)
\((\mathfrak{G}^*\text{SELF} \ast (\mathfrak{G}^*\text{SELF})\)
ipow x / 2)\)
\(\mathfrak{G}^*\text{SELF} \ast (\mathfrak{G}^*\text{SELF})\)
\(\text{ipow x / 2})\)
\(\uparrow 1.0 /\ \mathfrak{G}^*\text{SELF}\ \text{ipow 0-x})\)
\(\mathfrak{G}\)epart\(\rightarrow\)
\((\mathfrak{G}^*y + 2 \times \mathfrak{G}^*\text{epart x} \times x)\)
\(\mathfrak{G}\)eval\(\rightarrow\)
\((\mathfrak{G}^*\text{isit})\)
\(\mathfrak{G}\)print\(\rightarrow\)
\((\mathfrak{G}^*\text{print - SELF})\)
\(\text{fprint}\)
\(\text{fprint SELF})\)
)
to t fprint (ev)
t
to fprint n p q s :: fuzz z (\(\uparrow\text{tf1.Normalize to }[1,10):f0.\)
(:n < 1⇒
    (Gp + -(10.0 / n)
        epart 10.0)
    Gp + n epart 10.0)
Gn + fuzz + n / 10.0 ipow p.
(n*10.0/Gp-p+1, Gn-n/10.0 'ugly fix for now'))
'tf1. Scientific or decimal+f0.'
(Gs + fuzz*2.
'4<p<6p(Gq + 0.5
    p < 0⇒
        (disp>=[1 to 1-p])
    Gs + s * 10.0 ipow p)
Gq-p. Gp=0)
'tf1. Now print (s suppresses trailing zeros)+f0.'
do 9
    (disp + 48 + n ipart.
    Gp + p - 1.
    Gn + 10.0 * n fpart.
    p < 0⇒
        (p = "1⇒(disp + 49))
    n < Gs + 10.0 * s (done))
    (p = "1⇒(disp + 48))
    q < 0⇒(disp=0145.
    q print))
PUT fprint Gfuzz 5.0 * 10.0 ipow "9.
PUT fprint Gz fill string 4
0.00
PUT float Gfprint #fprint.
done

'th90.+f2.
TEXT DISPLAY ROUTINES
+th60.+f0.

+th75.+f1.
Display frames are declared with five parameters. They are a left x, a width, a top y, a heigh
**t, and a string. Hence --
G 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 2
**56 bits(raster units) wide and 256 bits high. The string (buf) serves as the text buffer, an
**d is altered by + and scrolling.

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 actual
**y used by the machine language will reduce the size of the window, if necessary, to be confi
**ned by the physical display. Clipping and scrolling are done on the basis of window boundar
**ies. If a character is in the window it will be displayed. If a string or character cause
**flow of the bottom of the window, scrolling will occur.

The four instance variables defining the frame are "frmx", "frmwd", "frmy", and "frmht". Thi
** rectangle may be smaller or larger than its associated window as well as the physical disp
**lay. Frame boundaries are the basis for word-wraparound. (Presently, if frm+ frmht will
**cause overflow of the window bottom[ winx+winht], frmht will get changed to a height consis
**nt with the bottom of the window. This has been done to manage scrolling, but may get change
**d 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 strin
**g 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 "e" 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, t
** the last character resulting from a "e".
"lastln" 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 "e" call.

"Mark" is set by dread (see below) and points to the character in the buffer which represents
the last 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
not 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 int
section 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 or a character to wide for a
frame -- hence no room for scan conversion of even one line of text. A 4 means that the fra
me height has been increased in order to accommodate the input. A 5 means the bottom of the w
indow (i.e. window x + window height) has been overflowed -- hence that scrolling took place.
A 6 means that both 4 and 5 are true.

"justify" is a toggle for right justifying the contents of a dispframe. The default is 0 an
"d means no justification. Setting it to 1 causes justification on frame boundaries.

The "font" 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 $somthing + file <fontfilename> conte
nts. Then you can say disp ($)font+something) or you can declare the font at the same time a
"s the dispframe is declared as e.g.

$yourframe + dispframe 3 40 3 40 string 20 font something.

(to dispframe input
: winx winwd winht frmx frmwd frmy frmht
last mark lastln chax chary reply justify buff font editor
: sub frame dread reread defont ()

$ -> (0 CODE 51)
' &h75. + f1.
.ss s is number -> (append this ascii char)
.s s is string -> (append string)
.error.
' &h60. + f0.'

$'eval
' &h75. + f1.
Allows access to instance variables. For example,
yourframe's ($frmx+32)
will alter the value of window x in the instance of dispframe
called $yourframe.
' &h60. + f0.'

$hasmouse ($frmx+nx $frmx+frmwd+ ($frmy+my+$frmy+frmht) +false)
' &h75. + f1.
Tells you if the mouse is within a frame.
' &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 begining through last. A handy
way to clean up a cluttered world.
' &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" on that frame.
'th60.+f0.'

\(\text{hide}\) (4 CODE 51 frame white)

\(\text{put}\) (input, (at, \(\text{G}^{-}\text{w}^{-}\text{win}^{-}\text{x}^{-}\text{=}\text{G}^{-}\text{f}^{-}\text{rm}^{-}\text{x}^{-}\text{=}\text{G}^{-}\text{w}^{-}\text{in}^{-}\text{y}^{-}\text{=}\text{G}^{-}\text{f}^{-}\text{r}^{-}\text{m}^{-}\text{y}^{-}\text{=}\text{G}^{-}\text{c}^{-}\text{h}^{-}\text{a}^{-}\text{r}^{-}\text{y}^{-}\text{=}\text{G}^{-}\text{l}^{-}\text{a}^{-}\text{s}^{-}\text{t}^{-}\text{n}^{-}\text{=}0. \text{SELF}^{-}\text{+}\text{input}. \text{?}\text{char}^{-}\text{x}^{-}\text{=}\text{win}^{-}\text{x}}\)
'th75.+f1.
For them as would rather do it themselves.
'th60.+f0.'

\(\text{wclear}\) (5 CODE 51)
'th75.+f1.
Wclear clears the intersection of a window and the physical display.
'th60.+f0.'

\(\text{scroll}\) (2 CODE 51)
'th75.+f1.
Scroll removes the top line of text from the frame's string buffer,
and moves the text up one line.
'th60.+f0.'

\(\text{clear}\) (1 CODE 51)
'th75.+f1.
Clear does an fclear and sets the "last" pointer into the string buffer
to 0 and "lstrn" to 1. It has the effect of clearing out the string
buffer as well as clearing the frame area.
'th60.+f0.'

\(\text{mfindc}\) (7 CODE 51)
'th75.+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:
\(\text{myvec+yourframe mfindc mx my} \)
'th60.+f0.'

\(\text{mfindw}\) (8 CODE 51)
'th75.+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:
\(\text{myvec+yourframe mfindw mx my} \)
'th60.+f0.'

\(\text{mfindt}\) (6 CODE 51)
'th75.+f1.
Find token.
Takes two arguments -- \(x\) and \(y\) (typically msex and msey).
Returns vector:
vec[1] = token count, a la 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] = top 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--
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
	+t60.+f0.'

<read-- (^dread)
	't75.+f1.
Makes a code vector out of keyboard input. See read below.
	+t60.+f0.'

<reread-- (^rread :)
	't75.+f1.
Used by redo and fix. Goes back n(its argument), prompts and does
a read from there. See reread below.
	+t60.+f0.'

<input \_ sub :: SELF show. ^input)
	't75.+f1.
Evals its argument in a sub-window. Used by fix and shift-esc.
See sub below.
	+t60.+f0.'

<knows=> (ev)
	't75.+f1.
Whilst at the KEYBOARD, one can say
"yourframe knows(DOIT)"
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.
	+t60.+f0.'

<frame \_ (apply frame)
	't75.+f1.
Draws a border of the given color around the frame. E.g.,
yourframe frame -1.
	+t60.+f0.'

<is => (ISIT eval)

isnew 
(\c^winx:+frnx. \c^winwd:+frnwd. \c^charx+\c^winy:+frmy.
:frnt.\c^winht=682-winy. :buf. \c^lsthn=1.
\c^mark+\c^last-\c^charx\c^reply+\c^justify+0.
\c^font + (\c^font=>(input is string=>(input)defont)defont)
\c^noframe=>() frame black) ))

dispframe knows
to read t flag {
disp-20. \c^flag+false. \c^mark+last.
(null //DRIBBLE() DRIBBLE flush).
repeat (0500exp-\c^t+kbdx(,
t010>s(last<mark>=buf[last])
	't75.+f1.
Backspace only up to prompt.
	+t60.+f0.'
buf[last]=047=>(\c^flag+flag is false))
	't75.+f1.
Backspace out of string flips flag.
	+t60.+f0.'
t=036=>(flag=>() done)
	't75.+f1.
DOIT checks if in a string.

\[ t=0 \rightarrow (\text{flag is false}) \]
\[ t=1 \rightarrow \text{true} \]

Flag is true if in a string

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

Font is defined in the string.

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

Shift-Esc make sub-eval.

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

Disp-13. The read of stream of buf from mark+1 to last.

to sub disp (disp disp frame frmx+48 frmy+64 frmht-28 string 300
text font disp clear. (display) eval)

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

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

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

to frame a (a + turtle at frmx - 1 frmy - 1.
a'width + 2. a'sink <white> 1 <black> 3 <color> 3 )
do 2 (a turn 90 go frmy+12 turn 90 go frmht+2)

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

draws a double line around the frame.

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

to reread n i p reader ((n = 0) (n = 1))
p=mark. p = mark-2. disp show.

\[ \text{for i to n} \]
\[ (p = buf[1 to p-1] find last 20. \]
\[ p<1>(\text{done}) \]
\[ i<n+1>(\text{error disp(no code)}) \]
\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

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

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

there. Also erases the line just typed.

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

done

to dclear (CODE 52)

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

This function takes five parameters -- x width y height value, and "clears" the display rectangle thus defined to the "value" given. A 0 value, for example, puts all zeros into the rectangle.

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

to dcomp (CODE 53)

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

Just like dclear only complement rectangle.

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

to dmove (CODE 54)

\[ t=0 \rightarrow \text{false} \]
\[ t=1 \rightarrow \text{true} \]

This function takes seven parameters -- source x width source y height destination x destination y mode. 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 overlapping portion of the destination wins. The mode is either store or or. Store (clearing destination area before scanning in source) is indicated with a mode 0. Or (which lays the source bits on top of whatever is in the destination rectangle) is indicated by mode - anything but zero.
to dmove (CODE 55)
  'th75.tf1.
Dmove 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 (dmove winx winy
winy+fontheight winht+fontheight winx winy 0). A sample call --
dmove 0 256 0 256 256 256 0.
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.
  th60.tf0.'

to redo (??(disp reread :) eval)
  'th75.tf1.
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.
  th60.tf0.'

to fix vec (G-vec-disp reread :)
(disp sub G(vecd vec)) eval)
  'th75.tf1.
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.
  th60.tf0.'

'th90.tf2.
TURTLES
  th60.tf0.'

to turtle var : pen ink width dir x xf y yf frame : f (CODE 21
  'th75.tf1.
CODE 21 is equivalent to:
<go>(draw a line of length :)
<turn>(turn right : (degrees))
<goto>(draw a line to :x, :y)
  th60.tf0.'
<var>(Gvar + 2. G->(??var + :)
  var eval)
<pendn>(Gpen + 1. ??SELF)
<penup>(Gpen + 0. ??SELF)
<black>(Gink + "3. ??SELF)
<white>(Gink + "1. ??SELF)
<xor>(Gink + "2. ??SELF)
<is>(#SELF eval)
<homes>(Gx + frame % frmwd/2.
  Gy + frame % frmht/2.
  Gxf + Gyf + 0. Gdir=270. ??SELF)
<erase>(frame fclear. ??SELF)
<up>(Gdir + 270. ??SELF)
<isnewa>(Gink + "3. Gpen + Gwidth + 1.
  Gframe + ("@frame.(:) f).
  1ota(x, y, Gxf + Gyf + 0. Gdir=270)
SELF home)
)
PUT turtle Gf dispframe 0 512 0 684 string 1 noframe.
Gturtle.

'th90.tf2.
THE TRUTH ABOUT FILES AND DIRECTORIES
a file is found in a directory ("dirinst") by its file name ("fname"), and has a one "page", 5
**12 character string ("sadr")). "rvee" is an optional vector of disk addresses used for rando
**m page access.

- **- <directory> file <string> old -- finds an old file named <string> in <directory> or returns f
> **alse if does not exist or a disk error occurs.

+ **<directory> file <string> new -- creates a new file or returns false if it already exists. i
> **f neither old or new is specified, an existing file named <string> will be found or a new fi
> **le 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 y
**ou create.

<directory> file <string> rename <string> -- renames file named by first string in <directory>
> ** with second string. currently not implemented for directory files.

<directory> file <string> load -- loads a previously "saved" memory image (Swat format), there
** by destroying your current state.

<directory> file <string> save -- saves your Smalltalk 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" = 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 unl
**less (s)he deals directly with the label or "sadr". it might be noted here that multiple inst
**nces of the same file do not know of each others activities or "sadr"s.

"status" is normally 0, -1 if end occurred with the last "set"; a positive number (machine lan
**guage pointer to offending disk command block (dcb)) signals a disk error.

the next 8 integers are the alto disk label block. "nextp" and "backp" are the forward and ba
**ckward alto address pointers. "unused" 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 0, although all files po
**ssess page 0 (the "leader" page). "version" numbers > 1 are not implemented. "sn1" and "sn2"
** are the unique 2-word serial number for the file.

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

(th60.+f0.)

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

((<word> (17 CODE 50)
  `(th75.+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-bytec+1>
     (SELF set to write (pagen+bytec/512 bytec mod 512))
     sadr<bytec+x [1] 0377
     th60.+f0.)

((<word> (7)
  `(th75.+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.t0.'

6)
"!h75.t1.
fi next word -- read an integer
(0=bytec \ 1 (() \ bytec-bytec+1)
!h(SELF next*256) + SELF next
\!h60.t0.'

\!intos (16)
"!h75.t1.
fi next into <string> -- read a string
for i to :x length(x[i]+SELF next).fx
\!h60.t0.'

25) CODE 50)
"!h75.t1.
fi next -- read a character
(nunch<\!bytec-bytec+1>
(SELF set to read (pagen-bytec/512)
-bytec mod 512 (:) ?0)) \!sadrf(bytec]
\!h60.t0.'

\!seto (\!to. (\!end(13)
"!h75.t1.
fi set to end -- set file pointer to end of file.
SELF set to read 037777 0
\!h60.t0.'

\!write(5)
"!h75.t1.
fi set to write <integer> <integer> -- set file
pointer to :page :char, if current page is dirty,
or "reset", "set to end" or page change
occurs, flush current page, read pages until
pages=page, allocate new pages after end if
necessary (-1 512 is treated as start of next
page, i.e. pagen+1 0), \!bytec=schar
\!h60.t0.'

\!read. 4) CODE 50)
"!h75.t1.
same as "write" except stop at end
\!h60.t0.'

\!skipnext (18 CODE 50)
"!h75.t1.
fi skipnext <integer> -- set character pointer relative to
current position. (useful for skipping rather than reading,
or for reading and backing up, but "end" may not work if
"bytec" points off the current page) \!bytec-bytec+:
\!h60.t0.'

\!end (10 CODE 50)
"!h75.t1.
fi end -- return false if end of file has not occurred.
nextp=0 \!nunch<(!false)) \!false
\!h60.t0.'

\!s. (\!s Eval)

\!flush (12 CODE 50)
"!h75.t1.
fi flush -- dirty=0 \!() write current page
\!h60.t0.'

\!writeseq (22 CODE 50)
"!h75.t1.
transfer words from memory to a file
:adr. :count. for i=adr to adr+count-1
(SELF next word + mem i)
+th60.+f0.'

(define) (21 CODE 50)
'+th75.+f1.
...from a file to memory...(mem i + SELF next word)
+th60.+f0.'

(define) (ISIT eval)

(define) (dirinst forget SELF)
'+th75.+f1.
remove file from filesopen list of directory
+th60.+f0.'

(define) (dirinst 's bitinst flush.
SELF flush. SELF remove. #closed)
'+th75.+f1.
fi close or #fi:fi close (if fi is global) -- flush bittable
and current page, remove instance from filesopen list of
directory
+th60.+f0.'

(define) (to. (SELF shorten pagen bytec) 14 CODE 50)
'+th75.+f1.
fi shorten to <integer> <integer> -- shorten a file SELF set
to read :spage :schar. #x:nextp. #x:nextp=0.
#numch=schar. #dirty=1. deallocate x and successors
+th60.+f0.'

(define) (disp 'fname)
'+th75.+f1.
file prints its name
+th60.+f0.'

(define) (11 CODE 50)
'+th75.+f1.
fi reset -- reposition to beginning of file
SELF set 1 0
+th60.+f0.'

(define) (SELF set to end.
#x = string bytec + 512 * pagen - 1.
SELF reset. #SELF next into x)

(define) (SELF set to end. #rvec + vector pagen.
for x to rvec length (SELF set x 0. rvec[x] = curadr))
'+th75.+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
+th60.+f0.'

(define) (20 CODE 50)
'+th75.+f1.
fi pages <integer> ... <integer> -- out of the same great
tradition as "mem" comes the power to do potentially
catastrophic direct disk i/o (not for the faint-hearted).
coreaddress. diskaddress. diskcommand. startpage.
numberofpages. coreincrement. if -1 = coreaddress,
copy "sad" to a buffer before the i/o call. diskaddress
(=1 yields "curadr") 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", copy label block from
instance. perform i/o call. copy "curadr" and label block
into instance. If -1=coreaddress copy buffer to G*sadrG*.

isnew  (G fname=nccheck :: fname is false)
       (error G (bad file name) nil)
     (null G dirinst + #curdir)
     (G dirinst + directory 's defdir. dirinst open).
      'th75.tlf1.
    set directory instance for file. If curdir is nil
    because file was not called from the context of a
directory instance, use the default directory
     'th60.tfo.'

<exists  (24 CODE 50.  #fname)
    'th75.tlf1.
  return false if file name does not occur in the
directory
     'th60.tfo.'

<delete  (15 CODE 50.  #deleted)
    'th75.tlf1.
  delete a file (see intro)
     'th60.tfo.'

G* sadr + (using (:) string 512).
    'th75.tlf1.
  set up file string buffer
     'th60.tfo.'

<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.  #fname)
     'th75.tlf1.
  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
     'th60.tfo.'

<load  (2 CODE 50.  8 CODE 50)
    'th75.tlf1.
  lookup an old file and load (overlay) a Swat
memory image; return via save.
     'th60.tfo.'

(<load  (2)
  <new  (dirinst 's filinst is file) (3) 19)
1) CODE 50.
    'th75.tlf1.
  find an old file or add a new entry (updating
create/write/read date and time, and file name
(as a Bpl string) in its leader page, special
handling for new directories), machine code
may return false
     'th60.tfo.'

<save  (SELF set to write 256 0.  SELF reset.
dirinst close.  9 CODE 50)
    'th75.tlf1.
  allocate a file, close the directory (other files
e.g. DRIBBLE, and directories should be already
closed), and write out the memory image as a
Swat file. when arriving here from a "load";
return false; otherwise return the file instance.
     'th60.tfo.'
<&{intostring=,#SELF intostring
    dirinst remember SELF })
  'th75.tf1.
  finally, file puts itself into the filesopen list of its
directory
  th60.tf0.'

file 's(ev)
to ncheck str i x :: legal (str < length < 255 ?false ?false)
    for i to str length
        (x = str[i].
         0140 < x < 0173 = ('lowercase')
         057 < x < 072 = ('digit')
         0 < legal[1 to 6] find x = ('legal')
         0100 < x < 0133 = ('uppercase')
         ?false)
    x=056 (?str ?str+ .chars)
  'th75.tf1.
  check that the file name is a proper length string containing only lower/upper case letters, d
**igits, or legal characters. if name does not end with a period, append one.
  th60.tf0.'

PUT ncheck G=legal fill string 6
  
  done

  'th75.tf1.
a directory is found in a directory ("dirinst"), has a bittable file ("bitinst") for allocatin
**g new pages, a file of file entries ("filinst" -- file names, disk addresses etc.), and a li
**st of currently open files ("filesopen" which is an "obset"). each file must ask its directo
**ry for the bittable when page allocation is necessary, and the system directory (via its loc
**al directory) for the disk number.

<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 calle
**d 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 be a directory, i.e. file was not called in
**the context of a directory, but globally
  th60.tf0.'

(to directory name exp: dirinst bitinst filinst filesopen: dirname bitname
curdir defdir ()
  <file> (SELF open. #apply file)
  'th75.tf1.
  di file <string>... -- open directory, create file instance
  (see file intro)
  th60.tf0.'

<open> (<curdir = SELF. filinst is file>)
  bitinst=filinst=filinst-old. <bitinst=filinst old>
  filinst = (new=filinst new) file filinst old.
  bitinst + dirinst's bitinst. dirinst remember SELF)
  'th75.tf1.
  di open -- (normally not user-called since access to the
directory always reopens it) initialize directory file and
bittable instances: a "subdirectory" uses the bittable of
its system directory and puts itself into that filesopen list.
  th60.tf0.'

<is> (ISIT eval)

<remember> (filesopen + ;)
<forget> (fileopen delete :)
	'rh75.tf1.
	add or delete file instances in fileopen.
	'rh60.tf0.'

<print> (disp+0133. fileopen print. disp=0135)
	'rh75.tf1.
	di or di print. --print the fileopen list.
	'rh60.tf0.'

<map> (SELF open. Gexp => . filinst reset.
	repeat (filinst end> (cr. done)
	1024 > Gname => filinst next word>
	(name < 2> () filinst skipnext 2*name-1)
	filinst skipnext 10.
	Gname => filinst next into string filinst next.

ter exp eval))
	'rh75.tf1.
	di map <expression> -- evaluate an expression for each

ter file name
	'rh60.tf0.'

<list> (SELF map Gname (disp+name. sp))
	'rh75.tf1.
	di list -- print the entry names contained in filinst
	'rh60.tf0.'

<flush> (fileopen map Gname (each flush))

<close> ((filinst is file> (SELF flush. Gfileopen + obset.
	Gfilinst + filinst's name.

dirinst is directory> (dirinst forget SELF. Gbitinst + false)
	Gbitinst + bitinst's name)). Gclosed)
	'rh75.tf1.
	di close (e.g. dp0 close) or Gdi-di close (to release

ter instance) --close a directory by closing all files and

directories in its fileopen list and deleting it from the

ter fileopen list of its directory. this is currently one way

ter to regain space by closing unwanted file instances, and

to change disk packs.
	'rh60.tf0.'

<use> (Gdefdir + SELF)
	'rh75.tf1.
	di use -- change the default directory.
	'rh60.tf0.'

<eval> (G$ eval)

<free> (SELF open. Gexp + 0.

terbitinst set to 1 609*dirinst 21.

do 609 (355 = Gname + bitinst next> (Gexp + exp + 8)
	name > 0> (repeat (Gexp + exp + name 21.
	0 = Gname + name 21> (done))))).
	'1872 - exp
	'rh75.tf1.
	di free -- return number of free pages from the

ter bittable associated with this directory.
	'rh60.tf0.'

<directory> (SELF open. @apply directory)
	'rh75.tf1.
	di directory <string>... -- open directory. create directory

ter instance
	'rh60.tf0.'

isnew (Gfileopen + obset.
	<device>Gdirinst + . Gfilinst + dirname.
directory *s(ev)
"
G=dirname * fill string 7
SYSDIR.
G=bitname * fill string 9
SYS.STAT.
done
'rh75.tf1.
names of the system directory and bittable
th60.tf0.'

dp0 + directory device 0.
dp1 + directory device 1.
dp2 + directory device 2.
dp3 + directory device 3.
dp0 use
'rh75.tf1.
cREATE some system directory instances which are initially closed.
dp0 is the same (default) directory that the Operating System
believes in.
dp1 can similarly access another Model 31 disk or a Model 44.
dp2 and dp3 provide a mechanism for writing on the other halves of
a Model 44, with some preliminary initialization; files written
through dp2 (dp3) can be read by dp0 (dp1) also. you should only
overwrite or delete such files via the directory that created them.--
one useful exception is that dp0 or the Operating System (e.g.
subsystems like CONCAT and ETHERMCA) can overwrite files in dp2
as long as the files remain the same length (e.g. Swat files).
'th60.tf0.'
to error adr ptr arec :: c sub {
(0=adr->mem 0102->(G=knows->ev $)) dson. :ptr))
G=arec->leech ARC.
disp sub G=((0=adr->(ptr print)
mem 0102->. disp=0377$) mem adr.
for adr=adr+1 to adr+(mem adr)2*9 (G
G'=ptr=mem adr.
disp=ptr$3. disp=ptr$ 0377))
cr c ev))
error knows
to c class code 11 12 adr vadr i cpe (null arec[5]..(.) G=arec->leech arec[5]. G=class->arec[0].
(GET class $ TITLE) print. G$: print.
G=vadr + G' vadr + arec[1]$)
G=11 + leech nil. G=12 + leech 11. 'Fasten seat belts.'
repeat (12$1$3 vadr.
05330=01777750[0]0[6]X(done) 'finds start of code vector'
G=vadr + vadr-1)
G=code + 12$1$.12$1$3 * 1.
G'=cpe + adr=vadr+1.
for i to code length
(i=cpe$5$ (disp=050) i$cpe+5$ (disp=056)
sp. (i=cpe$ (disp=031))
code[i] is vector (G (print) code[i] print).
}
to sub disp (G=disp + GET USER G=disp. ($) eval)
done
to kbck (1 CODE 20)
   'th75.tf1.
   Returns true if the keyboard has been hit.
   'th60.tf0.'

to button n (نك mouse 7)
   'th75.tf1.
   Returns true if that pattern is being held down
   'th60.tf0.'

'rh90.tf2.
THE SMALLTALK EDITOR ---
'rh60.tf0.'

to edit func t (一号func.
   $t GET func $DO.
   null t \(\) (no code)
   <title> (evapply veed classprint func header to $eval in GLOB)
   PUT func $DO veed t.
   $edited
   'th75.tf1.
   Edit picks up a code vector, makes sure it is not empty and calls
   veed to edit the code body. If you say edit foo title, veed will
   edit the header as well, and the changed form will be evalled upon
   exit to redefine the function, title and all.

   Veed 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.
   'th60.tf0.'

(to veed back newdisp menu x :: menuwidth menulen menustr
ed ed push edtarget gettwo begin getveec (}
   <knocks.ev>
   $back=false.
   disp fclear.
   disp's ($menu-dispframe frmx+frmwdf-menuwidth menuwidth
   frmy (frmht>139>(frmht) 140) string 70
   font font.
   menu + menustr.
   men 0425 + frmy + 103.
   $newdisp + dispframe frmx frmwdf-menuwidth+2
   frmy frmht string buf length
   font font noframe)
   x. $x = indisp newdisp (ed x).
   menu hide. disp display.
   $x )

veed knows

$menuwidth + 64.
$menustr=string 0.
$menulen + 10.
do menulen ({$x+}fill string 9.
   $menustr=menustr+x[1 to x[1 to 9]find 13]).
Add
Insert
Replace
Delete
Move
Up
Push
Enter
Leave
Exit

to ed ptr ln nnrun command temp i nv n1 fnth hfnth ( 
   $\text{command += 0.}$
   ;ptr.
   $\text{fnth += leech disp'sfont}[2]$)
   $\text{hfnth += fnth}/2$. 
   repeat(
      \text{l-ptr length.}
      backa(done with ptr)
      mem 0424 + menu's frmx + 48.
      menu show. disp clear
      $\text{nv += 0.}$
      for n to l-1
         (ptr[n] is vector (disp=044. sp
         $\text{nv += nv+1.}$ $\text{n1+n}$)
         ptr[n] print. disp=32)
       cr or.
       command += edcomp bugin menu menulen both. 
       mem 0424 + disp's (frmx + frmwd/2).
   )

) 'th75.tfl.
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.tfl.'

to edpush ins ($\text{ins+vector list.}$
   $\text{ptr+vecmid ptr n nnrun ins}$)

to gettwo t1 n2 ($\text{n-cdecomp edtarget top.}$
   $\text{n2+edcomp edtarget bot.}$
   $\text{nnrun + 1+n2-n.}$
   $\text{nnrun<1+(n+n2. nnrun+n2))}$

to bugin someframe max index( 
   $\text{someframe += .}$
   $\text{max += .}$
   repeat (button 0 = (repeat ( 
      button 7 = (disp sub $\text{(ev)}$)
      button 0 = ()
      done)
   )
)
index=somedefault mx my.
0 <= index[1] <= max

'th75.rfl.
returns token index, if within range, else
'th60.rf0.'
again
'th75.rfl.
causes an exit of this command by restarting ed's
repeat
'th60.rf0.'
)

to edtarget (# begin disp 1)

to getvec (nv=1>(G^n+n1. #true)
#ptr[G^n-edcomp edtarget both] is vector)

to edcomp compvec y hth (:compvec,
G^y=compvec[4].
G^hth=(<both)=(fnth)<top=(hfnth)
<bot=(G^y+y+hfnth. hfnth))
#compvec[1]
)
done

'th90.rf2.
BOOTSTRAPPING REVISITED
'th60.rf0.'
to classprint fn a b i j k flags clsv clsm arecv arecmm instv instm code (:
#:fn. G^code + GET fn G^DO. null code=(G'(no code))
G^a+leech #fn. G^b-vector l. G^b-leech b. G^clsm=G^arecem+G^instm+0.
'th75.rfl.
Preliminary symbols out of class table
'th60.rf0.'
for i=4 to 4+2*k by 2
'th75.rfl.
k is no. dbl entries -1, here
'th60.rf0.'
(G^k=a[i]
'th75.rfl.
0=class, 2=arec, 3=inst
'th60.rf0.'
flags=0<=(G'(DO TITLE SIZE) [1 to 3] find a[i]>
 (clsv|(G^clsm+clsm+1) + a[i]))
b[2]=k[3777. G^j+a[i+1]]
(flags=2=(arecv[j-6] = b[2]. arecmm[j-6]=(G^arecem+j-6))
instv[j]+1 = b[2]. instm[j]+1=(G'(instm+j+1))
)
'th75.rfl.
Now make up input form.
'th60.rf0.'
G^a + vector 6+arecem+instm+clsm.
a[3 to G^j+2+arecem] + arecv.
(0<instm+clsm) (a[G^j+j+1]=G^:. a[j+1 to G^j+instm] + instv.
<clsm) (a[G^j+j+1]=G^:. a[j+1 to G^j+clsm] + clsv)))
<header>(a[j+1]=code. #a)
for i to j (a[i] print. disp=32)
showpretty>(#show code 3) code print

to show showpretty (G^showpretty=true, showev $)
to showev shAtom shVal (shAtom. cr.
(shAtom is atom\$
  (\$shVal = shAtom eval.
  (null GET shVal \$DO\$
   \$print. shAtom print. \$e print.
   (shVal is vector\$ \$e print)
   (shVal is atom\$ \$e print)
   (null shVal\$ \$nil print))
  shVal print. \$e print)
class print shVal))
shAtom print)
disp=30.)
*t75.\$f1.
*****Keyboard translation*****
*t60.\$f0.

to kbd (3kmap[TTY])
\$kmap = string 0377.
  for i=0000 to 0377(kmap[i] = 0177) 'tf1.ILLEGAL GETS \$f0.'
  for i=001 to 0177(kmap[i] + i) 'tf1.1.\$f1.\$f1.\$f0.'
  'tf1.\$f1.\$f0.'
  'tf1.\$f0.'
  'tf1.\$f1.\$f0.'
\$kmap[0343]=kmap[0303]=kmap[0272]=03. 'tf1.\$f0.'
\$kmap[0344]=kmap[0304]=04. 'tf1.\$f0.'
\$kmap[0235]=kmap[0305]=kmap[023]=05. 'tf1.\$f0.'
\$kmap[0345]=kmap[0306]=kmap[0262]=06. 'tf1.\$f0.'
\$kmap[0347]=kmap[0307]=kmap[0273]=07. 'tf1.\$f0.'
\$kmap[0353]=kmap[0313]=kmap[0245]=013. 'tf1.\$f0.'
\$kmap[0356]=kmap[0275]=016. 'tf1.\$f0.'
\$kmap[0357]=kmap[0317]=kmap[0242]=017. 'tf1.\$f0.'
\$kmap[0360]=kmap[020]=kmap[0271]=020. 'tf1.\$f0.'
\$kmap[0361]=kmap[0321]=kmap[0261]=021. 'tf1.\$f0.'
\$kmap[0362]=kmap[0322]=kmap[0300]=022. 'tf1.\$f0.'
\$kmap[0363]=kmap[0323]=023. 'tf1.\$f0.'
\$kmap[0346]=kmap[0324]=024. 'tf1.\$f0.'
\$kmap[035]=kmap[0325]=kmap[0255]=kmap[0140]=025. 'tf1.\$f0.'
\$kmap[0366]=kmap[0326]=kmap[0265]=026. 'tf1.\$f0.'
\$kmap[0367]=kmap[0327]=kmap[0376]=027. 'tf1.\$f0.'
\$kmap[0370]=kmap[0330]=kmap[0246]=030. 'tf1.\$f0.'
\$kmap[0371]=kmap[0331]=kmap[0277]=031. 'tf1.\$f0.'
\$kmap[0372]=kmap[0332]=kmap[0276]=032. 'tf1.\$f0.'
\$kmap[0333]=kmap[0264]=033. 'tf1.\$f0.'
\$kmap[0334]=kmap[0267]=034. 'tf1.\$f0.'
\$kmap[0335]=kmap[0375]=035. 'tf1.\$f0.'
  'tf1.\$f0.'
\$kmap[0247]=0174. 'tf1.\$f0.'
\$kmap[0257]=0176. 'tf1.\$f0.'
\$kmap[0263]=043. 'tf1.\$f0.'
\$kmap[0270]=052. 'tf1.\$f0.'
\$kmap[0220]=kmap[0210]=kmap[020]=010. 'tf1.\$f0.'
\$kmap[0245]=kmap[0215]=kmap[025]=015. 'tf1.\$f0.'
\$kmap[0240]=kmap[0230]=kmap[030]=040. 'tf1.\$f0.'
  to filout disp list i show pretty (\$show pretty = \$show pretty + \$show pretty)
  disp off (disp is string) (\$disp=file disp \$0 error \$e (file error))
  (\$add=(disp set to end))
  (null :list->(defn map \$show each, cr))
  for i to filist length-1 (show vec filist[i], cr)
  disp shorten to here. disp close. \$dson.)

  'tf1.\$f0.'...Filout basically does a show in a context where the display is replaced by a file. filout pretty <file> or <string = file name> add <vector>. if "pretty" is used, the text representation is neater but
takes longer to generate. If "add" is used, function definitions are
appended to the file. If <vector> is not specified, "defs" is used.
\texttt{\textbackslash th60.\textbackslash tf0.}'

to filin fi :: ev (\texttt{\textless s\textgreater}(\texttt{\textbackslash 8\textbackslash eval})
dsoff.
  (\texttt{\textless fi is string\textgreater})(\texttt{\textbackslash f i - file fi old\textbackslash s()}
    dson \texttt{\textbackslash false}))
repea
  (\texttt{\textless fi end\textbackslash s\textgreater}(\texttt{\textbackslash done})
dsoff.
   cr (\texttt{\textbackslash read of fi\textbackslash s} eval print.
    dson).
  fi close.
)
filin's (to ev (repea(\texttt{\textless cr\textbackslash s\textgreater\textless read of fi\textbackslash s\textgreater\textless eval\textbackslash s\textgreater))}

'\texttt{\textbackslash th75.\textbackslash tf1.}
Filin basically does a read-eval-print loop, but gets its input from a
file instead of a dispframe.
\texttt{\textbackslash th60.\textbackslash tf0.}'

to type f t ((\texttt{\textless f is string\textgreater})(
  \texttt{\textbackslash f i + file f old\textbackslash s(f remove)
    \textbackslash false}))
  \texttt{\textbackslash t+string 30.}
repea(f end\textbackslash s\textbackslash s\textgreater\textless done\textbackslash s\textgreater disp=f next into t))

to t fool :: fontname (dispframe 's (\texttt{\textless defont\textbackslash s\textgreater - file fontname intostring}).
  \texttt{\textless disp=dispframe 16 480 514 168 string 520.}
  disp = version. \texttt{\textless def=\textbackslash s\textgreater - obsat.}
  \texttt{\textless fool=\textbackslash t.o. to to toAtm (CODE 19 def=toAtm, toAtm)
    PUT USER \texttt{\textless \textbackslash dO \textbackslash t\textbackslash s\textgreater (cr read eval print). \texttt{\textless t=0.)}
    PUT t \texttt{\textless fontname\textbackslash s\textgreater fill string 6}
ST8.AL
\texttt{\textless version=\textbackslash s\textgreater fill string 34}
Welcome to SMALLTALK [September 9]

to expand x ((x. disp 's (\texttt{\textless winy=\textbackslash t\textbackslash s\textgreater\textless frmy=frmy-x, frame black)
    disp show CODE 38))
  'th75.\textbackslash tf1.}
t is called to set up a display frame, and def and then self-destructs
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)
\texttt{\textbackslash th60.\textbackslash tf0.}'

(to junta scanner :: read1 tablsan rdnum mknum rdstr rbh1 type
  lebit digit sbigit atbits qbit
  (\texttt{\textless s\textgreater}(\texttt{\textbackslash 8\textbackslash eval})
    \texttt{\textless of\textless s\textgreater((\texttt{\textless scanner is string\textgreater})
      (\texttt{\textless scanner + stream of scanner\textgreater})
    \texttt{\textless scanner + tablsan scanner type.}
    \texttt{\textless \textbackslash read1 rbh1)
    \texttt{\textless disp read))}
  junta \texttt{\textbackslash s(ev)}

to read1 rbbuf rdth flag (\texttt{\textbackslash rbth}.
  \texttt{\textless \textbackslash rbbuf - stream of vector 10.}
  scanner read.
  \texttt{\textless \textbackslash rbbuf contents)}

to tablsan mask : source type seq isfil nxtchr (
(CODE 14 next.
CODE 14 is equivalent to...
mask=0=(G^t-string 1. t[1]+nxtchr.
G^nxtchr+source next. #atom t)

seq reset.
repeat
(0 = nxtchr=done).
0 = mask # type[nxtchr + 1]=done).
seq = nxtchr.
(G^nxtchr + source next)
#seq contents')
#skip(=nxtchr + source next)
#read(= (repeat
(rdth[nxtchr + 1]=eval))
isnew=)
#source.
#type.
#G^nxtchr + stream.
(source is file=(G^isfil + 1))
SELF skip))

to rdnnum sign base n fs(
G^nxtchr=025=(scanner skip. *11).
G^nxtchr=060=(8)10.
G^n = mknnum scanner next digbit base.
G^flag = false.
056 = nxtchr=
(scanner skip.
G^fs = scanner next digbit
0=fs length=(G^flag=true. #sign*n)
fsn = n + (mknnum 10)/10.0 ipow fs length.
nxtchr=014=(scanner skip. #n*(10.0 ipow #rdnum)*sign)
#n^sign
#sign*n)

to mknnum str base n i(f
G^str.
G^base.
G^n = 0.0.
for i to str length
(G^n = (n^base) + str[i]-060)
#n)

to rdstr t(scanner skip.
G^t=scanner next qtbit.
scanner skip.
nxtchr=047=(seq+047. #seq contents+rdstr)
#t)

'rh75.tf1.INITIALIZATION OF READ TABLES

rh60.tf0.'
G^rtbl = vector 256.
G^type = string 256.
G^atbits = lebbit + digit

to scanner n i j (n. v. repeat (i.
(->to(j): for k=i+1 to j+1 (type[k]=n. rtbl[k]=v)
type[i+1]=n. rtbl[i+1]=v)
< and> () done)
scanner 0 C=(rbuf+scanner next 0) 0 to 0377.
scanner lethib C=(rbuf+atom scanner next atbits) 0101 to 0132 and 0141 to 0172. 'letters'
scanner digbit C=(rbuf+rdnum. flag+(rbuf+C\^)). 060 to 071. 'digits'
scanner 0 C=(rbuf+rdnum. flag+(rbuf+C\^)). 025. 'high-minus'
scanner sephib C=(scanner next sephib) 011 and 014 and 015 and 040. 'tab, LF, FF, CR, blank'
scanner qtbit C=(rbuf+rdstr) 017. 'string-quote'
scanner 0 (scanner skip. rbuf + (read1 rth1) eval) 020. 'eval-paren'
scanner 0 (scanner skip. rbuf + (read1 rth1) eval) 050. 'left-paren'
scanner 0 (scanner skip. rbuf + nil. done) 051. 'right-paren'
scanner 0 (rbuf + nil. done) 0 and 036. 'null and DOIT'
for i to type length (type[i] + type[i] @ qbit)
done
read + #junta.
PUT read G TITLE G read. 'th75.tf1.cover our tracks\th60.tf0.'

to quit f s t : : r b  (dsoff.
    (null $s$)
    $f$ = file r.
    $t$ = $f$ intostring.
    $f$ reset.
    $f$ + s.
    $f$ + 13.
    $f$ + t.
    $f$ close).
file b load
PUT quit G r fill string 7
REM.CM.
PUT quit G b fill string 5
BOOT.

to os s : : r b  (G s + :.
    file b save
    (quit s + r))
PUT os G b fill string 9
BREAK.SV.
PUT os G r fill string 17
;RESUME BREAK.SV.

$G$ fill + nil

to junta (PUT USER G DO G (t). CODE 31)
    'th75.tf1allocates display over OS after setting up th60.tf0.'