§ ALLDEFS §

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 □ ≤≟≠≥ min max)

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

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

TO EDIT AND PRINT ALLDEFS -----

In order to look at Alldefs in Bravo with a Smalltalk font and achieve printing with Smalltalk characters, fonts 0 and 1 on your user.cm must be as follows:

FONT:0 ST 8 SMALLTALK 14
FONT:1 TIMESROMAN 12 TIMESROMAN 12
etc.

⟨SMALLTALK⟩STUSER.CM can be copied onto user.cm and has the appropriate fonts 0 and 1 specified. SMALLTALK14.AL is on ⟨ALTOFONTS⟩.The following .EP files must be on your disk in order to get Bravo to HardCopy Alldefs:

ST8.EP
ST8I.EP
ST8B.EP
TIMESROMAN12B.EP

BOOTSTRAPPING MAGIC.

to isnew (null instance⇒(☞instance←allocate permsize.
instance[0]←class. ⇑true)
⇑false).

§

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

to read (CODE 2)
§
Read keyboard input into a vector. This is almost identical in function to the SMALLTALK read routine, except that DOIT is signalled by ⟨CR⟩ at zero-th parenthesis level, and single-quote strings are ignored. It is only available in Nova versions.
§

§
MESSAGE HANDLING
§

to : (CODE 18)
§
to : name

◀☞ ⇒(:☞name nil ⇒(⇑name←caller message quotefetch)
(⇑caller message quotefetch)

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

◀# ⇒(:☞name nil ⇒(⇑name←caller message referencefetch)
(⇑caller message referencefetch)

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

(:☞name nil ⇒(⇑name← caller message evalfetch)
⇑caller message evalfetch)

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

to ◀ (CODE 17)
§
:☞token. token=caller.message.code[caller.message.pc]⇒
(caller.message.pc←caller.message.pc+1. ⇑true) ⇑false.
That is, if a match for the token is found in the message, then gobble it up and return true, else return false.
§

to ♎ (CODE 39)

to ♒ (CODE 36)

to ⇑ (CODE 13)
§
:x. then do a return, and apply x to any further message. Note that in (... ⇑x+3. ☞y←y-2), the assignment to y will never happen, since ⇑ causes a return.
§

to ☞ (CODE 9)
§
⇑:☞. That is, get the next thing in the message stream unevalled and active return it (which causes it to be applied to the message).
§

to # (:#)
§
Returns a REFERENCE to its argument/s binding.
§

§
CONTROL CLASSES
§

to repeat token (:#token. CODE 1)
§
repeat (token eval) Not a true apply to eval, and therefore token MUST be a vector.
§

to done x (◀with⇒(:x. CODE 25) CODE 25)
§
done causes a pop out of the nearest enclosing repeat, for, or do. "done with val" will cause the repeat to have value val
§

to again (CODE 6)
§
repeat (☞active←active caller. eq active. class #repeat⇒(done)).
That is, redo the most recent repeat, for, or do loop.
§

```
to if exp (:exp⇒(◀then⇒(:exp. ◀else⇒(☉. exp)exp)error ☞(no then))
                ◀then⇒(☉.  ◀else⇒(:exp) false)error ☞(no then))
        §
        The ALGOL "if ... then ... else ..."
        §

to for token step stop var start exp (
        ☞var ← ☉.
        ☞start ← (◀←⇒(:) 1).
        ☞stop ← (◀to⇒(:) start).
        ☞step ← (◀by⇒(:) 1).
        ◀do. :#exp. CODE 24)
        §
        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.
        §

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

§

INITIALIZING SYSTEM CLASSES

Here are the main kludges which remain from the time when we really didn/t understand classes very well, but wanted a working SMALLTALK. PUT and GET are two of the principle actions of class class. The new verson of SMALLTALK will have class as a class with these actions intensional.

§

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

§

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

§

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

§

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

§

```
to leech field bits | ptr (CODE 27
        isnew⇒(:ptr))
```

§

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

§

```
PUT USER ☞TITLE ☞USER
PUT falseclass ☞TITLE ☞false

PUT atom ☞DO ☞(CODE 29
```

§

```
        ◀◀←⇒(:x.  ⇑x -- Lookup SELF and replace its value by x.)
        ◀◀◀eval⇒(⇑ -- Lookup the binding of SELF)
        ◀◀◀=⇒(⇑SELF=:)
        ◀◀◀chars⇒(⇑ -- printname of SELF (a string))
```

§

```
        ◀◀◀is⇒(ISIT eval)
        ◀◀◀swap⇒(☞x←SELF eval. SELF←:. ⇑x)
        ◀◀◀print⇒(disp←SELF chars) )
```

§

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

```
        followed by the behavior here.
        §

to ev (repeat (cr read eval print))

PUT falseclass ☞DO ☞(CODE 11
        §
        ⇒⇒ (:☞.)
        or⇒ (⇑:)
        and⇒ (:.)
        <⇒ (:.)
        =⇒ (:.)
        >⇒ (:.)
        §

        is⇒(false⇒(⇑true) ?⇒ (⇑☞false) §. ⇑SELF)
        print⇒(☞false print) )

PUT vector ☞DO ☞(CODE 3 ⇒(⇑substr SELF x GLOB MESS)
        §
        isnew⇒(Allocate vector of length :.
                Fill vector with nils.)
        [⇒(:x. ].
                (←⇒(:y. ⇑y -- store y into xth element. )
                ⇑ xth element) )
        length⇒(⇑ length of string or vector)
        eval⇒(☞pc←0. repeat
                (null SELF[☞pc←pc+1]⇒(done)
                ☞val←SELF[pc] eval)
                ⇑val) sort of...
        §
        is⇒(ISIT eval)
        +⇒(:y is vector⇒(☞x←SELF[1 to SELF length+y length].
                ⇑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)
        §
        isnew⇒(Allocate string of length :.
                Fill string with 0377s.)
        [⇒(:x. ].
                (←⇒(:y. ⇑y -- store y into xth element. )
                ⇑ xth element) )
        length⇒(⇑ length of string or vector)
        §

        is⇒(ISIT eval)
        =⇒(:y is string⇒(SELF length=y length⇒(
                for x to SELF length (SELF[x]=y[x]⇒() ⇑false)) ⇑false)
                ⇑false)
        +⇒(:y is string⇒(☞x←SELF[1 to SELF length+y length].
                ⇑x[SELF length+1 to x length]←y[1 to y length])
                error ☞(string not found))
        print⇒(0 = ☞x ← SELF[1 to 9999] find first 39⇒
                (disp ← 39. disp ← SELF. disp ← 39)
                SELF[1 to x-1] print. SELF[x+1 to SELF length] print)
        )

PUT number ☞DO ☞(CODE 4
        §
        +⇒(⇑val+:)
        -⇒(⇑val-:)
        *⇒(⇑val*:)
        /⇒(⇑val/:)
        <⇒(⇑val<:)
        =⇒(⇑val=:)
```

```
        ◀)⇒(⇑val):)
        ◀≤⇒(⇑val≤:)
        ◀*⇒(⇑val*:)
        ◀≥⇒(⇑val≥:)
        ◀max⇒(☞x←:. SELF)x⇒(⇑SELF) ⇑x)
        ◀min⇒(☞x←:. SELF(x⇒(⇑SELF) ⇑x)
        ◀□⇒(◀+⇒(⇑val OR :)
                ◀-⇒(⇑val XOR :)
                ◀*⇒(⇑val AND :)
                ◀/⇒(⇑val LSHIFT :))
        §

        ◀\⇒(⇑:x*SELF/x)
        ◀is⇒(ISIT eval)
        ◀print⇒(SELF)0⇒(nprint SELF)
                SELF=0⇒(disp←060)
                SELF=0100000⇒(disp←base8 SELF)
                disp←025. nprint 0-SELF) )
        §
        For floating point stuff see FLOAT
        §

to - x (:x*¯1)
        §
        An often used abbreviation, has to work for float as well.
        §

to base8 i x s (:x. ☞s←string 7. for i to 7
        (s[8-i] ← 060 + x ▣7. ☞x ← x ◰¯3). ⇑s)
        §
        Returns a string containing the octal representation (unsigned) of its integer argument.
        §

☞ISIT ← ☞(◀?⇒(⇑TITLE) ⇑TITLE=§).

to nil x (#x)
        §
        nil is an "unbound pointer", which is used to fill vectors and tables.
        §

to null x (:x. 1 CODE 37)
        §
        Null returns true if its message is "nil", otherwise false.
        §

to eq x (CODE 15)
        §
        (⇑:x is-identical-to :) - compare 2 SMALLTALK pointers.
        §
```

§

UTILITIES

§

```
to mem x y (:x. CODE 26)
        §
        to mem x y (:x. ◀←⇒(⇑core/mem x ←:)⇑core/mem x)
        mem loads integers from and stores them into real core.
        Tee hee...
        mem 0430 ← 0  --set alto clock to zero
        mem 0430  ;read the clock
        for i to 16 (mem 0430+i ← cursor[i])  --put new bits into cursor
        mem 0424 ← mem 0425 ← 0.  --reset mouse x and y to 0.
        mem 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.
        §
```

```
to mouse x (:x. CODE 35)
        §
        x = 0-7  are a map on the mouse buttons. E.g. (4=mouse 4) comes back true if the top mouse
        button is depressed, (1=mouse 1)) comes back true if bottom mouse button depressed, (7=mouse
        7)) comes back true if all three mouse buttons depressed, etc. Mouse 8 returns the x coordinate
        of the mouse, mouse 9 returns the y, and mouse 10 returns the x-y point.
        §

to mx (⇑mouse 8)
to my (⇑mouse 9)

to point t | x y (CODE 23
        §
        CODE 23 is equivalent to:
                isnew⇒(☞x←:. ☞y←:)
                ↞x⇒(↞←⇒(☞x←:) ⇑x)
                ↞y⇒(↞←⇒(☞y←:) ⇑y)
                ↞+⇒(☞t←:. ⇑point x + t x  y + t y)
                ↞-⇒(☞t←:. ⇑point x - t x  y - t y)
                ↞=⇒(☞t←:. x=t x⇒(⇑y=t y) ⇑false)
                ↞≤⇒(☞t←:. x≤t x⇒(⇑y≤t y) ⇑false)
        §
        ↞max⇒(☞t←:. ⇑point x max t x y max t y)
        ↞min⇒(☞t←:. ⇑point x min t x y min t y)
        ↞/⇒(☞t←:. ⇑point x/t y/t)
        ↞*⇒(☞t←:. ⇑point x*t y*t)
        ↞\⇒(☞t←:. ⇑point x\t y\t)
        ↞is⇒(ISIT eval)
        ↞print⇒(☞point print sp x print sp y print))

to mp (⇑mouse 10)

point 0 0. § establishes point class in machine code so that mp can use it §

to core (⇑(mem 077)-mem 076)
        §
        Returns the amount of space left in your Smalltalk.
        §

to kbd (0 CODE 20)
        §
        Waits until a key is struck. Returns an ascii code when a key is struck on the keyboard. Use to
        kbck (1 CODE 20) to return true if kbd has a character, otherwise false. Used in
        multiprocessing.
        §

to disp x i (
        ↞←⇒(:x is string⇒(for i to x length (TTY←x[i])) TTY←x)
        ↞clear⇒() ↞sub⇒(:x eval))
        §
        This disp is used for bootstrapping. Later in these definitions (READER)it will be restored to
        an instance of "display frame."
        §

to TTY (0 CODE 20)
        §
        TTY←⟨integer⟩ will print an ascii  on the Nova tty. On altos, TTY prints in little error window
        at bottom of screen.
        §

to dsoff (mem 272←0)
        §
        Turns display off by storing 0 in display control block ptr. Speeds up Alto Smalltalk by factor
        of 2.
        §

to dson (mem 0420 ← 072)
        §
        Turns display back on by refreshing display control block pointer.
        §
```

```
to apply x y  (:#x. ◀to⇒(:y. ◀in⇒(:GLOB. CODE 10) CODE 10)
                ◀in⇒(:GLOB. CODE 10) CODE 10)

to evapply x y (:x. ◀to⇒(:y. ◀in⇒(:GLOB. CODE 10) CODE 10)
                ◀in⇒(:GLOB. CODE 10) CODE 10)
        §
        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.
        §

to cr (disp←13).  to sp (disp←32)

☞true←☞true
☞eval←☞eval
to is ( ◀?⇒(⇑☞untyped) &. ⇑false)
        §
        These are used to handle messages to classes which can/t answer questions invoking "is",
        "eval", etc.
        §

to t nprint substr (ev). t
        §
        prevent -to- from making these global.
        §

to nprint digit n (:n=0⇒()
        ☞digit←n mod 10. nprint n/10. disp←060+digit)
PUT number ☞nprint #nprint.
        §
        Prints (non-neg) integers in decimal with leading zeroes suppressed!
        §

to substr op byte s lb ub s2 lb2 ub2 ins d1 d2 (:#s. :lb.
        :ub. :MESS. ☞GLOB ← ub.  § rest of args from above §
        :ub. (◀]⇒() error ☞(missing right bracket))
        ☞byte ← ☞lb2 ← ☞ub2 ← 1.
        ◀find⇒
                (☞op ← (◀first⇒(1) ◀last⇒(2) 1) + (◀non⇒(2) 0).
                 :byte. CODE 40)
        ◀↔⇒
                (◀all⇒(:byte. ☞op ← 0. CODE 40)  § fill const §
                 :#s2. ☞op ← 5.
                 ◀[⇒(:lb2. ◀to. :ub2. ◀]. CODE 40)  § substr copy §
                        ☞ub2 ← 9999. CODE 40)
        ◀replace⇒
                (☞ins←:. ☞d1←lb. ☞d2←ub.
                 ☞ub2 ← s length+ins length-1+d2-d1.  § new length §
                 ☞s2 ← (s is string⇒(string ub2) vector ub2).
                 ☞lb ← 1. ☞ub2 ← ☞ub ← d1-1.
                 ☞op ← 6. CODE 40.  § copy up to replacement §
                 ☞lb ← d2+1. ☞ub ← s length.
                 ☞lb2 ← d1+ins length. ☞ub2 ← s2 length.
                 CODE 40.  § copy after the replacement §
                 ☞ub ← ins length. ub=0⇒(⇑s2) ☞s ← ins. ☞lb ← 1.
                 ☞lb2 ← d1. CODE 40)  § copy in the replacement §
        ☞op ← 6. ☞ub2 ← ub + 1 - lb.  § full copy §
        ☞s2 ← (s is string⇒(string ub2) vector ub2).
        CODE 40)
PUT string ☞substr #substr.
PUT vector ☞substr #substr.
done
        §
        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
        String syntax is identical.
```

```
        §

to fill t i l str (☞l ← :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
        §
        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])
        §
        ◀reset⇒(☞i ← 0)
        ◀'s⇒(⇑°eval)
        isnew⇒(☞s ←(◀of⇒(:) string 10).
                ☞i ← (◀from⇒((:) - 1) 0).
                ☞l ← (◀to⇒(:) s length))
        ◀is⇒(ISIT eval)
        ◀end⇒(⇑i = l)
        ◀print⇒((i > 0⇒(s[1 to i] print)). disp ← 1. l < i + 1⇒() s[i + 1 to l] print))

to { set (☞set←stream of vector 10. repeat(
        ◀}⇒(⇑set contents)
        set ← :))          § uses stream to accumulate a vector §

to indisp disp (:disp. ⇑°eval)   § redefines disp and evals a vector §

to stringof x (☞x←:. ⇑indisp stream (x print. disp contents))
        §
        uses stream and indisp to give you the print-string of anything.
        §

to obset i input | vec size end | each (
        ◀add⇒((size=☞end←end+1⇒(☞vec←vec[1 to ☞size←size+10]))
                vec[end]←:)
        ◀←⇒(0=vec[1 to end] find first :input⇒
                (SELF add input))
        ◀delete⇒(0=☞i←vec[1 to end] find first :input⇒(⇑false)
                vec[i to end]←vec[i+1 to end+1]. ☞end←end-1)
        ◀unadd⇒(☞input←vec[end]. vec[end]←nil.
                ☞end←end-1. ⇑input)
        ◀vec⇒(⇑vec[1 to end])
        ◀map⇒(☞i←0. ☞input←:. repeat
                (end<☞i←i+1⇒(done) input eval)⇑false)
        ◀print⇒(SELF map ☞(each print. sp))
        ◀is⇒(ISIT eval)
        isnew⇒(☞end←0. ☞vec←vector ☞size←4)
        )

to t each (ev)
t
to each (⇑vec[i])          § shorthand for mapping with obsets §
PUT obset ☞each #each.
done


§
FLOATING POINT
§

PUT float ☞DO ☞(0 CODE 42   § this does + - * / < = > ≤ ≠ ≥ §
        ◀ipart⇒(1 CODE 42)
        ◀fpart⇒(2 CODE 42)
        ◀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)
```

```
        epart⇒ (SELF < :x⇒(⇑0) SELF < x * x⇒(⇑1)
                                ⇑(☞y ← 2 * SELF epart x * x) +
                                (SELF / x ipow y) epart x)
        is⇒(ISIT eval)
        print⇒(SELF = 0.0⇒(disp ← 48. disp←46. disp←48)
                SELF < 0.0⇒(disp ← 025. fprint - SELF) fprint SELF)
        )
to t fprint (ev)
t
to fprint n p q s | | fuzz z (
   § Normalize to [1,10) §
        (:n < 1⇒(☞p ← -(10.0 / n) epart 10.0)
        ☞p ← n epart 10.0)
        ☞n ← fuzz + n / 10.0 ipow p.
        (n≥10.0⇒(☞p←p+1. ☞n←n/10.0  § ugly fix for now § ))
    § Scientific or decimal §
        (☞s ← fuzz*2. ¯4<p<6⇒(☞q ← 0. p < 0⇒ (disp←z[1 to 1-p])
                                        ☞s ← s * 10.0 ipow p)
                                ☞q←p. ☞p←0)
    § Now print (s suppresses trailing zeros) §
        do 9(disp ← 48 + n ipart. ☞p ← p - 1. ☞n ← 10.0 * n fpart.
        p < 0⇒((p = ¯1⇒(disp ← 46)) n < ☞s ← 10.0 * s⇒(done)))
                (p = ¯1⇒(disp ← 48))
                q ≠ 0⇒(disp←0145. q print))
PUT fprint ☞fuzz 5.0 * 10.0 ipow ¯9.
PUT fprint ☞z fill string 4
0.00
PUT float ☞fprint #fprint.
done
```

§

RECTANGLES

Rectangles are the main graphical object.
The paint-clause provides all the power of bit-BLT.

§

```
to rectangle t a b | origin extent (
        paint⇒(CODE 43)     § see <smalltalk>BITBLT.DOC §
        drag⇒(CODE 44)
        has⇒(☞t ← :. ⇑origin ≤ t ≤ origin + extent)
        comp⇒(SELF paint 14 ¯1)
        inset⇒(⇑rectangle (origin + ☞t ← :) extent - t + :)
        's⇒(⇑§ eval)
        moveby⇒(☞origin ← origin+:)
        moveto⇒(☞origin ← :)
        growto⇒(☞extent ← (:)-origin)
        center⇒(⇑origin + extent/2)
        dragto⇒(SELF drag 0 ☞t ← :.
                ☞origin ← t)
        makebuff⇒(☞a ← string 2*extent y*☞t←(extent x+15)/16.
                ☞b ← PNT a.
                BLT 0 0 b+2 t 0 0 extent x extent y mem 60 32 origin x origin y.
                ⇑a)     § copies all the bits into a string §
        loadbuff⇒(☞b ← PNT :.
                ☞t ← (extent x+15)/16.
                BLT 0 0 mem 60 32 origin x origin y extent x extent y b+2 t 0 0)
                § restores image from string made by makebuff §      isnew⇒(☞origin ← :.
☞extent ← :)
        print⇒(☞rectangle print sp.
                origin print sp extent print)
        outline⇒(☞a ← turtle at origin - point 1 1.
                a wide 2 's ink ← (with⇒(:) ¯3).
                do 2 (a turn 90 go extent x+2 turn 90 go extent y+2)))

to PNT (mem 255←:. ⇑mem 255)

to BLT (CODE 41)
```

BLT operation gray destbase destraster destx desty
width height sourcebase sourceraster sourcex sourcey

§
TURTLES
§

```
to turtle var | pen ink width dir x xf y yf frame | f (
        CODE 21
        §
        CODE 21 is equivalent to:
        go⇒(draw a line of length :)
        turn⇒(turn right : (degrees))
        goto⇒(draw a line to :(x), :(y))
        §
        place⇒(pen←0. SELF goto :. pen←1. ⇑SELF)
        's⇒(var ← 8. ←⇒(⇑var ← :) ⇑var eval)
        wide⇒(width←:. ⇑SELF)
        up⇒(dir ← 270. ⇑SELF)
        pendn⇒(pen ← 1. ⇑SELF)
        penup⇒(pen ← 0. ⇑SELF)
        black⇒(ink ← ¯3. ⇑SELF)
        white⇒(ink ← ¯1. ⇑SELF)
        xor⇒(ink ← ¯2. ⇑SELF)
        is⇒(ISIT eval)
        home⇒(SELF place frame's extent/2. dir←270. ⇑SELF)
        erase⇒(frame paint 12 0. ⇑SELF)
        isnew⇒(ink ← ¯3. pen ← width ← 1. dir←270.
                frame ← (frame⇒(:) f).
                x← xf← y← yf←0.
                SELF place (at⇒(:) frame's extent/2))
        )
```

§
BASIC TEXT
§

```
to textframe input | window frame last justtabspace scanmode reply buf font | defont(
        show⇒(of⇒(buf←:. last←(to⇒(:) buf length)
                        0 CODE 51)
                0 CODE 51)      § re-displays the text §
        charofpt⇒(1 CODE 51)          § takes one arg, a point. Returns
                index in buf of char being pointed at. Also leaves in
                reply the point which is upperleft of that char. §
        ptofchar⇒(2 CODE 51)          § takes one arg, an integer. Returns
                a point which is upperleft of the character buf[arg] §
        comp⇒(window paint 14 ¯1)
        's⇒(⇑8 eval)
        put⇒(buf←:. last←buf length. at.
                window's(extent←point 1000 1000. origin←:) :.
                window's(extent←point (:)-origin x :)
                        SELF ptofchar last+1 x SELF fontheight.
                SELF show)      § a quick way to put some text
                (first arg) at a point(second arg). Leaves window being
                just the enclosing rectangle of the text. §
        fontheight⇒(⇑(font[12]+font[14]))
        scrolln⇒(input←:. ⇑SELF charofpt
                frame's(origin+point extent x
                                (font[12]+font[14])*input))
                § takes nlines as arg. Returns number of chars which
                have to be removed from the beginning of the string
                in buf in order to effect a scroll of nlines lines §
        is⇒(ISIT eval)
        justify⇒(off⇒(justtabspace ← justtabspace ⊡077777)
                on. justtabspace ← justtabspace ⊞0100000)
        tab⇒(input ← :.
                justtabspace ← (justtabspace ⊡0100577) + (input ⊠8))
        space⇒(justtabspace ← (justtabspace ⊡0177400) + :)
        isnew⇒(window ← frame ← :.
                reply ← scanmode ← 0. justtabspace ← 0110004.
                buf←(with⇒(:) string 0). last←buf length.
                font ← (font⇒(:input is string⇒(input) defont)
```

?

justify

0010000 tab

+0000377 space

```
defont)))
```

§

SCROLLING WINDOWS

Text windows which also behave as output streams, via the message ←. The text scrolls when the window is full. The read message makes for a very simple dialogue window. Many features of the earlier dispframes have been pruned or relocated(see also rectangles).

§

```
to dispframe n t | strm text (
        ←⇒(↑t ← :. (t = 8⇒(strm's (↑i ← i-1)) strm ← t). SELF show)
        show⇒(repeat
                (text show of strm'ss to ↑t ← strm's i.
                 ↑n ← text's reply.
                 n is point⇒(done)
                 n ≥ t ⇒(done)
                 t < ↑n ← text scrolln 2 ⇒(done).
                 strm's(↑s←s[n+1 to i] ↑i←i-n)))
        sub⇒(↑n ← :. indisp dispframe
                text'sframe inset ↑t←point 10 7 t (n eval). SELF show)
        read⇒(SELF←20.  § the prompt symbol §
                ↑n ← 0. repeat
                        (4=↑t←kbd⇒(⇑↑(done))  § ctrl-d §
                         t=010⇒(n>0⇒(↑n←n-1.  § backspace §
                                 strm's(↑i←i-1).
                                 kbck⇒() SELF show)
                                                 SELF show)
                         t=036⇒(strm←036. SELF←015.  § do-it §
                                 ⇑read of strm's(s[i-n+1 to i-1]))
                         strm ← t. ↑n←n+1. kbck⇒() SELF show))
        clear⇒(text'swindow paint 12 0. ↑strm←stream)
        outline⇒(text'sframe outline)
        wclear⇒(text'swindow paint 12 0)
        's⇒(⇑⁸ eval)
        is⇒(ISIT eval)
        isnew⇒(↑text←textframe :.
                ↑n ← text fontheight.
                ↑t ← text'sframe'sextent y.
                text'sframe'sextent y ← t\n.
                SELF clear. SELF outline))
```

§

THE TRUTH ABOUT FILES AND DIRECTORIES

a file is found in a directory ("dirinst") by its file name ("fname"), and has a one "page", 512 character string ("sadr"). "rvec" is an optional vector of disk addresses used for random page access.

↑fi ← ⟨directory⟩ file ⟨string⟩ old -- finds an old file named ⟨string⟩ in ⟨directory⟩ or returns false if does not exist or a disk error occurs.

↑fi ← ⟨directory⟩ file ⟨string⟩ new -- creates a new file or returns false if it already exists. if neither old or new is specified, an existing file named ⟨string⟩ will be found or a new file created. if ⟨directory⟩ is not specified, the current default directory is used.

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

⟨directory⟩ file ⟨string⟩ 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), thereby 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" = 2 if any label block numbers ("nextp" thru "sn2") have been changed, 1 if "sadr" has been changed, 0 if the current page is clean. the user need not worry about this unless (s)he deals directly with the label or "sadr". it might be noted here that multiple instances 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 language pointer to offending disk command block (dcb)) signals a disk error.

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

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

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

        ⇐←⇒ (17 CODE 50)
                        §
                        fi ← ⟨number⟩ or ⟨string⟩
                        §

        ⇐next⇒ ((⇐word⇒ (⇐←⇒ (7)
                                §
                                fi next word ← ⟨number⟩ -- possibly increment pointer to word
                                boundary, write number.
                                §

                        6)
                                §
                                fi next word -- read a number
                                §

                ⇐into⇒ (16)
                                §
                                fi next into ⟨string⟩ -- read a string
                                §

                23) CODE 50)
                                §
                                fi next -- read a character
                                §

        ⇐set⇒ (⇐to. (⇐end⇒(13)
                §
                fi set to end -- set file pointer to end of file.
                SELF set to read 037777 0
                §

                ⇐write⇒(5)
                                §
                                fi set to write ⟨number⟩ ⟨number⟩ -- set file pointer to :spage :schar.
                                if current page is dirty, or "reset", "set to end" or page change occurs,
                                flush current page. read rvec[spage] or pages sequentially until
                                pagen=spage. allocate new pages after end if necessary spage=-1 is
                                interpreted as pagen, i.e. current page. ☞bytec←schar
                                §

                ⇐read. 4) CODE 50)
                                §
                                same as "write" except stop at end of file
                                §

        ⇐skipnext⇒ (18 CODE 50)
                §
```

```
        fi skipnext ⟨number⟩ -- set character pointer relative to current position. (useful for
        skipping rather than reading, or for reading and backing up)  SELF set to read pagen
        bytec + :.
         §

◀end⇒ (10 CODE 50)
         §
        fi end -- return false if end of file has not occurred.
        nextp=0⇒ (bytec⟨numch⇒(⇑false))⇑false
         §

◀'s⇒ (⇑8 eval)

◀flush⇒ (12 CODE 50)
         §
        fi flush -- dirty=0⇒ ( ) write current page
         §

◀is⇒ (ISIT eval)

◀remove⇒ (dirinst forget SELF)
         §
        remove file from filesopen list of directory
         §

◀close⇒ ((◀shortened⇒(SELF shorten to pagen bytec)
        dirinst 's bitinst flush. SELF flush)
        SELF remove. ⇑fname)
         §
        fi close or ☞fi←fi close (if fi is global) -- flush bittable and current page, remove
        instance from filesopen list of directory
         §

◀shorten⇒ (◀to. ◀here⇒ (SELF shorten pagen bytec) 14 CODE 50)
         §
        fi shorten to ⟨number⟩ ⟨number⟩ -- shorten a file. SELF set to read :spage :schar.
        ☞x←nextp. ☞nextp←0. ☞numch←bytec. ☞dirty←2. SELF flush. deallocate succeeding
        pages
         §

◀reopen⇒(31 CODE 50. dirinst remember SELF)
         §
        doesnt look it up unless necessary
         §

◀print⇒ (disp ← fname)
         §
        file prints its name
         §

◀reset⇒ (11 CODE 50)
         §
        fi reset -- reposition to beginning of file
        SELF set 1 0
         §

◀intostring⇒(SELF set to end.
        ☞x ← string bytec + 512 * pagen - 1.
        SELF reset.  ⇑SELF next into x)

◀copyto⇒(☞x←:. repeat
                (x ← sadr[bytec+1 to numch].
                 nextp=0⇒(done)
                 SELF set to pagen+1 0)
         x shorten to here)

◀random⇒ (SELF set to end.
        ☞rvec ← (◀reset⇒(vector pagen)
                        null rvec⇒(vector pagen)
                        rvec[1 to pagen]).
        0 < ☞x ← rvec[1 to rvec length] find first nil⇒(
```

```
            for x ← x to rvec length
            (SELF set x 0. rvec[x] ← curadr)))
        §
        fi random -- initialize a random access vector to be used in fi set... new pages
        appended to the file will not be randomly accessed
        §

    ◀pages⇒ (20 CODE 50)
        §
        fi pages ⟨number⟩ ... ⟨number⟩ -- 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, use "sadr". diskaddress (=-1 uses "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.
        §

    isnew⇒ (☞fname←ncheck :. fname is false⇒
            (⇑false)
        (null ☞dirinst ← #curdir⇒
            (☞dirinst ← directory 's defdir. dirinst open)).
            §
            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
            §

        ◀exists⇒ (22 CODE 50. ⇑fname)
                §
                return false if file name does not occur in the directory
                §

        ◀delete⇒ (15 CODE 50. ⇑☞deleted)
                §
                delete a file (see intro)
                §

        ☞sadr ← (◀using⇒ (:) string 512).
                §
                set up file string buffer
                §

        ◀rename⇒ (☞x ← ncheck :. x is false⇒
                (error ☞(bad new name)⇑nil)
            file x exists⇒ (error ☞(name already in use))
            2 CODE 50. ☞fname ← x. 21 CODE 50.
            SELF set 0 12. SELF ← fname length.
            SELF ← fname. SELF flush. ⇑fname)
                §
                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
                §

        ◀load⇒ (2 CODE 50. dirinst flush. 8 CODE 50)
                §
                lookup an old file and load (overlay) a Swat memory image; return
                via save.
                §

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

        ◀save⇒ (SELF set to write 256 0. SELF reset.
```

```
                dirinst close.  9 CODE 50)
                        §
                        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.
                        §

            ⇐intostring⇒(⇑SELF intostring)
            dirinst remember SELF) ))
                        §
                        finally, file puts itself into the filesopen list of its directory
                        §
file 's(ev)
to ncheck str i x || legal (☞str←:.
        (str is string⇒(0 < 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 99] find x⇒ ( § legal § )
                0100 < x < 0133⇒ ( § uppercase § )
                ⇑false)
        x=056⇒(⇑str) ⇑str+ ☞.chars)
§
check that the file name is a proper length string containing only lower/upper case letters, digits, or
legal characters. if name does not end with a period, append one.
§

PUT ncheck ☞legal fill string 6
+-()⇑⇒.
done

§
a directory is found in a directory ("dirinst"), has a bittable file ("bitinst") for allocating new pages,
a file of file entries ("filinst" -- file names, disk addresses etc.), and a list of currently open files
("filesopen" which is an "obset"). each file must ask its directory for the bittable when page allocation
is necessary, and the system directory (via its local directory) for the disk number.

☞di ← ⟨directory⟩ directory ⟨string⟩ old/new

currently, ⟨directory⟩ and old or new must be specified.

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

(to directory name exp hd | dirinst bitinst filinst filesopen | dirname bitname
        curdir defdir (

        ⇐open⇒ (☞curdir ← SELF.  filinst is file⇒ ()
                bitinst⇒(☞filinst←file filinst old. ☞bitinst←file bitinst old)
                ☞bitinst ← dirinst 's bitinst.
                ☞filinst ← (⇐new⇒(file filinst new) file filinst old).
                filinst ⇒ (dirinst remember SELF)⇑false)
                §
                di open -- (normally not user-called since access to the directory always usually opens
                it) initialize directory file and bittable instances:  a "subdirectory" uses the bittable of
                its system directory and puts itself into that filesopen list.
                §

        ⇐file⇒ (SELF open. ⇑apply file)
                §
                di file ⟨string⟩... -- open directory. create file instance (see file intro)
                §

        ⇐is⇒ (ISIT eval)

        ⇐remember⇒ (filesopen ← :)
```

```
«forget⇒ (filesopen delete :)
        §
        add or delete file instances in filesopen.
        §

«print⇒ (disp←0133. filesopen print. disp←0135)
        §
        di or di print. --print the filesopen list.
        §

«map⇒ (SELF open. ☞exp ← :. filinst reset.
        repeat (filinst end⇒ (cr. done)
        0 = 1024 * ☞hd← filinst next word⇒
                (filinst skipnext 0 max 2*1023 *hd-1)
        filinst skipnext 10.
        ☞name ← filinst next into string filinst next.
        filinst skipnext (2 * hd *1023)-13+name length.
        exp eval))
        §
        di map ⟨expression⟩ -- evaluate an expression for each file name
        §

«list⇒ (SELF map ☞(disp←name. sp))
        §
        di list -- print the entry names contained in filinst
        §

«flush⇒(filesopen map ☞(each flush))

«close⇒ ((filinst is file⇒ (SELF flush. ☞filesopen ← obset.
        ☞filinst ← filinst 's fname.
        dirinst is directory⇒ (dirinst forget SELF. ☞bitinst ← false)
        ☞bitinst ← bitinst 's fname)). ⇑☞closed)
        §
        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.
        §

«use⇒ (☞defdir ← SELF)
        §
        di use -- change the default directory.
        §

«'s⇒ (⇑8 eval)

«directory⇒ (SELF open. ⇑apply directory)
        §
        di directory ⟨string⟩... -- open directory. create directory instance
        §

isnew⇒ (☞filesopen ← obset.
        «device⇒(☞dirinst ← :. ☞filinst ← dirname.
                ☞bitinst ← bitname)
        ☞dirinst ← curdir. ☞filinst ← :. ☞bitinst ← false.
        «new⇒(SELF open new) «old. SELF open)))
        §
        store the directory file name in filinst; "subdirectories" use "curdir" as their directory,
        are flagged by "false bitinst" and then opened. for system directories, dirinst is a
        number indicating disk number, and bitinst contains the file name for the appropriate
        bittable. the directory is not immediately opened.
        §

directory 's(ev)
☞dirname ← fill string 7
SysDir.
☞bitname ← fill string 15
DiskDescriptor.
done
```

```
        §
        names of the system directory and bittable
        §

☞dp0 ← directory device 0.
dp0 use
        §
        create a system directory instance which is initially closed. dp0 is the same (default) directory
        that the Operating System believes in.  type ☞dp1 ← directory device 1 to similarly access
        another Model 31 disk or a Model 44.
        §

to error Badr Bptr Btoparec Barec Blev || c sub (
        (0=☞Badr←mem 0102⇒(☜knows⇒(ev ⇑) dson. :Bptr))
        ☞Btoparec←☞Barec←leech AREC.☞Blev←0
        disp sub ☞(disp←indisp stream ((0=Badr⇒(Bptr print)
                mem 0102←0. disp←0377☒ mem Badr.
                for Badr←Badr+1 to Badr+(mem Badr)☐⁻9 (
                        ☞Bptr←mem Badr.
                        disp←Bptr☐⁻8. disp←Bptr☒ 0377))
                disp contents)
        cr c ev))
error knows
to c Dclass Dcode l1 l2 Dadr Dvadr Di cpc (
        (☜top⇒(☞Barec←leech Btoparec[5]. ☞Blev←0))
        (☜down⇒
                (
                ☞Di←:.
                for l1 to Di-1
                        (
                        (null Barec[5]⇒(done))
                        ☞Barec←leech Barec[5].
                        ☞Blev←Blev+1
                        )
                )
        )
null Barec[5]⇒(.)
disp←indisp stream(
        ☞Barec←leech Barec[5]. ☞Dclass←Barec[0].
        ☞Blev←Blev+1.Blev print. sp.
        (GET Dclass ☞TITLE) print. ☞: print.
        ☞Dadr←☞Dvadr←Barec[1]☐
        ☞l1←leech nil. ☞l2←leech l1.
        repeat (
                l2[1]⊟Dvadr.
                05350=0177770☒l1[0]☒(done)  § find start of codevec §
                ☞Dvadr←Dvadr-1)
        ☞Dcode←l2[1]. l2[1]⊟1.
        ☞cpc←Dadr-Dvadr+1.
        for Di to Dcode length
                (Di<cpc-10⇒(again)Di<cpc-5⇒(disp←056)
                Di>cpc+10⇒(done)Di>cpc+5⇒(disp←056)
                sp. (Di=cpc⇒(disp←031))
                Dcode[Di] is vector⇒(☞() print) Dcode[Di] print).
⇑disp contents
))
to sub disp (☞disp ← GET USER ☞disp. (:) eval)
done

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

        to button | | bits (
                ☞bits←bits⊞mouse 7.
                ☜check⇒(⇑0<bits)
                 ☞mask←:. mask=mask☒bits⇒(☞bits←0. ⇑mask)
                 ⇑false)
        PUT button ☞bits 0
        §
```

§ 056 is period §

§ 031 is ⇒ §

Returns true if that pattern is being held down
The check mesg strobes and tells if any buttons have gone down
§

§
*****Keyboard translation*****
§

```
to kbd (⇑kmap[TTY])
☞kmap ← string 0377.
        for i←0200 to 0377(kmap[i] ← 0177) §ILLEGAL GETS ⇒ §
        for i←001 to 0177(kmap[i] ← i) §REGULAR ASCII/S §

                §CHAR -- KEYBOARD §
kmap[0341]←kmap[0301]←kmap[0274]←kmap[0254]←01.
                                §≤ -- ↑A or ↑, or ↑SHF, §
kmap[0342]←kmap[0302]←kmap[0236]←kmap[0237]←kmap[037]←kmap[036]←02.
                                §▁ -- ↑B or any TOP BLANK KEY. §

kmap[0343]←kmap[0303]←0343. §new escape -- ↑C or ↑c; §
kmap[0272]←03.  § ß -- ↑: §
kmap[0344]←kmap[0304]←04. §↑D  /done/  §
kmap[0345]←kmap[0305]←kmap[023]←05. §□ -- ↑E or ↑SHF ESC §
kmap[0346]←kmap[0306]←kmap[0262]←06. §@ -- ↑F or ↑2 §
kmap[0347]←kmap[0307]←kmap[0273]←07. §' -- ↑G or ↑; §
kmap[0353]←kmap[0313]←kmap[0245]←013. §♫ -- ↑K or ↑SHF5 §
kmap[0356]←kmap[0316]←kmap[0275]←016. §≠ -- ↑N or ↑= §
kmap[0357]←kmap[0317]←kmap[0242]←0174. §" -- ↑O or ↑SHF/ §
kmap[0360]←kmap[0320]←kmap[0271]←020. §➔( -- ↑P or ↑9 §
kmap[0361]←kmap[0321]←kmap[0261]←021. §! -- ↑Q or ↑1 §
kmap[0362]←kmap[0322]←kmap[0300]←022. §≥ -- ↑R or ↑SHF2 §
kmap[0363]←kmap[0323]←023.              §'s -- ↑s §
kmap[0364]←kmap[0324]←024.              §Ω -- ↑T §
kmap[0365]←kmap[0325]←kmap[0255]←kmap[0140]←025. §" -- ↑U or ↑- §
kmap[0366]←kmap[0326]←kmap[0265]←026. §% -- ↑V or ↑5 §
kmap[0367]←kmap[0327]←kmap[0376]←027. §✓ -- ↑W or ↑SHF6 §
kmap[0370]←kmap[0330]←kmap[0246]←030. §¯ -- ↑X or ↑SHF7 §
kmap[0371]←kmap[0331]←kmap[0277]←031. §➔ -- ↑Y or ↑SHF/ §
kmap[0362]←kmap[0322]←kmap[0276]←kmap[0256]←022.
                                §≥ -- ↑R or ↑. or ↑SHF. §
kmap[0333]←kmap[0264]←033. §$ -- ↑[ or ↑4 §
kmap[0334]←kmap[0267]←034. §& -- ↑\ or ↑7 §
kmap[0335]←kmap[0375]←035. §COMMENTCHAR -- ↑] §
kmap[0337]←kmap[0336]←kmap[0222]←kmap[0212]←kmap[022]←kmap[012]←036.
                §! -- LF or ↑ ← or ↑SHF← §
kmap[0247]←017. §/ -- SHF\ or ↑/ - THE NEW COMMENT CHAR - §
kmap[0257]←0176. §? -- SHF6 or ↑/ §
kmap[0263]←043. §# -- SHF3 or ↑3 §
kmap[0270]←052. §* -- SHF8 or ↑8 §
kmap[0220]←kmap[0210]←kmap[020]←010. §ALL BS/S §
kmap[0225]←kmap[0215]←kmap[025]←015. §ALL CR/S §
kmap[0240]←kmap[0230]←kmap[030]←040. §ALL SP/S §


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.)

to t fool || fontname (
        textframe 's (mem 70 ← ☞defont ←  file fontname intostring).
        ☞disp←dispframe rectangle point 2 514 point 508 168.
        disp ← version. ☞defs ← obset.
        ☞fool←#to. to to toAtm (CODE 19 defs←toAtm. toAtm)
        PUT USER ☞DO ☞(ev). ☞t←0.)
PUT t ☞fontname fill string 11
ST10.STRIKE
☞version←fill string 35
Welcome to SMALLTALK [August 2]

to ev (repeat(cr. disp ← stringof read eval))
```

```
(to displaysetup i displayspace | displaycolor displaywidth displayinset
                displayheight | MINDISPLAYWIDTH MAXDISPLAYWIDTH
                MINDISPLAYHEIGHT MAXDISPLAYHEIGHT
                MAXDISPLAYSPACE STARTINGDISPLAYWIDTH
                STARTINGDISPLAYHEIGHT STARTINGDISPLAYINSET
                black white initflag (

◀set⇒(☞displaywidth ← :. ☞displayheight ← :.
                ☞displayinset ← :. ☞displaycolor ← :.
        (displaycolor = 0 ⇒(☞displaycolor ← white) ☞displaycolor ← black)
        ☞displaywidth ←
        ((displaywidth max MINDISPLAYWIDTH)
                                                min MAXDISPLAYWIDTH)\2.
        ☞displayheight ←
        ((displayheight max MINDISPLAYHEIGHT)
                                                min MAXDISPLAYHEIGHT)\2.
        ☞displayinset ← (displayinset min MAXDISPLAYINSET) max 0.

        for i ← displayinset to 0 by ¯1 (
                (displaywidth + displayinset) > MAXDISPLAYWIDTH ⇒
                        (☞displayinset ← displayinset - 1. again)
                                done)

        for i ← displayheight + 1 to MINDISPLAYHEIGHT by ¯1(
                ☞displayspace ← displaywidth * displayheight.
                        (displayspace > MAXDISPLAYSPACE ⇒
                                (☞displayheight ← displayheight - 1. again)
                                done))

        for i ← displaywidth to MINDISPLAYWIDTH by ¯2(
                        ☞displayspace ← displaywidth * displayheight.
                                displayspace > MAXDISPLAYSPACE ⇒
                                (☞displaywidth ← displaywidth - 2. again)
                                done)
        ☞displayinset ← displayinset ⍁8.
        (initflag⇒()☺erase.)
        ☞STDISPLAYRECT ←
                        rectangle point 0 0 point (displaywidth * 16) displayheight.
        turtle's f ← STDISPLAYRECT. ☞☺ ← turtle.
        CODE 46.
        ☞displayinset ← displayinset ⍁¯8.
        (◀savedisp⇒(disp's text's (☞window ← ☞frame ←
                                STDISPLAYRECT inset point 2 2 point 2 2)))
        initflag ⇒(☞initflag ← false.) disp outline. disp show
        )

◀init⇒(☞MINDISPLAYWIDTH ← 12. ☞MAXDISPLAYWIDTH ← 36.
        ☞MINDISPLAYHEIGHT ← 56. ☞MAXDISPLAYHEIGHT ← 808.
        ☞MAXDISPLAYINSET ← 24.
        ☞STARTINGDISPLAYWIDTH ← 32.
        ☞STARTINGDISPLAYHEIGHT ← 684.
        ☞STARTINGDISPLAYINSET ← 2.
        ☞MAXDISPLAYSPACE ←
                STARTINGDISPLAYWIDTH * STARTINGDISPLAYHEIGHT.
        ☞black ← 040000. ☞white ← 0. ☞initflag ← true.
        ☞STDISPLAY ← displaysetup STARTINGDISPLAYWIDTH
                        STARTINGDISPLAYHEIGHT STARTINGDISPLAYINSET white)
◀'s⇒(⇑8 eval)
◀is⇒(ISIT eval)
◀refresh⇒(SELF set displaywidth displayheight displayinset displaycolor)
◀restart⇒(SELF set STARTINGDISPLAYWIDTH STARTINGDISPLAYHEIGHT
                        STARTINGDISPLAYINSET white)
isnew⇒(SELF set (:) (:) (:) (:))
        ))
         §
        Displaysetup allows the change of the display control block to be changed from Smalltalk. The
        init code is run from junta -- setting up STDISPLAY -- and should not be run again. None
        of the classs variables should be touched by the casual user.
         §

to expand x torigin (☞x ← (:)\2.☺ erase.
```

```
                STDISPLAY's(☞displayheight ← displayheight - x.
                (displayheight<MINDISPLAYHEIGHT⇒(
                ☞x ← x - (MINDISPLAYHEIGHT - displayheight)))
                ☞MAXDISPLAYSPACE ← displaywidth * displayheight.
            ☞torigin ← displayheight - (disp's text's frame'sextent y + 2).
             disp'stext'sframe  moveto point 2 (torigin < 2 ⇒(2) torigin).
             ☞initflag ← true).
            STDISPLAY refresh. CODE 38)
    §
    t is called to set up a display frame, and defs 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. A
    display height of less than 56 scan lines is protected against.
    §

§
THE SMALLTALK READ ROUTINE (name changed to protect ev)
§

(to junta scanner | | read1 tablscan rdnum mknum rdstr rtb1 type
        letbit digbit sepbit atbits qtbit crbit combit
        (⇙'s⇒(⇑°eval)
        ⇙of⇒((:scanner is string⇒ (☞scanner ← stream of scanner))
                ☞scanner ← tablscan scanner type. ⇑read1 rtb1)
        ⇑disp read))

junta 's(ev)

to read1 rbuf rdtb flag ( :rdtb. ☞rbuf ← stream of vector 10. scanner read.
        ⇑rbuf contents)

to tablscan mask | source type seq isfil nxtchr (
        ⇙next⇒ (CODE 14 next.
        § CODE 14 is equivalent to...
                :mask=0⇒(☞t←string 1. t[1]←nxtchr.
                ☞nxtchr←source next. ⇑atom t)
                seq reset. repeat
                        (0 = nxtchr⇒(done).
                        0 = mask ▫ type[nxtchr + 1]⇒(done).
                        seq ← nxtchr. ☞nxtchr ← source next)
                        ⇑seq contents §)
        ⇙skip⇒(☞nxtchr ← source next)
        ⇙read⇒(repeat (rdtb[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. ~1)1). ☞base ← (nxtchr=060⇒(8)10).
        ☞n ← mknum scanner next digbit base. ☞flag ← false.
        056 = nxtchr⇒ (scanner skip. ☞fs ← scanner 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) ⇑n*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 qtbit.
        scanner skip.
        nxtchr=047⇒(seq←047. ⇑seq contents+rdstr)
        ⇑t)


§
INITIALIZATION OF READ TABLES
§

☞rtb1 ← vector 256.
☞type ← string 256.
```

$ 056 is period $

```
type[1 to 256] ← all 0.
☞combit ← 2* ☞crbit ← 2* ☞sepbit ← 2* ☞letbit ← 2* ☞digbit ← 2* ☞qtbit ← 1.
☞atbits ← letbit + digbit.
to scanner n v i j (:n. :v. repeat (:i.
        for j ←1+i to 1+(◁to⇒(:) i)
                (type[j]←type[j]⊟n. rtb1[j]←v)
        ◁and⇒() done))
scanner combit+crbit+qtbit ☞(rbuf←scanner next 0) 0 to 0377.  § non-CR and non-quote §
scanner combit ☞(scanner skip. scanner next combit. scanner skip) 017.  § comment quote §
scanner qtbit ☞(rbuf ← rdstr) 047.  § string-quote §
scanner crbit 0  015.  § -CR- §
scanner sepbit ☞(scanner next sepbit) 011 and 014 and 015 and 040.  § tab, LF, FF, CR, blank §
scanner letbit ☞(rbuf←atom scanner next atbits) 0101 to 0132 and 0141 to 0172.  § letters §
scanner digbit ☞(rbuf←rdnum. flag⇒(rbuf←☞.)) 060 to 071.  § digits §
scanner 0 ☞(scanner next crbit) 032.  § ↑Z (BRAVO format) §
scanner 0 ☞(rbuf←rdnum. flag⇒(rbuf←☞.)) 025.  § high-minus §
scanner 0 ☞(scanner skip. rbuf ← (read1 rtb1) eval) 020.  § eval-paren §
scanner 0 ☞(scanner skip. rbuf ← read1 rtb1) 050.  § left-paren §
scanner 0 ☞(scanner skip. rbuf ← nil. done) 051.  § right-paren §
scanner 0 ☞(rbuf ← nil. done) 0 and 036.  § null and DOIT §
done
☞read ← #junta.
PUT read ☞TITLE ☞read.  § cover our tracks §

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 ☞r fill string 7
REM.CM.
PUT quit ☞b fill string 9
Sys.Boot.

☞fill ← nil

to junta (point 0 0. displaysetup init. PUT USER ☞DO ☞(t). CODE 31)
        § allocates display over OS after setting up t §
```