Inter-Office Memorandum

To	Smalltalk-80 Implementors	Date	February 19, 1981
From	Glenn Krasner	Location	Palo Alto
Subject	These Sources	Organization	LRG

XEROX

Disclaimers

What is contained here will surely *not* be the sources for the Smalltalk-80 second (and final) release. Most of the things you may find different are remnants from Smalltalk-76 syntax. Some of these are:

This source listing is *not* complete, but is a listing of only those classes and methods that will be met early when running the image.

≡ to mean equal object, rather than "==";

◁ in the stream code, which was used to do a "next" without advancing the stream pointer.

_ used as a terminator.

o and o← used instead of at: and at:put:.

keyword← used as a syntactic form of keyword selectors.

The formats of class definitions. In the ones here:
a) fields: is used to name the instance variables (like Smalltalk-80 instance variable names)
b) subclassof: is used rather than the Smalltalk-80 superclass
c) there are no meta-classes in this system.

Class UserView, which here does a lot of the user interaction, will not exist at all in Smalltalk-80.

We hope that these differences will not hinder your implementations. The second release will match the Smalltalk-80 described in the book.

Kernel Classes

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:23:32 pm.'_!
                                                        "Object"
Class new title: #Object
   subclassof: nil
   fields: ''
   declare: '';
   asFollows_!
```

Object is the superclass of all classes. It is an abstract class, meaning that it has no state, and its main function is to provide a foundation message protocol for its subclasses. Three instances of this class are defined: nil, true, and false.

```
Comparison
≤ x | |
   ↑self > x ≡ false
≡ x | |<primitive: 78>
   ↑self ≡ x "In case this is reached by perform:"
≠ x | |
   ↑self = x ≡ false
≥ x | |
   ↑self < x ≡ false
= x | |
   ↑self ≡ x
> x | |
   ↑x < self  "Not circular, subclasses need to define <"
empty | |
   ↑self length = 0
sameAs: object | |
   ↑self ≡ object
| otherBool | | "logical OR"
   self ifTrue: [↑true] ifFalse: [↑otherBool]

Classification
class | |<primitive: 24>
   user croak
is: x | |
   ↑self class ≡ x
ls: x | |
    "Is the class x a superclass or class of self"
   self class ≡ x
     ifTrue: [↑true].
   ↑self class lsa: x
isArray | |
   ↑false
isnt: x | |
   ↑self class ≡ x ≡ false
isNumber | |
   ↑false
species | |
   ↑self class

Construction
, x | v |
   v ← Vector new: 2.
   v o 1 ← self.
   v o 2 ← x.
   ↑v
```

```
asParagraph | |
   ↑self asString asParagraph
asStream | |
   ↑self asVector asStream
asVector | v |
   self ≡ nil
     ifTrue: [↑Vector new: 0].
   v ← Vector new: 1.
   v o 1 ← self.
   ↑v
copy | |
    "create new copy of self"
   (self is: Object)
     ifTrue: [↑self].
   ↑self class copy: self
inVector | vec |
   vec ← Vector new: 1 "Return me as the sole element of a new Vector.".
   vec o 1 ← self.
   ↑vec
recopy | |
    "recursively copy whole structure"
   (self is: Object)
     ifTrue: [↑self].
   ↑self class recopy: self

Aspects
o i | | <primitive: 63>
   "Old synonym for at:"
   ↑i subscripts: self
o i ← value | | <primitive: 64>
   "Old synonym for at:put:"
   ↑i subscripts: self ← value
asOop | |<primitive: 39>
   user croak
at: i | | <primitive: 63>
   ↑i subscripts: self
at: i put: value | | <primitive: 64>
   ↑i subscripts: self ← value
canunderstand: selector | |
   ↑self class canunderstand: selector
error: s | |
   ↑user notify: s
fields | |
   self class "Return an Array of all my field names or many of my subscripts."
isVariable
     ifTrue: [↑self class instvars concat: (self length ≤ 50
        ifTrue: [1 to: self length]
        ifFalse: [(1 to: 20) concat: (self length - 20 to: self length)])].
   ↑self class instvars
hash | |<primitive: 39>
   user croak
inspect | |
   user restartup: (InspectWindow new of: self)
inspectfield: n | fixedSize |
    "used by variable panes"
   self class isVariable
     ifTrue:
```

```
      [fixedSize ← self class instsize.
      n > fixedSize
        ifTrue: [↑self o (n - fixedSize)].
      ↑self instfield: n]
    ifFalse: [↑self instfield: n]
instfield: n | |<primitive: 37>
  "Old synonym for instVarAt:"
  user croak
instfield: n ← val | |<primitive: 38>
  "Old synonym for instVarAt:put:"
  user croak
length | | <primitive: 65>
  "Old synonym for size"
  user notify: 'Message not understood'
sizeInWords | class |
  class← self class.
  class isBytes ifTrue: [↑self length+1/2].
  class isVariable ifTrue: [↑class instsize+self length].
  ↑class instsize
title | |
  ↑self class title + '.' + self asOop base8

Printing
asString | strm |
  strm ← (String new: 16) asStream.
  self printOn: strm.
  ↑strm contents
filout | file |
  ↑user displayoffwhile: [
    (file ← dp0 file: self title asFileName.
    self fullprinton: file.
    file close)]
fullprinton: strm | |
  self ≡ nil
    ifTrue: [strm append: 'nil']
    ifFalse:
     [self ≡ false
       ifTrue: [strm append: 'false']
       ifFalse:
        [self ≡ true
          ifTrue: [strm append: 'true']
          ifFalse: [self class print: self on: strm]]]
print | |
  user show: self asString
printGenericOn: strm | t |
  t ← self class title.
  strm append: (('AEIO' has: t o 1)
    ifTrue: ['an ']
    ifFalse: ['a ']).
  strm append: t.
  self class isVariable ifTrue:
     [strm append: ' of size '.
     strm print: self length]
printOn: strm | t |
  self class ≡ Object
    ifTrue: [strm append: (self ≡ nil
       ifTrue: ['nil']
```

```
        ifFalse:
         [self ≡ false
           ifTrue: ['false']
           ifFalse:
            [self ≡ true
              ifTrue: ['true']
              ifFalse: ['unknown Object']]])]
    ifFalse: [self printGenericOn: strm]
printon: strm indent: level | |
  self printOn: strm

Compiler Defaults
's code | |
  ↑LADCompiler new evaluate: code asStream in: false to: self notifying: self
interactive | |
  ↑false
notify: errorString at: position in: stream | |
  ↑self notify: errorString at: position in: stream for: self class
notify: errorString at: position in: stream for: class | syntaxWindow |
  NotifyFlag
    ifTrue:
     [syntaxWindow ← SyntaxWindow new of: errorString at: position in: stream for:
class from: thisContext sender.
      "thisContext sender← nil."
      user restartup: syntaxWindow]
    ifFalse:
     [user notify: errorString.
      ↑false]

System Primitives
doesNotUnderstand: message | selector args |
  selector ← message selector.
  args ← message args.
  user notify: 'Message not understood: ' + selector.
  ↑self perform: selector withArgs: args
execute: method | |
  <primitive: 78>
  ↑self execute: method withArgs: (Vector new: 0)
execute: method with: arg | |
  <primitive: 78>
  ↑self execute: method with: {arg}
execute: method withArgs: vec | |
  <primitive: 86>
  user croak
nextInstance | |<primitive: 82>
   "return next in enumeration of my class, false if done"
  user croak
perform: selector | | "Send self the message, selector; it had better be unary"
  <primitive: 36>
  ↑self perform: selector withArgs: (Vector new: 0)
perform: selector with: arg1 | | "selector had better take 1 arg"
  <primitive: 36>
  ↑self perform: selector withArgs: {arg1}
perform: selector with: arg1 with: arg2 | | "selector had better take 2 args"
  <primitive: 36>
  ↑self perform: selector withArgs: {arg1 , arg2}
perform: selector with: arg1 with: arg2 with: arg3 | | "selector had better take 3 args"
```

```
    <primitive: 36>
    ↑self perform: selector withArgs: {arg1 , arg2 , arg3}
perform: selector withArgs: vec | | <primitive: 85>
    user croak
startup | |
     "loopless scheduling"
    self firsttime
      ifTrue:
       [[self eachtime] whileTrueDo: [].
       ↑self lasttime].
    ↑false

As yet unclassified
become: other | |<primitive: 35>
    user croak
do: aBlock || ↑self asStream do: aBlock
growTo: newLength | |<primitive: 33>
    user croak
_!
SystemOrganization classify: #Object under: 'Kernel Classes'._!
```

"UserView"

```
Class new title: #UserView
   subclassof: Object
   fields: 'screenrect "<Rectangle> current screen size"
      vtab "<Integer=0mod2> offset from hardware top"
      htab "<Integer=0mod16> offset from hardware left"
      scale "<Integer=1 or 2> 2 means double bits mode"
      color "<Integer=0 or 1> 1 means reverse field"
      projectWindow "my representative in an overview"
      disp "<dispframe> default message stream"
      sched "<Vector> Windows in this view"'
   declare: 'currentCursor mxoffset myoffset screenMenu ';
   asFollows_[
```

This is a melting-pot, incorporating the notions of user interaction (mouse, keyboard), display context (current screen view), and global operations (reading the clock, leaving the system).

```
Mouse, Cursor, Keys
anybug | |
   ↑self buttons > 0
anykeys | |
   ↑self keyset > 0
bluebug | |
   ↑self buttons = 1
buttons | |
   ↑self primMouseKeys land: 7
currentCursor | |
   ↑currentCursor
currentCursor: c | coff p |
   currentCursor ← c.
   coff ← c offset.
   p ← self mp.
   mxoffset ← coff x - htab.
   myoffset ← coff y - vtab.
   self cursorloc← p "Cursor ref pt contin across boundary".
   currentCursor beCursor
cursorloc← pt | |
   ↑self primCursorLoc← pt - (mxoffset ⊙ myoffset)
kbck | t |
   (t ← self rawkbck)
     ifTrue: [↑kbMap o t].
   NoteTaker
     ifTrue: [↑false].
   self purgealittle.
   ↑false
kbd | |
   [self rawkbck] whileFalseDo:
      [NoteTaker
        ifFalse: [self purgealittle]].
   ↑kbMap o self rawkbd
kbdnext | event |
   NoteTaker
     ifTrue:
      [self kbck
        ifTrue: [↑self kbd].
       ↑false]
```

```
    ifFalse:
     [ "returns next character (mapped) if any; otherwise false"
     [(event ← Events dequeue) or: [(event ← Events primitiveDequeue)]]
whileTrueDo:
        [event isKbdDown
          ifTrue: [↑kbMap ∘ event stroke]].
      ↑false]
keyset | |
   ↑(self primMouseKeys lshift: ¯3) land: 31
leftShiftKey | |
    "left shift key down?"
   ↑(self userBits ∘ 6) anymask: 8r100
mp | |
   ↑self primMousePt + (mxoffset ⊙ myoffset)
mpnext | |
   self redbug "return next mouse point if red button or tablet is down; otherwise
false"
    ifTrue: [↑self mp].
   ↑false
nobug | |
   ↑self anybug ≡ false
rawkbck | event rdpt stroke |
    "flush events until key down or no event."
   (NoteTaker and: ['NoEventQ'])
     ifTrue:
      [[stroke ← user primKbdPeek] whileTrueDo:
         [stroke ≥ 0
           ifTrue: [↑stroke].
         user primKbdNext].
       ↑false].
   [event ← Events peek] whileTrueDo:
      [event isKbdDown
        ifTrue: [↑event stroke].
      Events next].
   ↑false
rawkbd | stroke |
   (NoteTaker and: ['NoEventQ'])
     ifTrue:
      [[self rawkbck] whileFalseDo: [].
      ↑self primKbdNext].
   [stroke ← self rawkbck] whileFalseDo: [ "wait for activity"
      ].
   Events next.
   ↑stroke "if key down, return stroke"
redbug | |
   ↑self buttons = 4
userBits | |
   ↑self userBitsInto: (String new: 10)
userBitsInto: aString | | <primitive: 94>
   "Read the keyboard bits into bytes 1-8, keyset into byte 9, mouse into byte 10"
   user croak
waitbug | |
   [self anybug] whileFalseDo: [].
   ↑self mp
waitclickbug | |
   self waitnobug.
   ↑self waitbug
```

```
waitnobug | |
   [self anybug] whileTrueDo: [].
   ↑self mp
x | |
   ↑self mp x
y | |
   ↑self mp y
yellowbug | |
   ↑self buttons "From Smalltalk 5.4i on 30 October 1979 at 2:28:57 pm." = 2

Screen Views
bugScreenMenu | n i t3 |
   (t3 ← screenMenu bug) = 1
     ifTrue:
      [projectWindow ≡ nil
        ifFalse: [projectWindow runParent]]
     ifFalse:
      [t3 = 2
        ifTrue: [user snapshot]
        ifFalse:
         [t3 = 3
           ifTrue: [user quit]
           ifFalse:
            [t3 = 4
              ifTrue: [self schedule: ProjectWindow init]
              ifFalse:
               [t3 = 5
                 ifTrue: [self schedule: BrowseWindow default]
                 ifFalse:
                  [t3 = 6
                    ifTrue: [self schedule: (CodeWindow new class: UserView selector:
#workspace para: (UserView code: #workspace) formerly: false)]
                   ifFalse:
                    [t3 = 7
                      ifTrue:
                       [n ← Turtle init.
                       n color: black.
                       n width: 4.
                       n inking: oring.
                       (1 to: 50) do:
                          [:i | n go: i * 4.
                          n turn: 89]]]]]]]]
copyIn: p | |
   ↑UserView new screenrect: screenrect copy vtab: vtab htab: htab scale: scale color:
color projectWindow: p disp: disp sched: #()
displayoffwhile: expr | |
   ↑expr eval "Don't bother turning off the display"
install | |
   self screenextent: screenrect extent tab: htab ⊙ vtab
projectWindow | |
   projectWindow ≡ nil
     ifTrue:
      [projectWindow ← ProjectWindow new.
      projectWindow userview: self changes: Changes parent: projectWindow].
   ↑projectWindow
restoredisplay | |
screenrect | |
```

```
    ↑screenrect
screenrect: t1 vtab: t2 htab: t3 scale: t4 color: t5 projectWindow: t6 disp: t7 sched: t8 |
|
    screenrect ← t1.
    vtab ← t2.
    htab ← t3.
    scale ← t4.
    color ← t5.
    projectWindow ← t6.
    disp ← t7.
    sched ← t8

Window Scheduling
promote: window | |
    sched promote: window
restart | i |
    Events ≡ nil
      ifTrue:
        ["Events ← EventQueue init. Top init3.  initialize Event queue and Time
interrupt"].
    NormalCursor show.
    self restart: [(user run)]
restart: code | u |
    u ← code cleancopy.
    u sender← nil.
    thisContext sender releaseFully.
    thisContext sender← nil.
    code ← nil "release caller chain".
    MessageDict new freeMethods "release held code".
    disp frame flash.
    [true] whileTrueDo: [u eval]
restartup: window | | "Equivalent to schedule new window, restart, and enter."
    NoteTaker
     ?: [NormalCursor show.
       self schedule: window]
     ~: [thisContext sender releaseFully.
       thisContext sender← nil.
       NormalCursor show.
       self schedule: window.
       thisContext tempframe all← nil.
       self run: true]
restore | w |
    screenrect clear.
    projectWindow ≡ nil
      ifFalse: [projectWindow putTitle].
    disp outline.
    (sched length to: 1 by: ¯1) do: [:w | (sched o w) show]
run | |
    self run: false
run: topFlag | i w forward |
    "topFlag means sched o 1 already is awake"
    forward ← (topFlag
            ifTrue:
             [w ← sched o 1.
             [w eachtime] whileTrueDo: [].
             w lasttime]
            ifFalse: [true]).
```

```
    [true] whileTrueDo:
      [i ← 0.
      [(i ← i + 1) > sched length
        ifFalse:
         [w ← (forward
                ifTrue: [sched o i]
                ifFalse: [sched o (sched length + 1 - i)]).
          w firsttime]] whileFalseDo: [].
       i > sched length
        ifTrue:
         [ "check for bug in empty space"
         user yellowbug
           ifTrue: [self bugScreenMenu]]
        ifFalse:
         [sched promote: w.
         [w eachtime] whileTrueDo: [].
         forward ← w lasttime]]
scheduleWindow: window | |
    sched ≡ nil
      ifTrue: [sched ← window inVector]
      ifFalse: [sched ← window inVector concat: sched]
scheduleOnBottom: window | |
    sched ≡ nil
      ifTrue: [sched ← window asVector]
      ifFalse: [sched ← sched concat: window asVector]
topWindow | |
    ↑sched o 1
unschedule: window | t |
    0 < (t ← sched find: window)
      ifTrue: [sched ← sched o (1 to: t - 1) concat: sched o (t + 1 to: sched length)]

Notify Window
notifier: titleString stack: stack interrupt: flag | |
    self restoredisplay "Restore the full display.  Schedule a one-paned window to notify
the user that errorString happened.",
    ↑NotifyWindow new of: titleString stack: stack interrupt: flag
notify: errorString | notifyWindow | "Create a notify window looking at the Context
stack"
    NotifyFlag ifFalse:
       [user cr; show: '**NotifyFailed** '; show: errorString; cr.
       user print thisContext sender; cr.
       user print thisContext sender sender; cr.
       user print thisContext sender sender sender; cr.
       user show: '**click mouse to proceed'; waitclickbug. ↑NotifyFlag← true].
    notifyWindow ← self notifier: errorString stack: thisContext sender interrupt: false.
    notifyWindow
      ifTrue:
       [notifyWindow takeCursor.
       NoteTaker
         ifTrue:
          [self schedule: notifyWindow.
          user run]
         ifFalse:
          [thisContext sender← nil.
          Top currentPriority = 1
            ifTrue: [self restartup: notifyWindow]
            ifFalse:
```

```
        [self scheduleOnBottom: notifyWindow.
        Top errorReset]]]

Dialog Window
clear | |
    "clear disp of debris and characters"
    disp clear
clearshow: str | |
    disp clear.
    disp append: str.
    disp show
cr | |
    disp cr
croak | |
    self notify: 'A primitive has failed.'
ev | |
    disp ev
frame | |
    "return rectangle of dialogue window"
    ↑disp text frame
next← x | |
    disp cr "simulate a Vector Stream".
    disp print: x.
    disp show
print: x | |
    disp print: x.
    disp show
read | |
    ↑disp read
request: s | |
    ↑disp request: s
show | |
    disp outline.
    disp show
show: str | |
    disp append: str.
    disp show
space | |
    disp space
tab | |
    disp tab

Changes
changedMessages | |
    ↑Changes contents sort
noChanges | |
    Changes init

System quit/resume
backup | |
    "back up smalltalk on ivy and resume"
    ivy open.
    ivy delete: 'small.boot'.
    ivy store: 'small.boot'.
    ivy close
InLd: fileid | |<primitive: 75>
    "write out the core image, then load in OS"
```

```
    user notify: 'file problem'
  overlay: fileid | |
    self releaseExternalViews.
    E ≡ nil
      ifFalse: [E sleep]. "put the ethernet to sleep"
    self InLd: fileid.
    [user keyset > 0] "we start here after a resume"
      whileTrueDo: [user show: 'The keyset is stuck'.
      user cr]
  quit | |
    self quitFrom: self "yup"
  quitFrom: controller | |
    self overlay: #(0 0 0 0 0 ).
    screenrect clear.
    controller restore
  releaseExternalViews | t |
    SourceFiles notNil do: [:t | t close].
    dp0 close.
    dp1 close.
    (externalViews length to: 1 by: ¯1) do: "release (obsolete) known external views"
      [:t | (externalViews o t) release.
      externalViews o t ← nil].
    externalViews reset
  snapshot | i |
     "write the OT and Data of this Smalltalk out"
    self releaseExternalViews.
    self snapshotPrimitive.
    self currentDisplay: CurrentDisplay.
    currentCursor beCursor.
    self restore
  snapshotPrimitive | |<primitive: 83>
     "write the OT and Data of this Smalltalk out"
    user croak

Misc System Stuff
classInit | |
     " UserView classInit. "
    screenMenu ← Menu new string: 'exit to overview
snapshot
quit
open a subview
open a browser
open a workspace
turtle demo'
classNames | x |
     "an alphabetized Vector of all Smalltalk class titles uniqued"
    AllClassNames ≡ nil ifTrue: [AllClassNames ←
      (Smalltalk contents all: [:x | x] suchThat: [Smalltalk o x is: Class]) sort].
    ↑AllClassNames
core | n nobjects nwords class inst bytes | "user core"
    nobjects← nwords← 0.
    Class allInstancesDo:
      [:class|
      class isVariable
        ifTrue: [n← 0.
          bytes← class isBytes.
          class allInstancesDo:
```

```
                [:inst | n← n+1.
                nwords← nwords
                   +(bytes ifTrue: [inst length+1/2] ifFalse: [inst length])]]
          ifFalse: [n← class howMany].
        nobjects← nobjects+n.
        nwords← nwords+(class instsize+2*n)]. "+2 for header"
    ↑{nobjects , nwords}
file: file classes: classes changesOnly: ch | cl |
    "called by UserView release to write just changes or entire system on a
    new file.  also, see comment in Class archiveOn:changesOnly:.

    write class comment and message text onto a FileStream (which could refer
    to an AltoFile, ILFile, etc.). either just changes or everything are
    written and replaced with RemoteParagraph references"
    ch
      ifTrue: [file settoend]
      ifFalse: [file reset].
    file readwriteshorten.
    classes do: [:cl | Smalltalk o cl archiveOn: file changesOnly: ch].
    file close.
    file readonly
purgealittle | |<primitive: 0>

release | m |
    (m ← Undeclared contents "prepare to release this version (after editing UserView
version)
    and possibly copying Sources file (see writeSources:)") length > 0
      ifTrue: [user notify: 'Undeclared contains ' + m asString]
      ifFalse:
       [user displayoffwhile: [
          (m ← Sources directory "either create a new Sources file (write all messages)
or append only changes" checkName: '<Smalltalk>Smalltalk.Sources.' + user
versionName "for repeated releases in same version.
        should also work for Sources local (if renamed)".
          user writeSources: (m = Sources name
            ifTrue: [Sources]
            ifFalse: [Sources directory file: m]) "make workspace local".
          UserView md code: #workspace ← UserView code: #workspace.
          user writeChangedMessages: (phylum file: '<Smalltalk>ChangedMessages'))].
        user noChanges.
        user releaseMessage]
releaseMessage | |
    user clearshow: 'Welcome to ' + user version
version | |  "user version"
    ↑'Smalltalk-80 of February 8, 1981'
versionName | s |
    s ← self version asStream "skip Smalltalk".
    s skipTo: 32 "return version identification, e.g. 5.5f".
    ↑s upto: 32
workspace | |
    user notify: 'Not meant to be executed' "
XEROX - Learning Research Group

user screenextent: 640@580 tab: 0@50.
NotifyFlag ← true.
Changes init.
user changedMessages
```

```
user changedClasses
user changedCategories
Undeclared contents

to set the default printer
PrinterName←'Menlo'.
PrinterName←(PressFile new) selectPrinter: PrinterName.

to change phylum to access your account
user releaseExternalViews. phylum name: 'name' password: 'password'.

dpo filin: ↗('Changes.st').
(dpo file: 'changes.st') filout.
(dpo file: 'xxx') edit.
dpo pressfilin: ↗('xxx.press').
(dpo filesMatching: '*.st') sort
dpo list. dpo freePages
dpo delete: 'old'
dpo rename: 'old' newName: 'new'

for reinitializing Sources and phylum
Sources release. phylum release. Sources reopen.

to make Smalltalk Sources local
 | s. s ← 'Smalltalk.Sources.'.
(phylum asFtpDirectory) retrieve: '<Smalltalk>' + s + user versionName as: s; close.
Sources on: (dpo file: s).

to switch back to remote Sources
Sources close; on: (phylum file: '<Smalltalk>Smalltalk.Sources.' + user versionName).


to filin a remote Smalltalk file
phylum filin: ↗('<Small-goodies>NotifyChange.st').

to print a remote/local press file
(phylum pressfile: '<Smalltalk>xxx.press') toPrinter.
(dpo pressfile: 'xxx.press') toPrinter: 'Lilac'.

File noChanges.
BitRect new fromuser; edit.
user schedule: (defaultBitRectEditor newframe).

DocumentEditor new defaultdocument: 'test'.
DocumentEditor new init: (Document new fromPress: 'test.document').


user releaseExternalViews.
E sleep. E kill. E ← nil.
E ← Etherworld new. E broadcastFilter: true. E wakeup.
Sources reopen.

for primary Smalltalk access to file servers and printers at other sites.
substitute yourserver for phylum above, compile this workspace
PrinterName ← 'name-of-your-printer'.
Smalltalk declare: ↗yourserver.
yourserver ← ILFileDirectory new directory: 'name-of-your-server'.
```

```
yourserver name: 'Smalltalk-User' password: 'Smalltalk'.
Sources on: (yourserver file: '<Smalltalk>Smalltalk.Sources.' + user versionName).
Changes init.

user Swat.
"
writeChangedMessages: ChangedMessages | class m ms |
   ChangedMessages settoend "append changed messages to a file (usually on
[phylum])".
   ChangedMessages cr.
   ChangedMessages cr.
   ChangedMessages asParagraphPrinter stamp.
   class ← ''.
   user changedMessages do:
      [:m | ms ← m asStream.
      (ms upto: 32) = class
        ifTrue:
         [ChangedMessages append: ', '.
         ChangedMessages append: (ms upto: 32)]
        ifFalse:
         [ChangedMessages cr.
         ChangedMessages append: m.
         class ← m asStream upto: 32]].
   ChangedMessages close
writeSources: newSources | |
   user file: newSources classes: SystemOrganization changesOnly: newSources end
"write a new Sources file (usually on [phylum]Smalltalk.Sources.xxx
      (i.e. xxx = user versionName))
   if it's a new file or empty, write all Sources. otherwise it better be a copy of
   the previous Sources file (only changes will be appended. do the copy with ftp)" ≡
false.
   Sources close.
   Sources ← newSources

As yet unclassified
currentDisplay: form | |
   form class≡DisplayForm
   ifTrue:
    [ form bits class≡DisplayBitmap
     ifTrue:
     [CurrentDisplay ← form.
      screenrect ← 0 ⊙ 0 rect: form extent.
      vtab ← htab ← mxoffset ← myoffset ← 0.
      (DisplayBLTer ← BitBlt init) sourceForm: CurrentDisplay.
      DisplayBLTer destForm: CurrentDisplay.
      CurrentDisplay beDisplay]
     ifFalse: [user notify: 'bitmap should be DisplayBitmap']
    ]
   ifFalse: [user notify: 'form should be DisplayForm']
primCursorLoc← pt | |<primitive: 43>
   ↑self croak
primKbdNext | |<primitive: 45>
   ↑self croak
primKbdPeek | |<primitive: 44>
   ↑self croak
primMouseKeys | |<primitive: 46>
   ↑self croak
```

```
primMousePt | |<primitive: 42>
    ↑self croak
]
SystemOrganization classify: #UserView under: 'Kernel Classes'.]
UserView classInit]
```

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:25:01 pm.'_!
                                                        "LinearMeasure"
Class new title: #LinearMeasure
   subclassof: Object
   fields: ''
   declare: '';
   asFollows_!

This abstract class assists subclasses which measure something linear like dates or
times.

Comparison
< aMeasure | |
   ↑self subclassResponsibility
= aMeasure | |
   ↑self subclassResponsibility
between: min and: max | |
   ↑self ≥ min and: [self ≤ max]
compare: aMeasure | | "change to ↑(self-aMeasure) sign ?"
   self < aMeasure
     ifTrue: [↑1].
   self = aMeasure
     ifTrue: [↑2].
   ↑3
max: aMeasure | |
   self > aMeasure
     ifTrue: [↑self].
   ↑aMeasure
min: aMeasure | |
   self < aMeasure
     ifTrue: [↑self].
   ↑aMeasure
_!
SystemOrganization classify: #LinearMeasure under: 'Numbers'._!
```

Numbers

"Number"

```
Class new title: #Number
   subclassof: LinearMeasure
   fields: ''
   declare: 'ln2 fourthpi halfpi sqrt2 twopi degreesPerRadian pi radiansPerDegree ';
   asFollows_[
```

*The abstract class Number is at the top of the number hierarchy. Its subclasses are AbstractInteger, Float, and Fraction. Certain operators (+, −, *, <, etc.) provide automatic coercion directed by the operands generality indecies.*

```
Arithmetic
+ aNumber | |
   self subclassResponsibility
- aNumber | |
    "circular with negated"
   ↑self + aNumber negated
/ aNumber | |
    "circular with reciprocal"
   ↑self * aNumber reciprocal
// aNumber | |
    "integer quotient defined by division with truncation toward negative infinity.
9//4 = 2,  -9//4 = -3.  -0.9//0.4 = -3.  \\ returns the remainder from this
division."
   ↑(self / aNumber) floor
abs | |
    "return the absolute value (positive magnitude) of this number"
   self < 0
     ifTrue: [↑self negated].
   ↑self
divMod: aNumber | |
    "returns a vector of (quotient, remainder) defined by division with truncation
toward negative infinity."
   ↑{(self//aNumber) , (self\\aNumber)}
intdiv: aNumber | | "temporary"
   ↑self divMod: aNumber
negated | |
    "circular with -"
   ↑0 - self
negative | |
   ↑self < 0
positive | |
   ↑self >= 0
quo: aNumber | |
    "integer quotient defined by divsion with truncation toward zero.  -9 quo: 4 = -2,
-0.9 quo: 0.4 = -2.  rem: returns the remainder from this division."
   ↑(self / aNumber) truncated
reciprocal | |
    "circular with /"
   ↑1 / self
sign | |
   self > 0 ifTrue: [↑1].
   self < 0 ifTrue: [↑-1].
   ↑0
squared | |
   ↑self * self
\ aNumber | |
```

```
    ↑self \\ aNumber
\\ aNumber | | "modulo"
    "remainder defined in terms of //. returns a number with the same sign as
aNumber. e.g. 9\\4 = 1, -9\\4 = 3, 9\\-4 = -1, 0.9\\0.4 = 0.1 "
    ↑self - ((self//aNumber) * aNumber)

Conversions
asPtX | |
    "pretend to be a Point for Point +-*/"
asPtY | |
    "pretend to be a Point for Point +-*/"
asRectCorner | |
    "pretend to be a Rectangle for Rectangle +-*/"
asRectOrigin | |
    "pretend to be a Rectangle for Rectangle +-*/"

Subscripts
cansubscript: a | |
    ↑self asInteger cansubscript: a
subscripts: a | |
    ↑a ∘ self asInteger
subscripts: a ← val | |
    ↑a ∘ self asInteger ← val

Intervals, Points
⊙ y | |
    ↑Point new x: self y: y
to: limit do: aBlock | | ↑(self to: limit by: 1) do: aBlock

Compatibility
isLarge | |
    ↑false
isNumber | |
    ↑true
sameAs: anObject | | "temporary?"
    ↑self = anObject

Math functions
cos | |
    "for angles in radians"
    self < 0.0
      ifTrue: [↑(self + halfpi) sin].
    ↑(halfpi - self) sin
floorLog2 | i cnt |
    "floor of log base 2"
    self < 0
      ifTrue: [↑self negated floorLog2].
    self < 1
      ifTrue: [↑self reciprocal floorLog2 negated].
    i ← 1.
    cnt ← 0.
    [self ≥ i] whileTrueDo:
       [i ← i + i.
        cnt ← cnt + 1].
    ↑cnt - 1
floorLog: base | x |
    "gives floor log.base self"
```

```
    self < base
      ifTrue: [↑0].
     "self assumed positive"
    self < base squared
      ifTrue: [↑1].
    x ← 2 * (self floorLog: base squared). "binary recursion like ipow"
    ↑x + (self / (base ipow: x) floorLog: base)
raisedTo: x | |
    "fixed powers in log n steps"
    x = 0
      ifTrue: [↑1.0].
    x = 1
      ifTrue: [↑self].
    x > 1
      ifTrue: [↑(self * self raisedTo: x//2) * (self raisedTo: x\\2)].
    ↑(self raisedTo: x negated) reciprocal
sin | x x2 sum const | "for angles in radians"
    self < 0.0 ifTrue: [↑self negated sin negated]. "normalize to 0≤self≤(pi/4) "
    self > twopi ifTrue: [↑(self asFloat \\ twopi) sin].
    self > pi ifTrue: [↑(self - pi) sin negated].
    self > halfpi ifTrue: [↑(pi - self) sin].
    sum ← x ← self.
    x2 ← x * x.
    #(-0.16666667 0.0083333315 -1.98409e-4 2.7526e-6 -2.39e-8) do:
       [:const | "Now compute the series"
       sum ← const * (x ← x * x2) + sum].
    ↑sum
"
 (3.1416/4) sin
"
sqrt | guess i expt |
    self ≤ 0.0
      ifTrue: [self = 0.0 ifTrue: [↑0.0].
          user notify: 'sqrt invalid for x < 0']
      ifFalse:
       [ "copy and halve the exponent for first guess"
       guess ← self asFloat + 0.0.
       Float instsize = 3
          ?: [guess instfield: 1 ← (guess instfield: 1)//4 * 2] "old format"
          ~: [expt← (guess instfield: 1) bits: 1 to: 8.  "IEEE format"
             guess instfield: 1 ←
                (guess instfield: 1) bits: 1 to: 8 ← expt-127/2+127].
       1 to: 5 do:  "Newton's method:"
          [:i | guess ← self - (guess * guess) / (guess * 2.0) + guess].
       ↑guess]
tan | x x2 sum const |
    "for angles in radians"
    self < 0.0
      ifTrue: [↑self negated tan negated].
     " normalize to 0≤self≤(pi/4) "
    self > pi
      ifTrue: [↑(self asFloat \\ pi) tan].
    self > halfpi
      ifTrue: [↑(self - halfpi) tan negated].
    self > fourthpi
      ifTrue: [↑1.0 / (halfpi - self) tan].
    sum ← x ← self.
```

```
    x2 ← x * x.
    #(0.3333314 0.1333924 0.05337406 0.024565089 0.002900525 0.0095168091 ) do: [:const |
"Now compute the series"
        sum ← const * (x ← x * x2) + sum].
    ↑sum

Rounding
floor | trunc |
    "returns the nearest integer toward negative infinity"
    trunc ← self integerPart.
    self ≥ 0
      ifTrue: [↑trunc].
    self = trunc
      ifTrue: [↑trunc].
    ↑self truncated - 1
rounded | |
    "rounds to the nearest integer"
    ↑(self + (self < 0
      ifTrue: [0.5]
      ifFalse: [0.5])) integerPart
truncated | |
    "truncates to the nearest integer toward zero"
    self subclassResponsibility
truncateTo: aNumber | |
    "truncates to next multiple of aNumber toward zero"
    ↑(self//aNumber) * aNumber

Coercion mechanism
differentGenerality: aNumber | |
    ↑self generality ≠ aNumber generality
moreGeneralThan: aNumber | |
    ↑self generality > aNumber generality
retry: aUniqueString coercing: aNumber | t3 |
    (t3 ← self generality compare: aNumber generality) = 1
      ifTrue: [↑(aNumber coerce: self) perform: aUniqueString with: aNumber].
    t3 = 3
      ifTrue: [↑self perform: aUniqueString with: (self coerce: aNumber)].
    user notify: 'coercion attempt failed'

Conversion
asDirection | |
    ↑self cos ⊙ self sin
coerce: aNumber | |
    "must be defined by all Number classes."
    self subclassResponsibility.
    "e.g. for Integer,
    ↑ aNumber asInteger"
degreesToRadians | |
    ↑self * radiansPerDegree
fractionalPart | |
    ↑self - self truncated
integerPart | |
    ↑self - self fractionalPart
readFrom: aStream | start radix value |
    aStream ← aStream asStream.
    start ← aStream position.
    radix ← Integer new readFrom: aStream radix: 10.
```

```
    (radix is: Error) ifTrue: [↑radix].
    aStream ◁ 114
      ifTrue: "<radix>r<number>"
       [radix <= 1 ifTrue: [↑Error new message: 'INVALID RADIX'].
       start ← aStream position.
       value ← Integer new readFrom: aStream radix: radix.
       (value is: Error) ifTrue: [↑value]]
      ifFalse:
       [value ← radix.
       radix ← 10].
    aStream ◁ 46
      ifTrue:
       [(aStream end ≡ false and: [aStream peek isDigitRadix: radix])
          ifTrue: ".<fraction>"
          [aStream position← start.
          ↑Float new readFrom: aStream radix: radix]
          ifFalse:
          [aStream skip: -1.  "back up"
          ↑value]].
    aStream ◁ 101
      ifTrue: "e<exponent>"
       [aStream position← start.
       ↑Float new readFrom: aStream radix: radix].
    ↑value
to: stop | |
    "create an Interval"
    ↑Interval new from: self to: stop by: 1
to: stop by: step | |
    "create an Interval"
    ↑Interval new from: self to: stop by: step

Initialization
classInit | |
 "constants from Computer Approximations, pp. 182-183
    pi = 3.14159265358979323846264338327950288
    pi/2 = 1.57079632679489661923132169163975144
    pi/4 = 0.78539816339744830961566084581987572
    pi*2 = 6.2831853071795864769252867665590057 6
    pi/180 = 0.01745329251994329576923690768488612
    2.0 ln = 0.69314718055994530941723212145817657
    2.0 sqrt = 1.41421356237309504880168872420969808"

    pi ← 3.1415927.
    halfpi ← pi / 2.0.
    fourthpi ← pi / 4.0.
    twopi ← pi * 2.0.
    radiansPerDegree ← pi / 180.0.
    degreesPerRadian ← 180.0 / pi.
    ln2 ← 0.69314718.
    sqrt2 ← 1.4142136

Backward compatibility
ipow: x | | ↑self raisedTo: x

As yet unclassified
bits: interval | |
    ↑self bits: interval start to: interval stop
```

```
SystemOrganization classify: #Number under: 'Numbers'.
Number classInit
```

"Float"

```
Class new title: #Float
   subclassof: Number
   fields: ''
   declare: '';
   asFollows_!
```

These floating-point numbers are good for about 8 or 9 digits of accuracy, and the range is between plus and minus 10↑4000. Here are some valid floating-point examples:
8.0 13.3 0.3 2.5e6 1.27e-300 -12.987654e2412
Mainly: no imbedded blanks, little e for tens power, and a digit on both sides of the decimal point.

```
Arithmetic
≤ aNumber | |<primitive: 53>
   ↑self retry: #≤ coercing: aNumber
≠ aNumber | |<primitive: 56>
   aNumber isNumber ≡ false
     ifTrue: [↑true].
   ↑self retry: #≠ coercing: aNumber
≥ aNumber | |<primitive: 54>
   ↑self retry: #≥ coercing: aNumber
* aNumber | |<primitive: 57>
   ↑self retry: #* coercing: aNumber
+ aNumber | |<primitive: 49>
   ↑self retry: #+ coercing: aNumber
- aNumber | |<primitive: 50>
   ↑self retry: #- coercing: aNumber
/ aNumber | |<primitive: 58>
   ↑self retry: #/ coercing: aNumber
< aNumber | |<primitive: 51>
   ↑self retry: #< coercing: aNumber
= aNumber | |<primitive: 55>
   aNumber isNumber ≡ false
     ifTrue: [↑false].
   ↑self retry: #= coercing: aNumber
> aNumber | |<primitive: 52>
   ↑self retry: #> coercing: aNumber
hash | |
   ↑self instfield: Float instsize-1  "High bits of fraction"
reciprocal | |
   ↑1.0 / self

Conversion
asFloat | |
   ↑self
asFraction | denom |
    "only use 5 significant figures"
   denom ← (10.0 ipow: 4 - (self abs floorLog: 10.0)) asInteger.
   ↑(Fraction new numerator: (self * denom) rounded denominator: denom) reduced
asInteger | |<primitive: 60>
    "Return an Integer = self integerPart"
   ↑(self / 16384.0) asInteger * 16384 + (self \ 16384.0) asInteger
asLarge | me digits int i |
    "convert to LargeInteger"
   self < 0
```

```
    ifTrue: [↑(0.0 - self) asLarge negated].
  digits ← Stream default.
  self = 0.0
    ifTrue: [digits next← 0]
    ifFalse:
     [me ← self ipart.
     [me ≥ 1] whileTrueDo:
        [digits next← (me \ 256.0) asInteger.
        me ← me / 256.0]].
  digits ← digits contents.
  int ← LargeInteger new: digits length.
  (1 to: digits length) do: [:i | int ∘ i ← digits ∘ i].
  ↑int
coerce: aNumber | |
  ↑aNumber asFloat
copy | |
  ↑self
fpart | |<primitive: 59> .
  user croak
fractionalPart | |<primitive: 59>
  user croak
generality | |
  ↑80
ipart | |
   "Returns a Float with zero fractional part"
  ↑self - self fpart
maxMag | |
  self error: 'hash problem'
maxVal | |
  ↑self maxMag
minVal | |
  ↑self maxVal negated
readFrom: aStream | value |
  value ← Number new readFrom: aStream.
  (value is: Error) ifTrue: [↑value].
  ↑value asFloat
readFrom: aStream radix: radix | value exp int neg |
  aStream ← aStream asStream.
  radix ← radix asFloat.
  value ← 0.0.
  exp ← 0.
  neg ← (aStream ◁ 45 "-" ifTrue: [true] ifFalse: [aStream ◁ 21 "¯"]).
  [aStream end or: [(aStream peek isDigitRadix: radix) ≡ false]] whileFalseDo: [value
← value * radix + (aStream next - 48)].
  aStream ◁ 46 "."
    ifTrue: [[aStream end or: [(aStream peek isDigitRadix: radix) ≡ false]]
whileFalseDo:
        [value ← value * radix + (aStream next - 48).
        exp ← exp - 1]].
  aStream ◁ 101 "e"
    ifTrue:
     [int ← Integer new readFrom: aStream radix: 10.  "exp is always base 10"
     (int is: Error)
       ifTrue: [↑int].
     exp ← exp + int].
  neg
    ifTrue: [value ← value negated].
```

```
    exp = 0
      ifTrue: [↑value].
    exp > 0
      ifTrue: [↑value * (radix ipow: exp)].
    ↑value / (radix ipow: exp negated)
recopy | |
    ↑self
truncated | |<primitive: 60>
    "Return an Integer = self integerPart"
    ↑(self / 10000.0) truncated * 10000 + (self \ 10000.0) truncated

Printing
absPrintOn: strm digits: digits | fuzz x exp q i |
    "print me using digits significant figures"
    "x is myself normalized to [1.0, 10.0), exp is my exponent"
    exp ← (self < 1.0
          ifTrue: [(10.0/self floorLog: 10.0) negated]
          ifFalse: [self floorLog: 10.0]).
    x ← self / (10.0 ipow: exp).
    fuzz ← 10.0 ipow: 1 - digits.    "round the last digit to be printed"
    x ← 0.5 * fuzz + x.
    x ≥ 10.0                "check if rounding has unnormalized x"
      ifTrue:
        [x ← x / 10.0.
        exp ← exp + 1].
    (exp < 6 and: [exp > -4])
      ifTrue:                   "decimal notation"
        [q ← 0.
        exp < 0
          ifTrue: [strm append: '0.0000' o (1 to: 1 - exp)]]
      ifFalse:                  "scientific notation"
        [q ← exp.
        exp ← 0].
    [x >= fuzz] whileTrueDo:    "use fuzz to track significance"
        [i ← x truncated.
        strm next← 48 + i.
        x ← (x - i) * 10.0.
        fuzz ← fuzz * 10.0.
        exp ← exp - 1.
        exp = -1
          ifTrue: [strm append: '.']].
    [exp ≥ -1] whileTrueDo:
        [strm next← 48.
        exp ← exp - 1.
        exp = -1
          ifTrue: [strm append: '.']].
    q ≠ 0
      ifTrue:
        [strm append: 'e'.
        strm print: q]
printOn: strm | |
    self printOn: strm digits: 6
printOn: strm digits: digits | |
    "print me using digits significant figures"
    self > 0.0
      ifTrue: [self absPrintOn: strm digits: digits]
      ifFalse:
```

```
    [self = 0.0
      ifTrue: [strm append: '0.0']
      ifFalse:
       [strm append: '-'.
       self negated absPrintOn: strm digits: digits]]
!
SystemOrganization classify: #Float under: 'Numbers'.!
```

"Integer"

```
Class new title: #Integer
   subclassof: Number
   fields: ''
   declare: '';
   sharing: BitMasks;
   asFollows_[
```

This class serves as a common abstract superclass for all Integer implementations. My implementation subclasses are SmallInteger and LargeInteger.

```
Arithmetic
/ aNumber | | "temporary"
    "/ means div: when used between integers for now"
   (aNumber differentGenerality: self)
     ifTrue: [↑self retry: #/ coercing: aNumber].
   ↑self//aNumber
factorial | |
   self = 0
     ifTrue: [↑1].
   self < 0
     ifTrue: [self error: 'factorial invalid for: ' + self asString]
     ifFalse: [↑self * (self - 1) factorial]
intdiv: arg | |
   ↑{(self / arg), (self \ arg)}

Bit manipulation
bits: left to: right | width mask |
    "  0 is leftmost bit, 15 is rightmost"
   width ← right - left + 1.
   mask ← (width ≤ 15
          ifTrue: [bitson o (width + 1)]
          ifFalse: [¯1]).
   ↑(self lshift: right - 15) land: mask
bits: left to: right ← val | width mask |
    "  0 is leftmost bit, 15 is rightmost"
   width ← right - left + 1.
   mask ← (width ≤ 15
          ifTrue: [bitson o (width + 1)]
          ifFalse: [¯1]).
   ↑(self land: ((mask lshift: 15 - right) lxor: ¯1)) lor: ((val land: mask) lshift: 15 -
right)

Conversion
asFraction | |
   ↑self
base8 | |
   ↑self base: 8
base: b | strm |
   strm ← Stream default.
   strm print: b; append: 'r'.
   self printOn: strm base: b.
   ↑strm contents
readFrom: aStream | radix |
    "doesn't actually alter self; returns a new Integer or LargeInteger"
   aStream ← aStream asStream.
   radix ← self readFrom: aStream radix: 10.
```

```
    (radix is: Error) ifTrue: [↑radix].
    aStream ◁ 114 "r"
      ifTrue:
       [ "<radix>r<integer>"
       radix ≤ 1
         ifTrue: [↑Error new message: 'INVALID RADIX'].
       ↑self readFrom: aStream radix: radix]
      ifFalse: [↑radix] "no radix--this is the number"
readFrom: aStream radix: radix | char digit value none neg |
    "returns the next Integer or LargeInteger contained in aStream up to the first
 invalid character
    constants:
      ('0'o1) = 48
      ('A'o1) = 65
      ('A'o1) - 10 = 55"

    neg ← (aStream ◁ 45 "-" ifTrue: [true] ifFalse: [aStream ◁ 21 "¯"]).
    value ← 0.
    none ← true.
    aStream do:
      [:char | digit ← char - (char ≥ 65
             ifTrue: [55]
             ifFalse: [48]).
      (digit < 0 or: [digit ≥ radix])
        ifTrue:
         [(none or: [digit between: 0 and: 9])
           ifTrue: [↑Error new message: 'ILLEGAL/MISSING DIGIT'].
         aStream skip: -1.
         neg
           ifTrue: [↑value negated asInteger "catch 0 minVal"].
         ↑value]
        ifFalse:
         [value ← value * radix + digit.
         none ← false]].
    none
      ifTrue: [↑Error new message: 'INCOMPLETE NUMBER'].
    neg
      ifTrue: [↑value negated asInteger "catch 0 minVal"].
    ↑value
withDenominator: denom | |
    ↑self asFraction withDenominator: denom

Subscripts
cansubscript: a | |
    ↑self ≥ 1 and: [self ≤ a length]
subscripts: a | |
    (self cansubscript: a)
      ifTrue: [↑a o self asInteger].
    user notify: 'Subscript out of bounds: ' + self asString
subscripts: a ← val | t |
    (self cansubscript: a)
      ifTrue:
       [((a class isBits) and:
          [t ← val asInteger.
          (t<0 or: [t>(a class isBytes ifTrue: [255] ifFalse: [65535])])])
         ifTrue: [user notify: 'Improper store into String']
         ifFalse: [↑a o self asInteger ← t]]
```

```
    ifFalse: [user notify: 'Subscript out of bounds: ' + self asString]

Rounding
floor | |
    ↑self
rounded | |
    ↑self
truncated | |
    ↑self

Printing
printOn: strm | |
    self printOn: strm base: 10 "default print radix"

Comparison
hash | |
    ↑self

Backward compatibility
allmask: b | |
    ↑b = (self land: b)
anymask: b | |
    ↑0 ≠ (self land: b)
hibit | |
    ↑self highBit
land: n | |
    ↑self bitAnd: n
lor: n | |
    ↑self bitOr: n
lxor: n | |
    ↑self bitXor: n
nomask: b | |
    ↑0 = (self land: b)
]
SystemOrganization classify: #Integer under: 'Numbers'.]
```

"LargeInteger"

```
Class new title: #LargeInteger
   subclassof: Integer
   fields: ''
   declare: 'maxHi ';
   bytesize: 8;
   asFollows_[
```

LargeInteger implements arbitrary-precision integers, only assuming that SmallIntegers can represent numbers between -0400 and 017777. A LargeInteger consists of digits between 0 and 255, stored with the least significant digit first and no trailing zeroes unless the number as a whole is zero. Accessing beyond the end gives zeroes. The LargeNegativeInteger subclass implements negative numbers.

Messages required in SmallInteger for compatibility are digitLength, o, lastDigit. Messages which subclasses need to override for alternate implementation of sign are the same plus new:neg:.

```
Initialization
classInit | |
   maxHi ← 0 maxVal lshift: -8
new: length neg: neg | |
   neg
     ifTrue: [↑LargeNegInt new: length].
   ↑LargeInteger new: length

Backward compatibility
bits: int | |
   ↑self bitsAt: int
lshift: n | t b |
   "16-bit shifts for backward compatibility"
   (n ≥ 0 or: [self positive]) ifTrue: [t ← self arithmeticShift: n]
      ifFalse: [t ← (self land: 65535) asLarge arithmeticShift: n].
   b ← t o 2.
   ↑((b land: 127) - (b land: 128)) * 256 + (t o 1)

Arithmetic
* arg | |
   ↑(self digitMultiply: arg neg: self negative ≡ arg negative ≡ false) asInteger
+ arg | |
   arg negative
     ifTrue: [↑self digitSubtract: arg].
   ↑self digitAdd: arg
- arg | |
   arg negative
     ifTrue: [↑self digitAdd: arg].
   ↑self digitSubtract: arg
/ arg | ng quo |
   ng ← self negative ≡ arg negative ≡ false.
   quo ← (self digitDiv: arg neg: ng) o 1.
   (quo lastDigit = 0 and: [quo digitLength ≥ 2])
     ifTrue: [↑(quo growby: -1) asInteger].
   ↑quo asInteger
abs | |
   ↑self
negated | |
   (self o 1 = 0 and: [self digitLength = 1])
```

```
    ifTrue: [↑self].  "Zero"
  ↑self copyto: (self new: self digitLength neg: true)
\ arg | ng rem |
  ng ← self negative ≡ arg negative ≡ false.
  rem ← (self digitDiv: arg neg: ng) ∘ 2.
  (ng and: [0 ≠ rem])
    ifTrue: [↑(arg abs - rem) asInteger].
  ↑rem asInteger

Comparison
< arg | |
  arg negative
    ifTrue: [↑false].
  ↑(self digitCompare: arg) < 0
= arg | |
  arg isNumber
    ifTrue:
     [arg negative
       ifTrue: [↑false].
      ↑(self digitCompare: arg) = 0]
    ifFalse: [↑false]
> arg | |
  arg negative
    ifTrue: [↑true].
  ↑(self digitCompare: arg) > 0
compare: arg | |
  arg negative
    ifTrue: [↑3].
  ↑2 + (self digitCompare: arg)

Bit manipulation
arithmeticShift: n | m |
  m ← n asInteger.
  m ≥ 0
    ifTrue: [↑(self digitLshift: m \ 8 bytes: m / 8 lookfirst: true) asInteger].
  m ← 0 - m.
  ↑(self digitRshift: m \ 8 bytes: m / 8 lookfirst: self digitLength) asInteger
bitAnd: n | |
  n ← n asInteger.
  ↑self digitLogic: n op: #land: length: (self digitLength max: n digitLength)
bitOr: n | |
  n ← n asInteger.
  ↑self digitLogic: n op: #lor: length: (self digitLength max: n digitLength)
bitsAt: int | |
  ↑self bits: int start to: int stop
bitXor: n | |
  n ← n asInteger.
  ↑self digitLogic: n op: #lxor: length: (self digitLength max: n digitLength)
highBit | |
   "Return the index of the high order bit of the binary representation of this number"
  ↑self lastDigit hibit + (8 * (self digitLength - 1))

Aspects
∘ n | | <primitive: 63>
  self digitLength < n
    ifTrue: [↑0].
  ↑super ∘ n "user croak"
```

```
asStream | |
   ↑Stream new of: self
copyto: x | i |
   (1 to: (self digitLength min: x digitLength)) do:
      [:i| x↓i ← self↓i].
   ↑x
digitLength | | <primitive: 65>
   user croak
growby: n | |
   ↑self growto: self digitLength + n
growto: n | |
   ↑self copyto: (self species new: n)
lastDigit | |
   ↑self ↓ self digitLength
lastDigit← digit | |
   ↑self ↓ self digitLength ← digit
negative | |
   ↑false
positive | |
   ↑true
species | |
   ↑self class

Conversion
asFloat | |
    "Built for comfort, not for speed"
   ↑self asString asFloat
asInteger | |
   (self digitLength ≤ 2 and: [self ↓ 2 < maxHi])
     ifTrue: [↑(self ↓ 2 lshift: 8) + (self ↓ 1)].
   ↑self
asLarge | |
   ↑self
coerce: aNumber | |
   ↑aNumber asLarge
fromDoubleWord: s | |
   (self digitLength ≠ 4 or: [self negative]) ifTrue:
      [↑(self new: 4 neg: false) fromDoubleWord: s].
   self ↓ 1 ← s ↓ 4.
   self ↓ 2 ← s ↓ 3.
   self ↓ 3 ← s ↓ 2.
   self ↓ 4 ← s ↓ 1.
   ↑self asInteger
generality | |
   ↑40
isLarge | |
   ↑self

Printing
printOn: strm base: b | digits source dest i j t q rhi rlo |
   "Must have 2<=b<=16"
   i ← self digitLength.
   digits ← Stream default.
   dest ← (i ≤ 1
        ifTrue: [self]
        ifFalse: [String new: i]).
   source ← self.
```

```
    [i > 1] whileTrueDo:
        [rhi ← rlo ← 0.
        j ← i.
        [j > 0] whileTrueDo:
            [t ← (rlo lshift: 8) + (source o j).
            q ← (rhi lshift: 8) + (t / b).
            rhi ← q \ b.
            rlo ← t \ b.
            dest o j ← q / b.
            j ← j - 1].
        digits next← rlo asDigit.
        digits next← rhi asDigit.
        source ← dest.
        source o i = 0
          ifTrue: [i ← i - 1]].
    dest o 1 printOn: strm base: b.
    [t ← digits pop] whileTrueDo: [strm next← t]

Private arithmetic
digitAdd: arg | len i accum sum |
    accum ← 0.
    len ← self digitLength max: arg digitLength.
    sum ← self class new: len.
    i ← 1.
    [i ≤ len] whileTrueDo:
        [accum ← accum + (self o i) + (arg o i).
        sum o i ← accum land: 255.
        accum ← accum lshift: -8.
        i ← i + 1].
    accum ≠ 0
      ifTrue:
        [sum ← sum growby: 1.
        sum lastDigit← accum].
    ↑sum asInteger
digitCompare: arg | i len t4 t5 |
    len ← self digitLength.
    (t4 ← arg digitLength) < len
      ifTrue: [↑1].
    t4 > len
      ifTrue: [↑-1].
    (len to: 1 by: -1) do:
        [:i | (t5 ← arg o i) < (self o i)
          ifTrue: [↑1].
        t5 > (self o i)
          ifTrue: [↑-1]].
    ↑0
digitDiv: arg neg: ng | quo rem ql d div dh dnh dl qhi qlo i j k l hi lo r1 r2 r3 a t |
    l ← self digitLength "returns a vector of (quotient, remainder)" - arg digitLength +
1.
    l ≤ 0
      ifTrue: [↑{0 , self}].
    d ← 8 - arg lastDigit hibit.
    rem ← self digitLshift: d bytes: 0 lookfirst: false "makes a copy and shifts".
    div ← arg asLarge digitLshift: d bytes: 0 lookfirst: false "shifts so high order word is
≥128".
    quo ← self new: l neg: ng.
    dl ← div digitLength - 1 "last actual byte of data".
```

```
    ql ← l.
    dh ← div o dl.
    dnh ← (dl = 1
            ifTrue: [0]
            ifFalse: [div o (dl - 1)]).
    (1 to: ql) do:
       [:k |  "maintain quo*arg+rem=self"
       j ← rem digitLength "Estimate rem/div by dividing by dh." + 1 - k.
       r1 ← rem o j.
       r1 = dh
         ifTrue: [qhi ← qlo ← 15]
         ifFalse:
          [r2 ← rem o (j - 1) "First compute (r1,r2) = <qhi,qlo>*dh+t.  Note that r1,r2
are bytes, qhi,qlo are nibbles.".
          t ← (r1 lshift: 4) + (r2 lshift: -4).
          qhi ← t / dh.
          t ← (t \ dh lshift: 4) + (r2 land: 15).
          qlo ← t / dh.
          t ← t \ dh "Next compute (hi,lo) ← <qhi,qlo>*dnh".
          hi ← qhi * dnh.
          lo ← qlo * dnh + ((hi land: 15) lshift: 4).
          hi ← (hi lshift: -4) + (lo lshift: -8).
          r3 ← (j < 3
                  ifTrue: [0]
                  ifFalse: [rem o (j - 2)]) "Correct overestimate of q.  Max of 2 iterations
through loop -- see Knuth vol. 2".
          [(t < hi or: [t = hi and: [r3 < lo]]) and: [ "(t,r3) < (hi,lo)"
          qlo ← qlo - 1.
          lo ← lo - dnh.
          lo < 0
            ifTrue:
             [hi ← hi - 1.
             lo ← lo + 256].
          hi ≥ dh]] whileTrueDo: [hi ← hi - dh].
          qlo < 0
            ifTrue:
             [qhi ← qhi - 1.
             qlo ← qlo + 16]] "Subtract q*div from rem".
       l ← j - dl.
       a ← 0.
       (1 to: div digitLength) do:
          [:i | hi ← div o i * qhi.
          lo ← a + (rem o l) - ((hi land: 15) lshift: 4) - (div o i * qlo).
          rem o l ← lo land: 255.
          a ← (lo < 0
                  ifTrue: [(lo lshift: -8) lor: -256]
                  ifFalse: [lo lshift: -8]) - (hi lshift: -4).
          l ← l + 1].
       a < 0
         ifTrue:
          [ "Add div back into rem, decrease q by 1"
          qlo ← qlo - 1.
          l ← j - dl.
          a ← 0.
          (1 to: div digitLength) do:
             [:i | a ← (a lshift: -8) + (rem o l) + (div o i).
             rem o l ← a land: 255.
```

```
            l ← l + 1]].
        quo o (quo digitLength + 1 - k) ← (qhi lshift: 4) + qlo].
    rem ← rem digitRshift: d bytes: 0 lookfirst: dl.
    ↑{quo , rem}
digitMultiply: arg neg: ng | prod pl carry digit i j k xh xl low high |
    (arg o 1 = 0 and: [arg digitLength = 1])
      ifTrue: [↑0].
    pl ← self digitLength + arg digitLength.
    prod ← self new: pl neg: ng.
    (1 to: pl) do: [:i | prod o i ← 0].
    (1 to: self digitLength) do:
        [:i | (digit ← self o i) ≠ 0
          ifTrue:
            [k ← i.
            carry ← 0.
            xh ← digit lshift: -4.
            xl ← digit land: 15 "Loop invariant: 0≤carry≤0377, k=i+j-1".
            (1 to: arg digitLength) do:
                [:j | high ← arg o j "Do double-precision multiply in two parts. Integers
must be at least 13 bits for this to work." * xh.
                low ← arg o j * xl + ((high land: 15) lshift: 4) + carry + (prod o k).
                carry ← (high lshift: -4) + (low lshift: -8).
                prod o k ← low land: 255.
                k ← k + 1].
            prod o k ← carry]].
    prod o pl = 0
      ifTrue: [↑(prod growby: -1) asInteger].
    ↑prod asInteger
digitSubtract: arg | smaller larger i z sum sl al ng lastdigit |
    sl ← self digitLength.
    al ← arg digitLength.
    (sl = al
      ifTrue:
        [[self o sl = (arg o sl) and: [sl > 1]] whileTrueDo: [sl ← sl - 1].
        al ← sl.
        self o sl < (arg o sl)]
      ifFalse: [sl < al])
      ifTrue:
        [larger ← arg.
        smaller ← self.
        ng ← self negative ≡ false.
        sl ← al]
      ifFalse:
        [larger ← self.
        smaller ← arg.
        ng ← self negative].
    sum ← self new: sl neg: ng.
    lastdigit ← 1.
    z ← 0 "Loop invariant is -1≤z≤1".
    (1 to: sl) do:
        [:i | z ← z + (larger o i) - (smaller o i).
        (sum o i ← z land: 255) ≠ 0
          ifTrue: [lastdigit ← i].
        z ← (z < 0
              ifTrue: [-1]
              ifFalse: [z lshift: -8])].
    lastdigit = sl
```

```
    ifFalse: [sum ← sum growto: lastdigit].
  ↑sum asInteger

Private bit manipulation
digitLogic: arg op: op length: len | result i neg1 neg2 rneg z1 z2 rz b1 b2 b rdigits |
  neg1 ← self negative.
  neg2 ← arg negative.
  rneg ← ((neg1
      ifTrue: [-1]
      ifFalse: [0]) perform: op with: (neg2
      ifTrue: [-1]
      ifFalse: [0])) < 0.
  result ← self new: len neg: rneg.
  rz ← z1 ← z2 ← true.
  rdigits ← 1.
  (1 to: result digitLength) do:
    [:i | b1 ← self o i.
    neg1
      ifTrue: [b1 ← (z1
              ifTrue:
               [b1 = 0
                 ifTrue: [0]
                 ifFalse:
                  [z1 ← false.
                  256 - b1]]
              ifFalse: [255 - b1])].
    b2 ← arg o i.
    neg2
      ifTrue: [b2 ← (z2
              ifTrue:
               [b2 = 0
                 ifTrue: [0]
                 ifFalse:
                  [z2 ← false.
                  256 - b2]]
              ifFalse: [255 - b2])].
    b ← b1 perform: op with: b2.
    b = 0
      ifTrue: [result o i ← 0]
      ifFalse:
       [rdigits ← i.
       result o i ← (rneg
         ifTrue:
          [rz
            ifTrue:
              [rz ← false.
              256 - b]
            ifFalse: [255 - b]]
         ifFalse: [b])]].
  rdigits ≠ result digitLength
    ifTrue: [↑(result growto: rdigits) asInteger].
  ↑result asInteger
digitLshift: n bytes: b lookfirst: a | x f m len r i digit |
    "shift by 8*b+n bits, 0≤n<8. a true means check for a leading zero byte in the
result"
  x ← 0.
  f ← n - 8.
```

```
    m ← 255 lshift: 0 - n.
    len ← self digitLength + 1 + b.
    (a and: [(self lastDigit lshift: f) = 0])
      ifTrue: [len ← len - 1].
    r ← self class new: len.
    (1 to: b) do: [:i | r o i ← 0].
    (1 to: len - b) do:
       [:i | digit ← self o i.
       r o (i + b) ← ((digit land: m) lshift: n) lor: x "Avoid values > 8 bits".
       x ← digit lshift: f].
    ↑r
digitRshift: n bytes: b lookfirst: a | x i r f m digit |
    "shift right 8*b+n bits, 0≤n<8.  Discard all digits beyond a, and all zeroes at or
below a."
    n ← 0 - n.
    x ← 0.
    f ← n + 8.
    i ← a.
    m ← 255 lshift: 0 - f.
    digit ← self o i.
    [((digit lshift: n) lor: x) = 0 and: [i ≠ 1]] whileTrueDo:
       [x ← digit lshift: f "Can't exceed 8 bits".
       i ← i - 1.
       digit ← self o i].
    r ← self class new: i - b.
    a ← i.
    x ← self o (b + 1) lshift: n.
    (b + 1 to: a) do:
       [:i | digit ← self o (i + 1).
       r o (i - b) ← ((digit land: m) lshift: f) lor: x "Avoid values > 8 bits".
       x ← digit lshift: n].
    ↑r
]
SystemOrganization classify: #LargeInteger under: 'Numbers'.]
LargeInteger classInit]
```

"LargeNegInt"

```
Class new title: #LargeNegInt
   subclassof: LargeInteger
   fields: ''
   declare: 'minHi ';
   bytesize: 8;
   asFollows_]

A negative large integer.  Nearly all the work is done in LargeInteger:
LargeNegativeInteger just overrides a few messages.

Initialization
classInit | |
   minHi ← 0 - (0 minVal / 256)

Arithmetic
+ arg | |
   arg negative
     ifTrue: [↑self digitAdd: arg].
   ↑self digitSubtract: arg
- arg | |
   arg negative
     ifTrue: [↑self digitSubtract: arg].
   ↑self digitAdd: arg
abs | |
   ↑self negated
negated | |
   ↑self copyto: (self new: self digitLength neg: false)

Comparison
< arg | |
   arg negative
     ifTrue: [↑(self digitCompare: arg) > 0].
   ↑self
= arg | |
   (arg isNumber and: [arg negative])
     ifTrue: [↑(self digitCompare: arg) = 0].
   ↑false
compare: arg | |
   arg negative
     ifTrue: [↑3 - (self digitCompare: arg)].
   ↑1

Bit manipulation
highBit | |
   user notify: 'highBit not defined for negative numbers'

Conversion
asInteger | |
   (self digitLength ≤ 2 and: [self ∘ 2 < minHi or: [self ∘ 2 = minHi and: [self ∘ 1 =
0]]])
     ifTrue: [↑(0 - (self ∘ 2) lshift: 8) - (self ∘ 1)].
   ↑self

Printing
printOn: strm base: b | |
   strm append: '-'.
```

```
    super printOn: strm base: b

Aspects
negative | |
    ↑true
positive | |
    ↑false
_!
SystemOrganization classify: #LargeNegInt under: 'Numbers'._!
LargeNegInt classInit_!
```

"SmallInteger"

```
Class new title: #SmallInteger
    subclassof: Integer
    fields: ''
    declare: 'digitbuffer ';
    asFollows_]
```

Integers are 15 or 16-bit numbers, stored in two-s complement form. The allowable range is from 0 minVal to 0 maxVal. You can type them in octal by typing a leading radix specification, as in 8r377.

```
Arithmetic
≤ aNumber | | "primitive"
    ↑self retry: #≤ coercing: aNumber
≠ aNumber | | "primitive"
    aNumber isNumber ifFalse: [↑true].
    ↑self retry: #≠ coercing: aNumber
≥ aNumber | | "primitive"
    ↑self retry: #≥ coercing: aNumber
** aNumber | | "primitive" <primitive: 9>
    (aNumber is: SmallInteger)
      ifTrue: [↑self asLarge * aNumber]. "overflow"
    ↑self retry: #* coercing: aNumber
+ aNumber | | "primitive"
    (aNumber is: SmallInteger)
      ifTrue: [↑self asLarge + aNumber]. "overflow"
    ↑self retry: #+ coercing: aNumber
- aNumber | | "primitive"
    (aNumber is: SmallInteger)
      ifTrue: [↑self asLarge - aNumber]. "overflow"
    ↑self retry: #- coercing: aNumber
/ arg | | "primitive" <primitive: 10>
    0 = arg
      ifTrue: [user notify: 'Attempt to divide by 0']
      ifFalse:
       [arg isLarge
         ifTrue: [↑self asLarge / arg].
        ↑self / arg asInteger]
// aNumber | d | "div"
    d ← self quo: aNumber. "temporary crock"
    (self \\aNumber) = 0
      ifTrue: [↑d].
    (self < 0) ~= (aNumber < 0)
      ifTrue: [↑d - 1].
    ↑d
< aNumber | | "primitive"
    ↑self retry: #< coercing: aNumber
= aNumber | | "primitive"
    aNumber isNumber ifFalse: [↑false].
    ↑self retry: #= coercing: aNumber
> aNumber | | "primitive"
    ↑self retry: #> coercing: aNumber
compare: arg | |
    (arg is: SmallInteger)
      ifTrue:
       [self < arg
         ifTrue: [↑1].
```

```
      self = arg
        ifTrue: [↑2].
      ↑3]
    ifFalse: [↑4 - (arg compare: self)]
intdiv: arg | |
  (arg is: LargeInteger)
    ifTrue: [↑self asLarge intdiv: arg].
  ↑super intdiv: arg
quo: aNumber | |
  (aNumber moreGeneralThan: self)
    ifTrue: [↑self retry: #quo: coercing: aNumber].
  ↑self / aNumber "invoke integer divide primitive"
\ arg | | "primitive" <primitive: 11>
  "mod"
  0 = arg
    ifTrue: [user notify: 'Attempt to divide by 0']
    ifFalse:
     [arg isLarge
       ifTrue: [↑self asLarge \ arg].
      ↑self \ arg asInteger]
\\ aNumber | m |
  m ← self \aNumber. "invoke integer modulo primitive"
  m = 0
    ifTrue: [↑m].
  (self < 0) ~= (aNumber < 0)
    ifTrue: [↑m - aNumber].
  ↑m

Bit Manipulation
bitAnd: arg | |<primitive: 15>
  ↑arg bitAnd: self
bitOr: arg | |<primitive: 16>
  ↑arg bitOr: self
bitXor: arg | |<primitive: 14>
  ↑arg bitXor: self
highBit | i |
  (1 to: 16) do:
     [:i | (self land: biton o (17 - i)) ≠ 0
       ifTrue: [↑17 - i]].
  ↑0

Conversion
asFloat | |<primitive: 62>
  user croak
asInteger | |
  ↑self
asLarge | digits |
   "convert to LargeInteger"
  digits ← (self < 0 ifTrue: [LargeNegInt] ifFalse: [LargeInteger]) new: self digitLength.
  digits o 1 ← self o 1.
  digits length = 2
    ifTrue: [digits o 2 ← self o 2].
  ↑digits
inString | t |
  t ← String new: 1.
  t o 1 ← self.
  ↑t
```

```
Subscripts
instfield: i | |
    "small integer gives trouble"
  i = 1
    ifTrue: [↑self].
  user notify: 'arg too big'

Printing
printOn: aStream | |
  self < 0
    ifTrue:
        [aStream append: '-'.
        self negated printOn: aStream base: 10]
    ifFalse: [self printOn: aStream base: 10]
printOn: aStream base: b | i j x |
  (x ← self) < 0
    ifTrue:
     [i ← 1.
     digitbuffer o 1 ← 16384 \ b * 2 + self - ¯32768 \ b "Im trying to print 16-bit
nos.".
     x ← 16384 / b * 2 + (self - ¯32768 / b)]
    ifFalse: [i ← 0].
  [x ≥ b] whileTrueDo:
     [digitbuffer o (i ← i + 1) ← x \ b.
     x ← x / b].
  digitbuffer o (i ← i + 1) ← x.
  [i > 0] whileTrueDo:
     [x ← digitbuffer o i.
     aStream next← (x < 10
       ifTrue: [48 + x]
       ifFalse: [55 + x]).
     i ← i - 1]

Characters
asDigit | |
  self < 10 ifTrue: [↑self + 48 "0"] ifFalse: [↑self + 55 "A -10"]
asUppercase | |
  97 ≤ self
    ifTrue:
     [self ≤ 122
       ifTrue: [↑self - 32]]
isdigit | |
  self ≥ 48
    ifTrue: [ " 0 "
     ↑self ≤ 57].
   " 9 "
  ↑false
isDigitRadix: radix | |
  ↑(self - (self ≥ 65 ifTrue: [55] ifFalse: [48]) between: 0 and: radix-1)
isletter | |
  self ≥ 97
    ifTrue: [ " a "
     ↑self ≤ 122].
   " z "
  self ≥ 65
    ifTrue: [ " A "
```

```
      ↑self ≤ 90].
    " Z "
  ↑false
tokenish | |
    "test for token-chars"
  self isletter
    ifTrue: [↑true].
   "lower-case"
  self isdigit
    ifTrue: [↑true].
   "digits"
  ↑'_:$' has: self

Copying and Purging
copy | |
  ↑self
recopy | |
  ↑self

Initialization
classInit | |
    "Initialize the digit buffer"
  digitBuffer ← String new: 16

LargeInteger Compatibility
o n | t |
  n = 1 "behave like a Natural"
    ifTrue:
     [self < 0
       ifTrue: [↑((self land: 255) lxor: 255) + 1 land: 255].
      ↑self land: 255]
    ifFalse:
     [n = 2
       ifTrue:
        [self < 0
          ifTrue:
           [t ← (self lshift: -8) lxor: 255.
           (self land: 255) = 0
             ifTrue: [↑t + 1 land: 255].
           ↑t]
          ifFalse: [↑self lshift: -8]]
       ifFalse: [↑0]]
digitLength | |
  (self ≥ 256 "behave like a Natural" or: [self ≤ -256])
    ifTrue; [↑2].
  ↑1
last | |
  ↑self o self length
lastDigit | |
  ↑self o self digitLength
length | |
  (self ≥ 256 "behave like a Natural" or: [self ≤ -256])
    ifTrue: [↑2].
  ↑1

Backward compatibility
land: arg | |<primitive: 15>
```

```
    ↑arg land: self
lor: arg | |<primitive: 16>
    ↑arg lor: self
lshift: arg | |<primitive: 13>
    arg ← arg asInteger.
    arg class ≡ SmallInteger
      ifTrue: [↑self asLarge lshift: arg].
    user croak
lxor: arg | |<primitive: 14>
    ↑arg lxor: self

As yet unclassified
asOop | |
    self ≥ 0 ifTrue: [↑32768 + self]. "0...16K-1 --> 32K...48K-1"
    ↑65536 + self          "-16K...-1 --> 48K...64K-1"
bits: left to: right ← val | width mask |
     "  0 is leftmost bit, 15 is rightmost"
    width ← right - left + 1.
    mask ← (width < 15
            ifTrue: [biton o (width + 1) - 1]
            ifFalse:
             [width = 15
               ifTrue: [32767]
               ifFalse: [¯1]]).
    ↑(self land: ((mask lshift: 15 - right) lxor: ¯1)) lor: ((val land: mask) lshift: 15 -
right)
coerce: n | | ↑n asInteger
elementStream | |
    ↑Stream new of: (Vector new: self)
generality | | ↑ 20
inUniqueString | |
    self < 128
      ifTrue: [↑UST1 o (self + 1)].
    ↑self inString unique
maxVal | |
    ↑16383
minVal | |
    ↑¯16384
_!
SystemOrganization classify: #SmallInteger under: 'Numbers'._!
SmallInteger classInit_!
```

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:26:19 pm.'_!
                                                        "Array"
Class new title: #Array
   subclassof: Object
   fields: ''
   declare: '';
   asFollows_!
```

Array is an abstract class in the sense that it has no state, and instantiation is consequently not meaningful. However it defines the default message set inherited by its subclasses, notably String, Vector, and UniqueString. Notice that subscripting is not done here, except to handle the exceptional cases such as subscripting by other types as in ao(1 to: 3).

```
Reading and Writing
< v | |
    "for sorting vectors by first element"
   ↑self o 1 < (v o 1)
= arg | x |
   arg isArray
     ifTrue:
      [self length ≠ arg length
        ifTrue: [↑false].
      (1 to: self length) do:
         [:x | self o x = (arg o x)
           ifFalse: [↑false]].
      ↑true]
     ifFalse: [↑false]
> v | |
    "for sorting vectors by first element"
   ↑self o 1 > (v o 1)
all← val | i |
   (1 to: self length) do: [:i | self o i ← val]
last | |
   ↑self o self length
last← val | |
   ↑self o self length ← val

Copying and Altering
+ arg | |
   ↑self concat: arg
collect: aBlock | i len result |
   result← self species new: (len← self length).
   i←0.
   [(i←i+1) ≤ len]
      whileTrueDo: [resultoi← aBlock value: selfoi].
   ↑result
concat: arg | x s |
   x ← self species new: self length + arg length.
   self copyto: (s ← x asStream).
   arg copyto: s.
   ↑x
copy | |
   ↑self copy: 1 to: self length
copy: a to: b | |
   ↑self copy: a to: b to: (self species new: b - a + 1)
copy: a to: b to: t | i s me |
```

Basic Data Structures

```
    s ← t asStream.
    me ← Stream new of: self from: a to: b.
    (a to: b) do: [:i |  "general code wont stop at false"
       s next← me next].
    ↑t
copyto: t | |
    ↑self copy: 1 to: self length to: t
delete: obj | s each |
    s ← (self species new: self length) asStream.
    self do:
       [:each | obj = each
         ifFalse: [s next← each]].
    ↑s contents
do: aBlock | i len |
    len← self length.  i←0.
    [(i←i+1) ≤ len] whileTrueDo: [aBlock value: self∘i]
grow | |
    ↑self growto: (4 max: self length * 2)
growby: n | |
    ↑self growto: self length + n
growto: n | |
    ↑self copy: 1 to: self length "copyto:" to: (self species new: n)
insertSorted: x | a c i |
     "self is assumed to be sorted"
    i ← self findSorted: x.
    c ← (a ← self species new: self length + 1) asStream.
    self ∘ (1 to: i) copyto: c.
    c next← x.
    self ∘ (i + 1 to: self length) copyto: c.
    ↑a
notNil | t i |
     "copy self (which contains no falses) removing all nils"
    t ← (self species new: self length - (self count: nil)) asStream.
    self do:
       [:i | i ≡ nil
         ifFalse: [t next← i]].
    ↑t asArray
replace: a to: b by: s | x xs |
    x ← self species new: self length + s length - (1 + b - a).
    xs ← x asStream.
    self copy: 1 to: a - 1 to: xs.
    s copy: 1 to: s length to: xs.
    self copy: b + 1 to: self length to: xs.
    ↑x
without: index | s me i |
     "if index in range, return self without ∘index"
    (index cansubscript: self)
      ifTrue:
       [s ← (self species new: self length - 1) asStream.
       me ← self asStream.
       (1 to: self length) do:
          [:i | i = index
            ifTrue: [me next]
            ifFalse: [s next← me next]].
       ↑s asArray]
```

Searching

```
all: variable suchThat: expr | s i x |
    "a copy of some of me"
  s ← (self species new: self length) asStream.
  (1 to: self length) do:
    [:i | x ← self ∘ i.
    variable value← x.
    expr eval
      ifTrue: [s next← x]].
  ↑s contents
count: x | i n |
  n ← 0.
  (1 to: self length) do:
    [:i | x = (self ∘ i)
      ifTrue: [n ← n + 1]].
  ↑n
find: x | i |
  (1 to: self length) do:
    [:i | self ∘ i = x
      ifTrue: [↑i]].
  ↑0
findSorted: x | lo mid hi |
    " returns index of largest element ≤ x "
  hi ← self length + 1.
  lo ← 1.
  [lo < hi] whileTrueDo:
    [ "binary search; self must be sorted"
    self ∘ (mid ← lo + hi / 2) > x
      ifTrue: [hi ← mid]
      ifFalse: [lo ← mid + 1]].
  ↑hi - 1 " 0≤result≤length "
has: x | |
  ↑(self find: x) ≠ 0

Permutation
promote: t | n |
  n ← self find: t.
  n = 0
    ifFalse:
     [self ∘ (n to: 2 by: ¯1) ← self ∘ (n - 1 to: 1 by: ¯1).
     self ∘ 1 ← t]
reverse | |
  ↑Substring new data: self map: (self length to: 1 by: ¯1)
sort | |
  self sort: 1 to: self length "Permute my elements so they are sorted nondescending.
Note: if I am a substring, only my map will be permuted.  In certain situations, this
may not be what you expect."
sort: i to: j | di dij dj tt ij k l n |
  (n ← j + 1 "Sort elements i through j of self to be nondescending." "The prefix d
means the data at." - i) ≤ 1
    ifFalse:
     [ "Nothing to sort." "Sort di,dj."
     di ← self ∘ i.
     dj ← self ∘ j.
     di > dj
       ifTrue:
        [self swap: i with: j.
        tt ← di.
```

```
        di ← dj.
        dj ← tt].
      n = 2
        ifFalse:
        [ "They are the only two elements."
        ij ← i + j lshift: ¯1 "ij is the midpoint of i and j." "Sort di,dij,dj.  Make dij be
their median.".
        dij ← self o ij.
        di > dij
          ifTrue:
           [self swap: i with: ij.
           dij ← di]
          ifFalse:
           [dj < dij
             ifTrue:
              [self swap: j with: ij.
              dij ← dj]].
        n = 3
          ifFalse:
           [ "They are the only three elements." "Find k>i and l<j such that dk,dij,dl
are in reverse order.  Swap k and l.  Repeat this procedure until j and k pass each
other."
           k ← i.
           l ← j.
           [[self o (l ← l - 1) > dij] whileTrueDo: [].
           [self o (k ← k + 1) < dij] whileTrueDo: [].
           k ≤ l] whileTrueDo: [self swap: k with: l "Now l<k (either 1 or 2 less), and di
through dl are all less than dk through dj.  Sort those two segments."].
           self sort: i to: l.
           self sort: k to: j]]]
swap: i with: j | t |
   t ← self o i.
   self o i ← self o j.
   self o j ← t

Conversion
asStream | |
   ↑Stream new of: self
transform: each to: expr | s i |
    "a copy of me with each element transformed"
   s ← (self species new: self length) asStream.
   (1 to: self length) do:
      [:i | each value← self o i.
      s next← expr eval].
   ↑s asArray

Mapping
cansubscript: a | i |
   self do:
      [:i | (i cansubscript: a)
        ifFalse: [↑false]]
subscripts: x | |
    "subarrays"
   ↑Substring new data: x map: self
subscripts: x ← val | |
    "subrange replacement"
   self length = val length
```

```
        ifTrue: [user notify: 'lengths not commensurate']
        ifFalse:
         [val copyto: (Substring new data: x map: self).
         ↑val]

Compatibility
isArray | |

isIntervalBy1 | |
   ↑false
species | |
   ↑Vector

Comparing
hash | |
    "make sure = arrays hash =ly"
   self length = 0
     ifTrue: [↑17171].
   ↑(self o 1) hash + (self o self length) hash
_]
SystemOrganization classify: #Array under: 'Basic Data Structures'._]
```

"Interval"

```
Class new title: #Interval
   subclassof: Array
   fields: 'start stop step length'
   declare: '';
   asFollows_!
```

I am an arithmetic progression from start in steps of step, not exceeding stop

```
Initialization
from: t1 to: t2 by: t3 | |
   start ← t1.
   stop ← t2.
   step ← t3.
   length ← 1 + (stop - start / step).
   step < 0
     ifTrue:
      [start < stop
        ifTrue: [length ← 0]]
     ifFalse:
      [stop < start
        ifTrue: [length ← 0]]

Reading and Writing
o x | |
   ((x isNumber) and: [x≥1 and: [x≤length]])
     ifTrue: [↑start + (step * (x - 1))]
     ifFalse: [↑super o x]
o x ← val | |
   user notify: 'Intervals are not for writing into'
do: aBlock | var |
   var← start-step.
   step>0
      ifTrue: [[(var← var+step)≤stop] whileTrueDo: [aBlock value: var]]
      ifFalse: [[(var← var+step)≥stop] whileTrueDo: [aBlock value: var]]
length | |
   ↑length
start | |
   ↑start
stop | |
   ↑stop

Compatibility
= int | |
   ↑start = int start and: [(stop = int stop and: [length = int length])]
canSubscript: a | |
   ↑length ≤ 0 or: [((start canSubscript: a) and: [(length - 1 * step + start
canSubscript: a)])]
hash | |
   ↑(((start lshift: 2) lxor: stop) lshift: 1) lxor: length
isIntervalBy1 | |
   ↑step = 1
_!
SystemOrganization classify: #Interval under: 'Basic Data Structures'._!
```

"Stream"

```
Class new title: #Stream
   subclassof: Object
   fields: 'array position limit'
   declare: '';
   asFollows_]
```

Streams provide fast sequential access to arrays (implemented in microcode for Strings and Vectors). A subclass can handle end conditions if desired (disk files do this).

```
Initialization
close | |
   limit ← position.
   position ← 0
default | |
   self of: (String new: 16)
of: t1 | |
   array ← t1.
   position ← 0.
   limit ← array length
of: t1 from: pos to: lim | len |
   array ← t1.
   limit ← (lim > (len ← array length)
         ifTrue: [len]
         ifFalse: [lim]).
   position ← (pos ≤ 1
         ifTrue: [0]
         ifFalse: [pos - 1])
release | |
   array ← nil

Sequential reading and writing
◁ x | y |
   (y ← self next)
    ifTrue:
     [ "peek for matching element"
     x = y
       ifTrue: [↑y].
      "gobble it if found"
     position ← position - 1.
     ↑false]
    ifFalse: [↑false]
append: x | i |
    "Array arg"
   x do: [:i | self next← i].
   ↑x
dequeue | |
    "use it as a FIFO"
   ↑self dequeue: 1
dequeue: n | t |
   position < n
     ifTrue: [↑false].
   t ← (array ○ (1 to: n)) copy.
   array ○ (1 to: position - n) ← array ○ (n + 1 to: position).
   position ← position - n.
   ↑t
```

```
integerScan | sign base maxdigit c val |
    "get the next Integer or LargeInteger (Float?) from a Stream.
   copied from String asInteger"
   sign ← (self ◁ 21  "-"
          ifTrue: [-1]
          ifFalse: [1]).
   base ← (self ◁ 48  "0"
          ifTrue: [8]
          ifFalse: [10]).
   maxdigit ← 48 + base.
   val ← 0.
   [(c ← self next) and: [c ≥ 48 and: [c < maxdigit]]] whileTrueDo:
      [val ← val * base + (c - 48)].
   c
     ifTrue: [self skip: -1].
   (base = 8 and: [val > 32767 and: [sign = 1 and: [val < 65536]]])
     ifTrue: [↑val - 65536]. "Special hack for 16-bit octal numbers"
   ↑(val * sign) asInteger
into: x | |
   ↑self into: x endError: true "generate an error if the Stream is exhausted before x is
filled"
into: x endError: err | i t len |
   i ← 0 "Array result".
   len ← x length "read until count or stream is exhausted".
   [i < len and: [(t ← self next)]] whileTrueDo: [x ∘ (i ← i + 1) ← t].
   err
     ifTrue:
      [t
        ifTrue: [↑x].
       user notify: 'only read first ' + i asString]
     ifFalse: [ "return number that were read"
       ↑i]
next | |<primitive: 20>
    "simple result"
   self myend
     ifTrue: [↑self pastend].
   ↑array ∘ (position ← position + 1)
nextElement: element | |
   self next← element
nextString | len |
   ↑self into: (String new: ((len ← self next) < 192
     ifTrue: [len]
     ifFalse: [ "up to 191 chars (BCPL compat)"
       len - 192 * 256 + self next])) endError: true "up to 16383 chars"
next← x | |<primitive: 21>
    "simple arg"
   self myend
     ifTrue: [↑self pastend← x].
   ↑array ∘ (position ← position + 1) ← x
peek | x |
   (x ← self next)
     ifTrue:
      [position ← position - 1.
       ↑x].
    "peek at next element"
   ↑false
pop | |
```

```
    "use it as a LIFO"
    position < 1
      ifTrue: [↑false].
    position ← position - 1.
    ↑array o (position + 1)
  pop: n | t |
    position < n
      ifTrue: [↑false].
    t ← self last: n.
    position ← position - n.
    ↑t
  upto: x | y s |
    s ← (String new: 250) asStream.
    self do:
      [:y | y = x
        ifTrue: [↑s contents].
      s next← y].
    ↑s contents

  Test and alter position
  empty | |
    "for"
    ↑position = 0
  end | |
    ↑position ≥ limit
  limit | |
    ↑limit
  myend | |
    ↑position ≥ limit
  pastend | |
    ↑false
  pastend← x | |
    array ← array grow.
    limit ← array length.
    ↑self next← x
  position | |
    ↑position
  position← ti | |
    position ← ti
  reset | |
    position ← 0
  settoend | |
    position ← limit
  skip: x | |
    position ← position + x
  skipTo: x | y |
    self do:
      [:y | y = x
        ifTrue: [↑true]].
    ↑false

  Static reading and writing
  o x | |
    ↑array o x
  o x ← val | |
    ↑array o x ← val
  contents | |
```

```
    ↑array copy: 1 to: position
insert: x | i |
    "treat as LIFO queue, insert in front"
    "grow array if necessary"
    position = limit
      ifTrue:
       [array ← array grow.
       limit ← array length].
    (1 to: position) do: [:i | array o (position - i + 2) ← array o (position - i + 1)].
    array o 1 ← x.
    position ← position + 1
last | |
    position ≠ 0
      ifTrue: [↑array o position]
last: n | |
    ↑(array o (position - n + 1 to: position)) copy
rest | |
    ↑array copy: position + 1 to: limit

Character printing
cr | |
    self next← 13
crtab: n | i |
    self next← 13.
    (1 to: n) do: [:i | self next← 9]
print: obj | |
    obj printOn: self
space | |
    self next← 32
tab | |
    self next← 9

Coercions
asArray | |
    ↑array
asStream | |

asVector | | "Convert a string to a vector of tokens"
    ↑LADScanner new scanTokens: self

As yet unclassified
do: aBlock | |
    [self end] whileFalseDo: [aBlock value: self next].
    ↑nil
reverseContents | a s i |
    a ← array species new: position.
    s ← a asStream.
    (position to: 1 by: ¯1) do: [:i | s next← array o i].
    ↑a
]
SystemOrganization classify: #Stream under: 'Basic Data Structures'.]
```

"String"

```
Class new title: #String
   subclassof: Array
   fields: ''
   declare: 'StringBlter ';
   bytesize: 8;
   asFollows_]
```

I am an array of bytes, integers between 0 and 255 usually representing ascii characters .

```
Reading and Writing
all← val | |
   self fill: 1 to: self length with: val
fill: a to: b with: val | i |
   (a to: b) do: [:i |  "eventually use BitBlt?"
      self o i ← val]
word: x ← y | |
    "write word in String"
   self o (x + x - 1) ← y lshift: ¯8.
   self o (x + x) ← y land: 255.
   ↑y

Copying and Altering
concat: s | len |
   ((len ← self length) + s length > 20 and: [(s ls: String)])
     ifTrue: [↑self replace: len + 1 "this concatenates more quickly if BitBlt is used" to:
len by: s from: 1 to: s length].
   ↑super concat: s
copy: a to: b | |
   ↑(self species new: 1 + b - a) copy: 1 to: 1 + b - a with: self from: a to: b
copy: a to: b with: s from: c to: d | i |
   ((b - a "like replace, except in place. self o(a to: b) ← so(c to: d).
   use BitBlt unless size too small, StringBlter≡false, or index/sizes too large" > 12 and:
[StringBlter]) and: [(BitBlt new stringCopy: self from: a to: b with: s from: c to: d)])
     ifFalse:
      [(self ≡ s and: [(c < a and: [d ≥ a])])
        ifTrue: [(b - a "overlap of second range with below first in same string.
       copy in reverse order: self o(b to: a by: ¯1) ← self o(d to: c by: ¯1)" to: 0 by: ¯1)
do: [:i | self o (a + i) ← self o (c + i)]]
        ifFalse: [s copy: c to: d to: (Stream new of: self from: a to: b)]]
findString: str startingAt: start | i t |
   str length = 0
     ifTrue: [↑0].
   t ← str o 1.
   (start to: self length - str length + 1) do:
      [:i | self o i = t
        ifTrue:
         [self o (i to: i + str length - 1) = str
           ifTrue: [↑i]]].
   ↑0
growto: n | len |
   (len ← self length) ≤ n
     ifFalse: [len ← n].
   ↑(self species new: n) copy: 1 to: len with: self from: 1 to: len
recopy | |
   ↑self copy
```

```
replace: a to: b by: s | |
   (s Is: String)
     ifTrue: [↑self replace: a to: b by: s from: 1 to: s length].
   ↑self replace: a to: b by: s asArray from: 1 to: s position
replace: a to: b by: r from: c to: d | s t |
   s ← self species new: self length + (d - c) - (b - a) "use BitBlt unless
StringBlter≡false or index/sizes too large".
   (StringBlter and: [(BitBlt new stringReplace: s with: self from: a to: b and: [r] from:
c to: d)])
     ifTrue: [↑s].
    "see Array concat:"
   t ← Stream new of: s.
   self copy: 1 to: a - 1 to: t.
   r copy: c to: d to: t.
   self copy: b + 1 to: self length to: t.
   ↑s
subst: repl for: key | key1 i nskip result |
   nskip ← 0.
   key1 ← key o 1.
   result ← Stream default.
   (1 to: self length) do:
     [:i |  " the Boyer Slow string replacement "
     nskip > 0
       ifTrue: [nskip ← nskip - 1]
       ifFalse:
        [self o i = key1
          ifTrue:
           [self o (i to: (self length min: i + key length - 1)) = key
             ifTrue:
              [result append: repl.
               nskip ← key length - 1]
             ifFalse: [result next← self o i]]
          ifFalse: [result next← self o i]]].
   ↑result contents

Comparison
- s | i c ldiff |
   (1 to:  "Return a negative, zero, or positive integer as I compare < = or > s" "The
collation sequence is ascii with case differences ignored."
   ((ldiff ← self length - s length) < 0
     ifTrue: [self length]
     ifFalse: [s length]]) do:
      [:i | (c ← UpperCase o (self o i + 1) - (UpperCase o (s o i + 1))) ≠ 0
        ifTrue: [↑c]].
   ↑ldiff
< s | |
   ↑(self compare: s) "Return true iff I collate before s.  The collation sequence is ascii
with case differences ignored." = 1
> s | |
   ↑(self compare: s) "Return true iff I collate after s.  The collation sequence is ascii
with case differences ignored." = 3
compare: s | i len lcomp u1 u2 t7 |
   lcomp ← ((t7 ← self length) < (len ← s length)
          ifTrue:
           [len ← self length.
           1]
          ifFalse:
```

```
            [t7 = len
              ifTrue: [2]
              ifFalse: [3]]).
    (1 to: len) do:
       [:i | (u1 ← UpperCase o (self o i + 1)) = (u2 ← UpperCase o (s o i + 1))
         ifFalse:
          [u1 < u2
            ifTrue: [↑1].
           ↑3]].
    ↑lcomp
hash | l m |
    (l ← m ← self length) ≤ 2
      ifTrue:
       [l = 2
         ifTrue: [m ← 3]
         ifFalse:
          [l = 1
            ifTrue: [↑(self o 1 land: 127) * 106].
           ↑21845]].
    ↑self o 1 * 48 + (self o (m - 1) + l)
match: text | star pound pattern scanning p t back |
    star ← 42 "*".
    pound ← 35 "#".
    pattern ← self asStream.
    text ← text asStream.
    scanning ← false.
    [true] whileTrueDo:
       [(p ← pattern next) = star
         ifTrue:
          [pattern end
            ifTrue: [↑true].
           scanning ← pattern position]
         ifFalse:
          [(t ← text next) ≡ false
            ifTrue: [↑t ≡ p].
           p ≡ false
            ifTrue:
             [scanning
               ifTrue:
                [back ← scanning - pattern position.
                 pattern skip: back.
                 text skip: back]
               ifFalse: [↑false]]
            ifFalse:
             [(UpperCase o (t + 1) = (UpperCase o (p + 1)) or: [p = pound])
               ifFalse:
                [scanning
                  ifTrue:
                   [back ← scanning - pattern position.
                    pattern skip: back.
                    text skip: back + 1]
                  ifFalse: [↑false]]]]]

Conversion
asDecimalDigits | strm sign c val |
    "Not asInteger, because the result may be a Float if it's too big"
    strm ← Stream new of: self.
```

```
    sign ← strm ◁ 21.
    val ← (self length > 4
           ifTrue: [0.0]
           ifFalse: [0]).
    strm do:
       [:c | (c < 48 or: [c > 57])
         ifTrue: [user notify: self + ' isn''t a valid integer']
         ifFalse: [val ← val * 10 + (c - 48)]].
    sign
      ifTrue: [↑val * ¯1].
    ↑val
asFileName | |
    ↑dp0 checkName: self fixing: true
asFloat | strm int frac exp |
    strm ← Stream new of: self.
    int ← strm upto: 46.
    frac ← strm upto: 101.
    exp ← strm rest asInteger - frac length.
    int ← (int concat: frac) asDecimalDigits asFloat.
    exp = 0
      ifTrue: [↑int].
    exp > 0
      ifTrue: [↑int * (10.0 ipow: exp)].
    ↑int / (10.0 ipow: 0 - exp)
asInteger | sign base maxdigit c val i i1 i2 |
    self length = 0
      ifTrue: [↑0].
    i1 ← 1.
    i2 ← self length.
    sign ← (self o 1 = 21
           ifTrue:
            [i1 ← 2.
             ¯1]
           ifFalse: [1]).
    base ← (self o self length = 72
           ifTrue:
            [i2 ← i2 - 1.
             16]
           ifFalse:
            [self o i1 = 48
              ifTrue: [8]
              ifFalse: [10]]).
    maxdigit ← (base = 16
           ifTrue: [70]
           ifFalse: [47 + base]).
    val ← (self length > 4
           ifTrue: [0.0]
           ifFalse: [0]).
    (i1 to: i2) "octal and hex packed into 16-bit integers" do:
       [:i | c ← self o i.
       (c < 48 or: [c > maxdigit])
         ifTrue: [user notify: self + ' isn''t a valid integer']
         ifFalse: [val ← val * base + (c ≤ 57
                  ifTrue: [c - 48]
                  ifFalse: [c - 55])]].
    val > 32767
```

```
    ifTrue:
     [(base = 8 or: [base = 16])
       ifTrue:
        [sign = 1
          ifTrue:
           [val < 65536
             ifTrue: [↑(val - 65536) asInteger].
           user notify: 'Warning: Octal or Hex number exceeds 16 bits']]].
   ↑(val * sign) asInteger
asParagraph | |
   ↑Paragraph new text: self alignment: 0
asUppercase | s c |
   s ← Stream default.
   self do: [:c | s next← UpperCase o (c + 1)].
   ↑s contents
asVector | |
   ↑self asStream asVector
hasBeenUniqued | |
   ↑#a hasInterned: self
printOn: strm | x |
    "print inside string quotes"
   strm next← 39.
   self do:
      [:x | strm next← x.
      x = 39
        ifTrue: [strm next← x]].
   strm next← 39 "imbedded quotes get doubled"
unique | u |
    "copy and intern"
   ↑#a intern: self

Compatibility
species | |
   ↑String
stringhash | |
   ↑self hash
!
SystemOrganization classify: #String under: 'Basic Data Structures'.!
```

"Substring"

```
Class new title: #Substring
   subclassof: Array
   fields: 'data map'
   declare: '';
   asFollows_]
```

I am an array that consists of a set of elements (specified by map) of an array (data)

```
Initialization
data: t1 map: t2 | |
   data ← t1.
   map ← t2

Reading and Writing
o x | |
   ↑data o (map o x)
o x ← val | |
   ↑data o (map o x) ← val
length | |
   ↑map length

Copying and Altering
swap: i with: j | t |
   t ← map o i "By permuting my map (a writable Array), swap elements i and j.".
   map o i ← map o j.
   map o j ← t

Conversion
asStream | |
   map isIntervalBy1
     ifTrue: [ "direct stream for simple substrings"
       ↑Stream new of: data from: map start to: map stop].
   ↑Stream new of: self from: 1 to: map length

Compatability
species | |
   ↑data species
_]
SystemOrganization classify: #Substring under: 'Basic Data Structures'._]
```

"UniqueString"

```
Class new title: #UniqueString
   subclassof: String
   fields: ''
   declare: '';
   bytesize: 8;
   asFollows_]
```

I am a string that is unequal to every other instance of my subclass

```
Initialization
classInit | i a v |
    "make up table of 1-char atoms"
   v ← Vector new: 128.
   a ← String new: 1.
   (1 to: 128) do:
      [:i | a o 1 ← i - 1.
      v o i ← a unique].
   UST1 ← v
hasInterned: s | i v |
    "↑false if String s hasnt been interned, else ↑s unique"
   s length = 1
     ifTrue:
      [s o 1 < 128
        ifTrue: [↑UST1 o (s o 1 + 1)]].
   v ← USTable o (s stringhash \ USTable length + 1).
   (1 to: v length) do:
      [:i | v o i ≡ nil
        ifFalse:
         [s length = (v o i) length
           ifTrue:
            [s = (v o i)
              ifTrue: [↑v o i]]]].
   ↑false
intern: s | ustr h |
   (ustr ← self hasInterned: s)
     ifTrue: [↑ustr].
    "check if already exists"
   ustr ← ((s is: UniqueString)
          ifTrue: [s]
          ifFalse: [ "create a new one"
           (UniqueString new: s length) str: s]).
   h ← s stringhash \ USTable length + 1.
   USTable o h ← {USTable o h , ustr}.
   ↑ustr
str: s | j |
   (1 to: s length) do: [:j | super o j ← s o j].
   ↑self
unique | |


Reading and Writing
o x ← val | |
   user notify: 'UniqueStrings are not for writing into'

Selectors
isarrow | |
```

```
    "ends with ←"
   self length ≤ 1
     ifTrue: [↑false].
   ↑self o self length = 95
isinfix | x |
   ↑(self o 1) isletter ≡ false
iskeyword | x |
    "ends with colon"
   self length ≤ 1
     ifTrue: [↑false].
   x ← self o self length.
   x = 58
     ifTrue: [↑true].
   ↑x = 3
keywords | result strm i l char colon ocolon |
    "return a vector of the keywords that compose me"
   'o←' = self
     ifTrue: [↑#('o' '←' )].
   result ← (Vector new: 10) asStream.
   strm ← Stream default.
   colon ← ':' o 1.
   ocolon ← '§' o 1.
   i ← 1.
   l ← self length.
   [i ≤ l] whileTrueDo:
      [char ← self o i.
      strm append: char.
      ((char = colon or: [char = ocolon]) or: [i = l])
        ifTrue:
         [result next← strm contents.
         strm reset].
      i ← i + 1].
   ↑result contents
numArgs | len n i char | "the number of arguments I take as a selector"
   (self o 1) isletter≡false ifTrue: [↑1].
   len← self length.
   n ← 0. "count colons, dots, and arrows"
   i ← 0.
   [(i← i+1) ≤ len] whileTrueDo:
      [(char ← self o i) = 58
        ifTrue: [n ← n + 1]
        ifFalse:
         [char = 3
           ifTrue: [n ← n + 1]
           ifFalse:
            [char = 95
              ifTrue: [n ← n + 1]]]].
   ↑n

Comparison
= x | |
   ↑self ≡ x
hash | |<primitive: 39>

stringhash | |
   ↑super hash
```

```
Compatibility
copy | |
    ↑self
recopy | |
    ↑self
species | |
    ↑String

Conversion
asString | |
    ↑super copy
printOn: strm | |
    strm append: self
!
SystemOrganization classify: #UniqueString under: 'Basic Data Structures'.!
UniqueString classInit!
```

```
                                                        "Vector"
Class new title: #Vector
   subclassof: Array
   fields: ''
   declare: '';
   bytesize: true;
   asFollows_]

Vector is a VariableLengthClass.  The length of a Vector may not be less than 0,
nor may it be greater than 8196.
A field of a Vector may contain any other object.
A new Vector contains nil in every field.
To create a new Vector of length 8, say:
   Vector new: 8

Microcode Primitives
o x   "subscript"
   [0 > x or: x > self length ⇒[user notify: 'Subscript out of bounds']
   x class ≡ Integer ⇒[Return the xth element of the Vector]
   ↑ super o x]
o x ← val  "assign value to element"
   [0 > x or: x > self length ⇒[user notify: 'Subscript out of bounds']
   x class ≡ Integer ⇒[Store the value val as the xth element. ↑val]
   ↑ super o x ← val]

Copying and Altering
, x | v |
   v ← self growby: 1 "use a stream if youre in a hurry".
   v last← x.
   ↑v

Conversion
asVector | |

printOn: strm | i |
   strm append: '('.
   (1 to: self length) do:
      [:i | strm print: self o i.
      strm space].
   strm append: ')'

As yet unclassified
base: b | strm i |
    "convert vector of numbers to a base"
   strm ← Stream default.
   (1 to: self length) do:
      [:i | strm append: (self o i base: b).
      strm space].
   ↑strm contents
_]
SystemOrganization classify: #Vector under: 'Basic Data Structures'._]
```

Sets and Dictionaries

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:27:19 pm.'_!
                                                        "Association"
Class new title: #Association
   subclassof: Object
   fields: 'key value'
   declare: '';
   asFollows_!

This class has not yet been commented

As yet unclassified
key | |
   ↑key
printOn: strm | |
   strm print: key.
   strm append: '->'.
   strm print: value
value | |
   ↑value
value← val | |
   ↑value ← val
_!
SystemOrganization classify: #Association under: 'Sets and Dictionaries'._!
```

"ClassOrganizer"

```
Class new title: #ClassOrganizer
   subclassof: Object
   fields: 'globalComment commentVector groupVector'
   declare: 'default ';
   asFollows_]
```

ClassOrganizers contain the formatting information for printing classes. Each String in commentVector describes a category comprising the messages contained in the Vector which is the corresponding entry in groupVector.

```
Initialization
classInit | |
   default ← 'As yet unclassified'
init: sortedVec | |
   self globalComment← 'This class has not yet been commented'.
   commentVector ← default inVector.
   groupVector ← sortedVec inVector

Access to parts
asStream | v t |
   v ← Stream new of: (Vector new: 200).
   groupVector do: [:t | v append: t].
   ↑v contents asStream
categories | |
   ↑commentVector
category: str | i |
   i ← commentVector find: str.
   i = 0
     ifTrue: [user notify: 'No such category: ' + str]
     ifFalse: [↑groupVector o i]
classify: selector under: heading | s h n |
   (selector is: Vector)
     ifTrue: [selector do: [:s | self classify: s under: heading]]
     ifFalse:
      [s ← commentVector find: heading.
      (s > 0 and: [(groupVector o s has: selector)])
        ifTrue: [↑self].
      (h ← self invert: selector)
        ifTrue:
         [heading = default
           ifTrue: [↑self].
         n ← commentVector find: h.
         groupVector o n ← groupVector o n delete: selector].
      s = 0
        ifTrue: [s ← self insert: heading].
      groupVector o s ← groupVector o s insertSorted: selector.
      n ← commentVector find: default.
      (n > 0 and: [(groupVector o n) length = 0])
        ifTrue: [self deleteCategory: default]]
delete: selector | i |
   "delete this from all categories"
   (1 to: groupVector length) do:
      [:i [ (groupVector o i has: selector)
        ifTrue:
         [groupVector o i ← groupVector o i delete: selector.
         ((groupVector o i) length = 0 and: [commentVector o i=default])
```

```
            ifTrue: [self deleteCategory: commentVector o i].
          ↑self]]
deleteCategory: cat | index |
   index ← commentVector find: cat.
   groupVector ← groupVector without: index.
   commentVector ← commentVector without: index
globalComment | |
   ↑globalComment asParagraph text
globalCommentItself | |
   ↑globalComment "used only by Class archiveOn:changesOnly:"
globalComment← t1 | |
    "String or RemoteParagraph"
   globalComment ← t1
has: sel | t |
   groupVector do:
      [:t | (t has: sel)
        ifTrue: [↑true]].
   ↑false
insert: heading | di dgroup hi |
    "force default category to end, delete if empty"
   (di ← commentVector find: default) > 0
     ifTrue: [dgroup ← groupVector o di].
   commentVector ← {(commentVector without: di) , heading}.
   groupVector ← {(groupVector without: di) , (Vector new: 0)}.
   hi ← commentVector length.
   (di = 0 or: [dgroup length = 0])
     ifTrue: [↑hi].
   commentVector ← {commentVector , default}.
   groupVector ← {groupVector , dgroup}.
   ↑hi
invert: selector | i |
   (1 to: groupVector length) do:
      [:i | (groupVector o i has: selector)
        ifTrue: [↑commentVector o i]].
   ↑false

Conversion to text
asParagraph | s i |
   s ← Stream default.
   s print: self globalComment.
   (1 to: commentVector length) do:
      [:i | s cr.
      s print: ((commentVector o i) inVector concat: groupVector o i)].
   ↑s contents asParagraph
fromParagraph: para | t i j g |
   user displayoffwhile: [
      (t ← para asVector.
      self globalComment← t o 1.
      commentVector ← Vector new: t length - 1.
      groupVector ← Vector new: t length - 1.
      (1 to: t length - 1) do:
         [:i | g ← t o (i + 1).
         commentVector o i ← g o 1.
         [0 = (j ← g find: #←)] whileFalseDo: [ "reconstitute ← suffixes"
            g ← g replace: j - 1 to: j by: (g o (j - 1) + '←') unique inVector].
         groupVector o i ← (g copy: 2 to: g length) sort])]
```

```
SystemOrganization classify: #ClassOrganizer under: 'Sets and Dictionaries'.
ClassOrganizer classInit
```

"HashSet"

```
Class new title: #HashSet
   subclassof: Object
   fields: 'objects'
   declare: '';
   asFollows_]
```

HashSets are pure sets of objects with no associated values. However, since they allow callers to determine the location of objects in the hash table, subclasses such as Dictionary and MessageDict can provide parallel tables to hold values. Such subclasses must intercept growto: and reorder their own tables then.

```
Initialization
copy | |
    "Π a copy of me"
   ↑self class new copyfrom: self
copyfrom: hset | |
    "take on state of hset"
   objects ← hset objects copy
init | |
   self init: 4
init: size | |
   objects ← Vector new: (size max: 1)

Access to parts
asStream | |
   ↑self contents asStream
contents | obj strm |
   strm ← (Vector new: objects length) asStream.
   objects do:
      [:obj | obj ≡ nil
        ifFalse: [strm next← obj]].
   ↑strm contents

Searching
find: obj | i |
    "Πindex if found, else false"
   i ← self findornil: obj.
   objects o i = obj
     ifTrue: [↑i].
   ↑false
findorerror: name | i |
   i ← self findornil: name.
   objects o i = name
     ifTrue: [↑i].
    "allow the user to put a correct value into i"
   user notify: name asString + ' cannot be found'.
   ↑i
has: obj | |
   ↑objects o (self findornil: obj) = obj

Insertion and deletion
delete: obj | i j l |
   (obj is: Vector)
     ifTrue: [obj do: [:i | self delete: i]]
     ifFalse:
      [i ← self findorerror: obj.
```

```
      objects o i ← nil.
      l ← objects length.
      [objects o (i ← (i = l
            ifTrue: [1]
            ifFalse: [i + 1])) ≡ nil] whileFalseDo:
         [i = (j ← self findornil: objects o i)
          ifFalse: [self swap: i with: j]]]
insert: obj | i |
   self findorinsert: obj.
   ↑obj

Private
findorinsert: obj | i |
    "insert if not found, "
   i ← self findornil: obj.
   objects o i = obj
     ifTrue: [↑i].
    "found it"
   self sparse
     ifTrue:
      [objects o i ← obj.
      ↑i].
    "insert if room"
   self growto: objects length * 2 "grow".
   ↑self findorinsert: obj "and insert"
findornil: obj | i loc |
    "↑index if found or available slot"
   loc ← obj hash \ objects length.
   (1 to: objects length) do:
      [:i | loc ← (loc = objects length
            ifTrue: [1]
            ifFalse: [loc + 1]).
      objects o loc ≡ nil
        ifTrue: [↑loc].
      objects o loc = obj
        ifTrue: [↑loc]].
   ↑1 "table full - caller must check for hit"
growto: newSize | copy item |
   copy ← self class new init: newSize.
   self do: [:item | copy rawinsert: item].
   objects ← copy objects
objects | |
   ↑objects
objects← t1 | |
   objects ← t1
rawinsert: t1 | t2 |
    "assumes there is room for the new one"
   t2 ← self findornil: t1.
   objects o t2 ← t1.
   ↑t2
sparse | i n |
   n ← objects length "↑true if (1 max: 1/4 of table) is nil".
   (1 to: objects length) do:
      [:i | objects o i ≡ nil
        ifTrue:
         [(n ← n - 4) ≤ 0
           ifTrue: [↑true]]].
```

```
    ↑false
swap: i with: j | |
    objects swap: i with: j

As yet unclassified
lookup: obj | i |
    "nobject if found, else false (object=obj but may not ≡obj)"
    i ← self findornil: obj.
    objects o i = obj
      ifTrue: [↑objects o i].
    ↑false
!
SystemOrganization classify: #HashSet under: 'Sets and Dictionaries'.!
```

"Dictionary"

```
Class new title: #Dictionary
   subclassof: HashSet
   fields: 'values'
   declare: '';
   asFollows_[
```

Dictionaries are sets with associated values. They are very handy but not terribly efficient. Most of their work is done by HashSet.

```
Initialization
copyfrom: dict | |
   self objects← dict objects copy.
   values ← dict values copy
init: size | |
   values ← Vector new: size.
   super init: size

Searching
o name | |
   ↑values o (self findorerror: name)
o name ← val | |
   ↑values o (self findorerror: name) ← val
lookup: name | x |
   (x ← self find: name)
     ifTrue: [↑values o x].
   ↑false

Inserting and Deleting
delete: name | |
   (name is: Vector)
     ifTrue: [super delete: name]
     ifFalse:
      [values o (self findorerror: name) ← nil.
      super delete: name]
insert: name with: value | |
   self insert: name.
   values o (self findorerror: name) ← value

Private
growto: size | name copy |
   copy ← self class new init: size "create a copy of the new size".
   self do: [:name | copy rawinsert: name. "hash each entry into it"
      copy insert: name with: self o name].
   self copyfrom: copy
swap: i with: j | |
   values swap: i with: j.
   super swap: i with: j
values | |
   ↑values

Inversion
invert: obj | i |
   (1 to: values length) do:
      [:i | values o i = obj
        ifTrue: [↑objects o i]].
   ↑false
```

```
SystemOrganization classify: #Dictionary under: 'Sets and Dictionaries'.
```

"LADSet"

```
Class new title: #LADSet
   subclassof: Object
   fields: 'tally'
   declare: '';
   bytesize: true;
   asFollows_]
```

This class has not yet been commented

```
As yet unclassified
asStream  | |
   ↑self contents asStream
at: index | |
   ↑self o index
at: index put: anAssociation | |
   ↑self o index ← anAssociation
elementAt: index | |
   ↑self o index
elementAt: index put: value | |
   ↑self o index ← value
fullCheck | |
   self length - tally ≤ (self length / 4)
     ifTrue: [self grow]
grow | |
   self growTo: self length * 2
growTo: size | copy i |
   copy ← self species init: size.
   1 to: self length do:
      [:i | self o i ≡ nil ifFalse: [copy noCheckAdd: self o i]].
   self become: copy
init | |
   tally ← 0
shrink | |
   [tally < (self length/2)] whileTrueDo: [self growTo: self length/2]
swap: one with: other  | save |
   save ← self o one.
   self o one ← self o other.
   self o other ← save
tally | |
   ↑tally
_]
SystemOrganization classify: #LADSet under: 'Sets and Dictionaries'._]
```

"LADDictionary"

```
Class new title: #LADDictionary
   subclassof: LADSet
   fields: ''
   declare: '';
   bytesize: true;
   asFollows_]
```

This class has not yet been commented

```
As yet unclassified
accept: assoc at: index | | "preserves original assoc if its there"
   (self elementAt: index)≡nil
      ifTrue: [tally← tally+1.
         self elementAt: index put: assoc.
         self fullCheck]
      ifFalse: [(self elementAt: index) value← assoc value]
add: anAssociation | |
   self accept: anAssociation at: (self findOrNil: anAssociation key).
   ↑anAssociation
at: key ifAbsent: expr | loc |
   loc← self find: key.
   loc=0 ifTrue: [↑expr eval].
   ↑self elementAt: loc
contents | s i |
   s ← Stream new of: (Vector new: tally).
   1 to: self length do:
      [:i | self o i ≡ nil ifFalse: [s next← self keyAt: i]].
   ↑s contents
delete: key  | loc old len entry |
   (loc ← self find: key)=0 ifTrue: [self error: 'key not found'].
   self elementAt: loc put: nil.
   tally← tally - 1.
   len ← self length.
   [loc ← loc = len ifTrue: [1] ifFalse: [loc + 1].
      (self elementAt: loc)≡ nil]
      whileFalseDo:
         [old ← self findOrNil: (self elementAt: loc) key.
         loc=old ifFalse: [self swap: loc with: old]]
find: key | index |
   index← self findOrNil: key.
   (self elementAt: index) ≡ nil ifTrue: [↑0].
   (self keyAt: index) = key
      ifTrue: [↑index]
      ifFalse: [↑0]
findOrNil: key | index len probe pass |
   len ← self length. pass ← 1.
   index ← key hash \ len + 1.
   [(probe ← self elementAt: index) ≡ nil or: [probe key = key]]
   whileFalseDo:
      [(index ← index + 1) > len ifTrue:
         [index ← 1.
         pass ← pass + 1.
         pass > 2 ifTrue: [↑self grow findOrNil: key]]].
   ↑index
has: key | |
   ↑(self find: key) ≠ 0
```

```
keyAt: index | |
   ↑(self elementAt: index) key
noCheckAdd: anAssociation | |
   self elementAt: (self findOrNil: anAssociation key) put: anAssociation.
   tally ← tally + 1.
   ↑anAssociation
!
SystemOrganization classify: #LADDictionary under: 'Sets and Dictionaries'.!
```

"IdentityDictionary"

```
Class new title: #IdentityDictionary
   subclassof: LADDictionary
   fields: 'values'
   declare: '';
   bytesize: true;
   asFollows_[
```

This class has not yet been commented

```
As yet unclassified
accept: assoc at: index | |  "preserves original assoc if its there"
   (self elementAt: index)≡nil
      ifTrue: [self elementAt: index put: assoc key.
         tally← tally+1.
         valuesoindex← assoc value.
         self fullCheck]
      ifFalse: [valuesoindex← assoc value]
at: key | |
   ↑values o (self find: key)
at: key put: value | index |
   index← self findOrNil: key.
   (self elementAt: index) ≡ nil ifTrue:
      [tally← tally+1.
      self elementAt: index put: key].
   ↑valuesoindex ← value
copy | v copy |
   v← values. values← values copy.
   copy← super copy. values← v.
   ↑copy

delete: key | |
   self become: (self copy deleteDangerously: key)
deleteDangerously: key  | loc old len entry |
   (loc ← self find: key)=0 ifTrue: [self error: 'key not found'].
   self elementAt: loc put: nil.  valuesoloc← nil.
   tally← tally - 1.
   len ← self length.
   [loc ← loc = len ifTrue: [1] ifFalse: [loc + 1].
      (self elementAt: loc)≡ nil]
      whileFalseDo:
         [old ← self findOrNil: (self elementAt: loc).
         loc=old ifFalse: [self swap: loc with: old]]
findOrNil: key | index len probe pass |
   len ← self length. pass ← 1.
   index ← key hash \ len + 1.
   [(probe ← self elementAt: index) ≡ nil or: [probe ≡ key]] whileFalseDo:
      [(index ← index + 1) > len ifTrue:
         [index ← 1.
         pass ← pass + 1.
         pass > 2 ifTrue: [↑self grow findOrNil: key]]].
   ↑index
growTo: size | copy i |
   copy ← self species init: size.
   1 to: self length do:
      [:i | self o i ≡ nil ifFalse: [copy noCheckAdd: self o i with: valuesoi]].
   self become: copy
```

```
init | |
   tally← 0.
   values← Vector new: self length
invert: value | index |
   index← values find: value.
   index=0 ifTrue: [↑false].
   ↑self elementAt: index
keyAt: index | |
   ↑self elementAt: index
method: key | |  "**change this to at: in all clients**"
   ↑self at: key
noCheckAdd: key with: value | index |
   index← self findOrNil: key.
   self elementAt: index put: key.
   tally ← tally + 1.
   ↑values◦index← value
swap: one with: other  | save |
   super swap: one with: other.
   values swap: one with: other
]
SystemOrganization classify: #IdentityDictionary under: 'Sets and Dictionaries'.]
```

Graphics

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:27:58 pm.'_!
                                                                    "BitBlt"
Class new title: #BitBlt
   subclassof: Object
   fields: 'effect screen dest destRaster destX destY
         width height source sourceRaster sourceX sourceY
         clipX clipY clipWidth clipHeight
         sourceForm destForm screenForm'
   declare: 'pageOneCursor ';
   asFollows_!
```

BitBlt copies bits from one rectangle to another in core. x, y, width and height are in bits, raster is in words, and base is a core address. Mode is storing, oring, xoring or erasing. If source or destination is a Smalltalk object, then you have to lock it, as in copyToString, below. The primitive does no bounds checking, so watch out.

```
Access to Parts
effect: anInteger | |
   effect ← #(35 39 38 36 44 45 41 33 51 55 54 52 19 23 22 20 ) o
      ((anInteger land: 15) + 1)
screen: t1 | |
   screenForm ← t1.
   screen ← screenForm bits
source: t1 | |
   source ← t1
window | |
   ↑clipX ⊙ clipY rect: clipX + clipWidth ⊙ (clipY + clipHeight)

Setup
classInit | |
   pageOneCursor ← 281 "location of hardware cursor"
clipRect: rect | |
   clipX ← rect minX.
   clipY ← rect minY.
   clipWidth ← rect width.
   clipHeight ← rect height
destForm: t1 | |
   destForm ← t1.
   self clipRect: (0 ⊙ 0 rect: destForm extent).
   destRaster ← destForm width + 15 / 16.
   dest ← destForm bits
destOrigin: destOrigin | |
   destX ← destOrigin x.
   destY ← destOrigin y
extent: extent | |
   width ← extent x.
   height ← extent y
fromDisplay | |
   self sourceForm: CurrentDisplay
init | |
   self effect: 0.
   self screen: black.
   destX ← destY ← width ← height ← sourceX ← sourceY ← 0
sourceForm: t1 | |
   sourceForm ← t1.
   sourceRaster ← sourceForm width + 15 / 16.
   source ← sourceForm bits
```

```
sourceRect: rect | |
   sourceX ← rect minX.
   sourceY ← rect minY.
   width ← rect width.
   height ← rect height
toDisplay | |
   self destForm: CurrentDisplay.
   source ≡ nil ifTrue: [self sourceForm: CurrentDisplay.]
window: rect | |  "this is public"
   self clipRect: (rect intersect: (0 ⊙ 0 rect: destForm extent))

Operations
callBLT | |<primitive: 66> "This copies the bits"
   user croak
copyRect: rect toPoint: point effect: eff screen: halftone | |
   self screen: halftone "careful -- assumes dest, destRaster, source and sourceRaster
are set!!".
   self effect: eff.
   self destOrigin: point.
   self sourceRect: rect.
   self callBLT
stringCopy: destString from: start to: stop with: replacement from: rstart to: rstop | |
   NoteTaker ifTrue: [↑false].
   "Copies equal subranges from one string to another.  Works for BitBlt parameters
up to 4096. maybe too much set up for short strings.  Currently, called by String
copy:to:with:from:to:"
   width ← 1 + stop - start.
   width = 0 ifTrue: [↑destString].
   ((start > 4096 or: [rstart > 4096]) or: [width ≥ 4096]) ifTrue: [↑false].
   ((width < 0 or: [width ≠ (1 + rstop - rstart)]) or: [((start < 1 or: [stop > destString
length]) or: [(rstart < 1 or: [rstop > replacement length])])]
     ifTrue: [↑false].
   destRaster ← destY ← sourceRaster ← sourceY ← 0.
   self effect: 0.
   self screen: black.
   height ← 1.
   width ← width * 8.
   dest ← destString.
   destX ← start - 1 * 8.
   source ← replacement.
   sourceX ← rstart - 1 * 8.
   self callBLT.
   destString o 1 ← destString o 1. "mark dirty"
   ↑destString
stringReplace: destString with: sourcestring from: start to: stop and: replacement
from: rstart to: rstop | |
   NoteTaker ifTrue: [↑false].
   "Works for BitBlt parameters less than 4096. Replaces a subrange of a string.
   Called only by String replace:to:by:from:to:.  Concatenates into destString:
      sourcestringo(1 to: start - 1)
      replacemento(rstart to: rstop)
      sourcestringo(stop + 1 to: sourcestring length).
   assumes String arguments"
   destString length = 0 ifTrue: [↑destString].
   (replacement is: String) ifFalse: [↑false].
   ((stop ≥ 4096 or: [sourcestring length - stop ≥ 4096]) or: [(start + rstop - rstart ≥
4096 or: [rstart > 4096])])
```

```
    ifTrue: [↑false].
  ((start < 1 or: [stop > sourcestring length]) or: [(rstart < 1 or: [rstop > replacement
length])])
    ifTrue: [↑false].
  destRaster ← destY ← sourceRaster ← sourceY ← 0.
  self effect: 0.
  self screen: black.
  height ← 1.
  dest ← destString.
  width ← start - 1 * 8.
  width = 0 ifFalse:
    [source ← sourcestring.
    sourceX ← destX ← 0.
    self callBLT].
  destX ← width.
  width ← 1 + rstop - rstart * 8.
  width = 0 ifFalse:
    [source ← replacement.
    sourceX ← rstart - 1 * 8.
    self callBLT].
  destX ← destX + width.
  width ← sourcestring length - stop * 8.
  width = 0 ifFalse:
    [source ← sourcestring.
    sourceX ← stop * 8.
    self callBLT].
  destString o 1 ← destString o 1. "mark dirty"
  ↑destString
!
SystemOrganization classify: #BitBlt under: 'Graphics'.!
BitBlt classInit!
```

"Bitmap"

```
Class new title: #Bitmap
   subclassof: String
   fields: ''
   declare: '';
   bytesize: 16;
   asFollows_]

This class has not yet been commented

As yet unclassified
fromString: s | i |
   (1 to: self length) do: [:i | self o i ← s o (i + i) + (s o (i + i - 1) lshift: 8)]
printOn: strm | |
   strm append: 'a Bitmap of length '; print: self length
_]
SystemOrganization classify: #Bitmap under: 'Graphics'._]
```

"DisplayBitmap"

```
Class new title: #DisplayBitmap
   subclassof: Bitmap
   fields: ''
   declare: '';
   bytesize: 16;
   asFollows_]

This class has not yet been commented
_]
SystemOrganization classify: #DisplayBitmap under: 'Graphics'._]
```

"Form"

```
Class new title: #Form
   subclassof: Object
   fields: 'extent bits offset figure ground'
   declare: 'white over color formmenu black under dotsetter blankcursor brush aurora
aurorarunning SPARE reverse ';
   asFollows_[
```

This class is a virtual bit map represented as a smalltalk String

```
INIT
classInit | |
   blankcursor ← nil. "Form classInit."
   black ← 0 - 1 "sets up colors and effects for BITBLT.".
   white ← 0.
   over ← 0.
   under ← 1.
   reverse ← 2.
   brush ← Form new extent: 5 ⊙ 5.
   brush black.
   color ← 1.
   formmenu ← Menu new string: 'brush
black
white
line
arc
erase
size
figure
ground
.
   dotsetter ← BitBlt new init " a BitBlt for pattern access.".
   dotsetter extent: 1 ⊙ 1.
   aurorarunning ← false.
   aurora ← nil "Aurora new"
close | |

extent: t1 | |
   extent ← t1.
   self extent: extent figure: 0 ground: 1 offset: 0 ⊙ 0 "creates a virtual bit map with
width = (extent x) and height = (extent y) with the bits all 1."
extent: t1 figure: t2 ground: t3 offset: t4 | |
   extent ← t1.
   figure ← t2.
   ground ← t3.
   offset ← t4.
   bits ← Bitmap new: extent y "creates a virtual bit map with width = (extent x) and
height = (extent y) with the bits all 1." * (extent x + 15 / 16)
fromuser | r |
   r ← Rectangle new "create a new Form whose rectangle is specified by the user. "
fromuser.
   self extent: r extent.
   bits ← r bitsIntoString
fromuserevenword | r |
   r ← Rectangle new "create a new Form whose rectangle is specified by the user,
      truncated to nearest multiple of 16 (for Spruce printing). " fromuserevenword.
   self extent: r extent.
```

```
    bits ← r bitsIntoString

  PATTERN ACCESS
  bits | |
    ↑bits "return the string containing the bits)"
 bits: t1 | |
    bits ← t1 "reset the string containing the bits)"
  black | i |
    (1 to: bits length "sets all bits in the form to black ( to ones)") do: [:i | bits o i ←
 8r177777]

 MODULE ACCESS
  extent | |
    ↑extent "return the extent (width⊙height) of the Form"
  figure: t1 | |
    figure ← t1 "set the figure ( color assiciated with black) for the form "
  ground: t1 | |
    ground ← t1 "set the ground ( color assiciated with white) for the form "
 height | |
    ↑extent y "return the height of the Form"
  length | |
    ↑bits length
 offset | |
    offset ≡ nil
      ifTrue: [↑0 ⊙ 0].
    ↑offset
  width | |
    ↑extent x "return the width of the Form"

 DISPLAY
 displayat: path effect: effect clippedBy: cliprect | r i clippedrect |
    (path is: Point) "basic form display primitive"
      ifTrue:
       [r ← Rectangle new origin: path extent: self extent.
       r bitsFromString: bits mode: effect clippedBy: cliprect.
       auroraruning
         ifTrue: [user displayoffwhile: [
             (clippedrect ← r intersect: user screenrect.
             aurora destination: clippedrect.
             aurora source: clippedrect.
             aurora figure: figure.
             aurora ground: ground.
             aurora function: 1103 "AoverB".
             aurora doit.
             aurora function: 0.
             aurora doit)]]]
      ifFalse:
       [(path is: Path)
         ifTrue: [(1 to: path length) do: [:i | self displayat: path o i effect: effect
 clippedBy: cliprect]]]

 SYSTEM
 copy | t |
    t ← Form new "return a copy of myself" extent: extent.
    t bits: bits copy.
    ↑t
 presson: press in: r | hs y |
```

```
    (hs ← press scale * self height) > r height
      ifTrue: [↑self].
     "not enough room left on current page.
       assume for now that it will at least fit on an entire page"
    press setp: r origin x ⊙ (y ← r corner y - hs).
    press bitmap: self bits: bits.
    ↑y

As yet unclassified
extent: t1 bits: t2 offset: t3 | |
    extent ← t1.
    bits ← t2.
    offset ← t3
!
SystemOrganization classify: #Form under: 'Graphics'.!
Form classInit!
```

"CursorForm"

```
Class new title: #CursorForm
    subclassof: Form
    fields: ''
    declare: '';
    asFollows_]
```

I am a 16 x 16 dot matrix suitable for use as the Alto hardware cursor

```
Initialization
fromString: t1 | |
    bits ← t1.
    self fromString: bits offset: 0 ⊙ 0
fromString: t1 offset: t2 | |
    extent ← 16⊙16.
    bits ← t1.
    offset ← t2
fromtext: str | |
    self fromtext: str offset: 0 ⊙ 0
fromtext: str offset: t2 | i s n c |
    extent ← 16⊙16.
    offset ← t2.
    bits ← Bitmap new: 16 "Not great, but compatible with printon.".
    s ← str asStream.
    s next.
    (1 to: 16) do:
        [:i | n ← 0.
        [(c ← s next) = 48 or: [c = 49]] whileTrueDo: [n ← n + n + (c - 48)].
        bits o i ← n]

Printing
hardcopy: pf | |
    self hardcopy: pf at: user mp "use current cursor position" - offset
hardcopy: pf at: loc | rect |
    rect ← loc extent: 16 ⊙ 16 "print cursor image at some point location into a
presssfile".
    pf setp: (pf transrect: rect) origin.
    pf bitmap: rect bits: bits
printOn: strm | i |
    strm append: 'CursorForm new fromtext: '''.
    (1 to: 16) do:
        [:i | strm cr.
        bits o i printOn: strm base: 2].
    strm append: ''' offset: '.
    strm print: offset.
    strm append: '.'

Showing
beCursor || <primitive: 96>
    user print: 'new cursor'
show | |
    user currentCursor: self
showwhile: expr | oldcursor value |
    oldcursor ← user currentCursor.
    self show.
    value ← expr eval.
    oldcursor show.
```

```
↑value
SystemOrganization classify: #CursorForm under: 'Graphics'.
```

"DisplayForm"

```
Class new title: #DisplayForm
   subclassof: Form
   fields: ''
   declare: '';
   asFollows_!
```

This class has not yet been commented

```
As yet unclassified
beDisplay || <primitive: 97>
   user print: 'new display'
_!
SystemOrganization classify: #DisplayForm under: 'Graphics'._!
```

"HalfTone"

```
Class new title: #HalfTone
   subclassof: Form
   fields: 'oldGray'
   declare: '';
   asFollows_!

Besides being a form for use as a halfTone gray, I cache the old integer gray value
for efficient backward compatibility
"!
SystemOrganization classify: #HalfTone under: 'Graphics'._!
```

"Point"

```
Class new title: #Point
    subclassof: Object
    fields: 'x y'
    declare: '';
    asFollows_!
```

I am an x-y pair of numbers usually designating a location on the screen

```
Initialization
copy | |
    ↑x ⊙ y
x: t1 y: t2 | |
    x ← t1.
    y ← t2

Arithmetic
≤ pt | |
    ↑x ≤ pt x and: [y ≤ pt y]
≥ pt | |
    ↑x ≥ pt x and: [y ≥ pt y]
* scale | |
    ↑Point new "Return a Point that is the product of me and scale (which is a Point or
Number)" x: x * scale asPtX y: y * scale asPtY
+ delta | |
    ↑Point new "Return a Point that is the sum of me and delta (which is a Point or
Number)" x: x + delta asPtX y: y + delta asPtY
- delta | |
    ↑Point new "Return a Point that is the difference of me and delta (which is a Point
or Number)" x: x - delta asPtX y: y - delta asPtY
/ scale | |
    ↑Point new "Return a Point that is the quotient of me and scale (which is a Point
or Number)" x: x / scale asPtX y: y / scale asPtY
< pt | |
    ↑x < pt x and: [y < pt y]
= pt | |
    ↑x = pt x and: [y = pt y]
> pt | |
    ↑x > pt x and: [y > pt y]
abs | |
    "absolute value of a point"
    ↑Point new x: x abs y: y abs
length | |
    ↑(x asFloat * x asFloat + (y asFloat * y asFloat)) sqrt
max: t | |
    ↑Point new x: (x max: t x) y: (y max: t y)
min: t | |
    ↑Point new x: (x min: t x) y: (y min: t y)
truncateTo: grid | |
    ↑Point new x: (x truncateTo: grid) y: (y truncateTo: grid)

Conversion
asPtX | |
    ↑x
asPtY | |
    ↑y
asRectCorner | |
```

```
    "pretend to be a Rectangle for Rectangle +-*/"
asRectOrigin | |
    "pretend to be a Rectangle for Rectangle +-*/"
corner | |
    ↑self + (1 ⊙ 1)
extent | |
    ↑1 ⊙ 1
extent: p | |
    "infix creation of rectangles"
    ↑Rectangle new origin: self extent: p
height | |
    ↑1
origin | |
    ↑self
printOn: strm | |
    strm print: x.
    strm append: '⊙'.
    strm print: y
rect: p | |
    "infix creation of rectangles"
    ↑Rectangle new origin: self corner: p
width | |
    ↑1

Access to parts
hash | |
    ↑(x lshift: 2) lxor: y
x | |
    ↑x
x← t1 | |
    x ← t1
y | |
    ↑y
y← t1 | |
    y ← t1
_]
SystemOrganization classify: #Point under: 'Graphics'._]
```

"Rectangle"

```
Class new title: #Rectangle
   subclassof: Object
   fields: 'origin corner'
   declare: '';
   asFollows_[
```

I am a pair of points, usually representing a rectangular area on the screen

```
Initialization
copy | |
   ↑origin copy "new rectangle" rect: corner copy
fromuser | t |
   origin ← OriginCursor showwhile: [ "Show the origin cursor until the user presses a
mouse button,
      then get my origin"
         (user waitbug)] "Show the corner cursor and complement me until the user
presses
      a button again.  The loop is arranged so
      that complementing stays on for a little while.".
   t ← origin.
   CornerCursor showwhile: [([corner ← t.
      t ← user mpnext] whileTrueDo:
         [self comp.
         t ← t max: origin.
         self comp])]
fromuserevenword | t |
   origin ← OriginCursor showwhile: [ "Show the origin cursor until the user presses a
mouse button,
      then get my origin"
         (user waitbug)] "Show the corner cursor and complement me until the user
presses
      a button again.  The loop is arranged so
      that complementing stays on for a little while.".
   t ← origin.
   CornerCursor showwhile: [([corner ← t.
      t ← user mpnext] whileTrueDo:
         [self comp.
         t ← (t x + 15 truncateTo: 16) @ t y max: origin.
         self comp])]
origin: t1 corner: t2 | |
   origin ← t1.
   corner ← t2
origin: t1 extent: extent | |
   origin ← t1.
   corner ← origin + extent

Aspects
corner | |
   ↑corner
corner← t1 | |
   corner ← t1
extent | |
   ↑corner - origin
hash | |
   ↑super hash
height | |
```

```
    ↑corner y - origin y
maxX | |
    ↑corner x
maxY | |
    ↑corner y
minX | |
    ↑origin x
minY | |
    ↑origin y
origin | |
    ↑origin
origin← t1 | |
    origin ← t1
width | |
    ↑corner x - origin x
width← w | |
    "change my right x to make my width w"
    corner x← origin x + w

Arithmetic
* scale | |
    ↑Rectangle new "Return a Rectangle which is the product of me and scale (which is
a Rectangle, Point, or Number)" origin: origin * scale asRectOrigin corner: corner *
scale asRectCorner
+ delta | |
    ↑Rectangle new "Return a Rectangle which is the sum of me and delta (which is a
Rectangle, Point, or Number)" origin: origin + delta asRectOrigin corner: corner + delta
asRectCorner
- delta | |
    ↑Rectangle new "Return a Rectangle which is the difference of me and delta (which
is a Rectangle, Point, or Number)" origin: origin - delta asRectOrigin corner: corner -
delta asRectCorner
/ scale | |
    ↑Rectangle new "Return a Rectangle which is the quotient of me and scale (which
is a Rectangle, Point, or Number)" origin: origin / scale asRectOrigin corner: corner /
scale asRectCorner
= r | |
    ↑origin = r origin and: [corner = r corner]
center | |
    ↑origin + corner / 2
empty | |
    ↑origin < corner ≡ false
has: pt | |
    ↑origin ≤ pt and: [pt < corner]
inset: p1 | |
    ↑origin + p1 rect: corner - p1
inset: p1 and: p2 | |
    ↑origin + p1 rect: corner - p2
intersect: r | |
    ↑Rectangle new origin: (origin max: r origin) corner: (corner min: r corner)
max: rect | |
    ↑Rectangle new origin: (origin min: rect origin) corner: (corner max: rect corner)
minus: r | s yorg ycor |
    "return Vector of Rectangles comprising
            the part of me not intersecting r "
    "Make sure the intersection is non-empty"
    (origin ≤ r corner and: [r origin ≤ corner])
```

```
        ifFalse: [↑self inVector].
    s ← (Vector new: 4) asStream.
    r origin y > origin y
        ifTrue: [s next← origin rect: corner x ⊙ (yorg ← r origin y)]
        ifFalse: [yorg ← origin y].
    r corner y < corner y
        ifTrue: [s next← origin x ⊙ (ycor ← r corner y) rect: corner]
        ifFalse: [ycor ← corner y].
    r origin x > origin x
        ifTrue: [s next← origin x ⊙ yorg rect: r origin x ⊙ ycor].
    r corner x < corner x
        ifTrue: [s next← r corner x ⊙ yorg rect: corner x ⊙ ycor].
    ↑s contents

Altering
dragTo: dest | v i |
    self blt: dest mode: storing.
    v ← dest rect: dest + self extent.
    (self minus: v) do: [:i | i clear].
    origin ← dest.
    corner ← v corner
growby: pt | |
    corner ← corner + pt
growto: t1 | |
    corner ← t1
moveby: pt | |
    origin ← origin + pt.
    corner ← corner + pt
moveto: pt | |
    corner ← corner + pt - origin.
    origin ← pt

Conversion
asRectCorner | |
    ↑corner
asRectOrigin | |
    ↑origin
bitmap | extent |
    extent ← corner - origin.
    ↑Bitmap new: extent y * (extent x + 15 / 16)
bitsFromString: str | |
    self bitsFromString: str mode: storing "default stores bits onto display"
bitsFromString: str mode: mode | t3 |
    (t3 ← BitBlt new toDisplay) effect: mode.
        t3 sourceForm: (Form new extent: corner - origin bits: str offset: nil).
        t3 destOrigin: origin.
        t3 sourceRect: (0 ⊙ 0 rect: corner - origin).
        t3 callBLT
bitsFromString: bitmap mode: mode clippedBy: clipRect | t5 |
    "Load the screen bits within my area from those stored in bitmap.  If clipRect is not
nil, then load only those bits within both  myself and clipRect"
    (t5 ← BitBlt new toDisplay window: clipRect) effect: mode.
    t5 sourceForm: (Form new extent: corner - origin bits: bitmap offset: nil).
    t5 destOrigin: origin.
    t5 sourceRect: (0 ⊙ 0 rect: corner - origin).
    t5 callBLT
bitsIntoString | str |
```

```
    str ← self bitmap.
    self bitsIntoString: str mode: storing.
    ↑str
bitsIntoString: str mode: mode | t3 |
    (t3 ← BitBlt new fromDisplay) effect: mode.
        t3 destForm: (Form new extent: corner - origin bits: str offset: nil).
        t3 destOrigin: 0 ⊙ 0.
        t3 sourceRect: (origin rect: corner).
        t3 callBLT
hardcopy: pf | |
    self hardcopy: pf thickness: 2
hardcopy: pf thickness: th | r |
    ((self inset: 0 - th) minus: self) do: [:r | pf showrect: r color: 0]
printOn: strm | |
    strm print: origin.
    strm append: ' rect: '.
    strm print: corner

Image
blt: dest mode: mode | |
    DisplayBLTer copyRect: self toPoint: dest effect: (mode land: 3) screen: black
clear | |
    "default is backround"
    self color: background mode: storing
clear: color | |
    self color: color mode: storing
color: color mode: mode | |
    DisplayBLTer copyRect: self toPoint: origin effect: 12 + (mode land: 3) screen: color
comp | |
    self color: black mode: xoring
flash | |
    self comp.
    self comp
reverse | |
    self color: black mode: xoring

Border
border: thick color: color | t3 |
    "paints a border withoud disturbing interior"
    (t3 ← Rectangle new origin: origin - (thick ⊙ thick) corner: corner x + thick ⊙
origin y) clear: color.
    t3 moveto: origin x - thick ⊙ corner y.
    t3 clear: color.
    t3 origin← corner x ⊙ (origin y - thick).
    t3 clear: color.
    t3 moveto: origin - (thick ⊙ thick).
    t3 clear: color
outline | |
    "default border is two thick"
    self outline: 2
outline: thick | t |
    t ← ¯1 ⊙ ¯1 * thick.
    (self inset: t) clear: black.
    self clear: white
↵
SystemOrganization classify: #Rectangle under: 'Graphics'.↵
```

Text Display

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:28:47 pm.'_!
                                                        "CharLine"
Class new title: #CharLine
    subclassof: Object
    fields: 'starti stopi spaces padwidth'
    declare: '';
    asFollows_!

A CharLine specifies the parameters of a measured line of text.
stopi is not really needed, as it can be inferred by the successors starti.

!Initialization
starti: t1 stopi: t2 spaces: t3 padwidth: t4 | |
    starti ← t1.
    stopi ← t2.
    spaces ← t3.
    padwidth ← t4

!Access to Parts
padwidth | |
    ↑padwidth
spaces | |
    ↑spaces
starti | |
    ↑starti
stopi | |
    ↑stopi

!Operations
= line | |
    ↑(starti = line starti and: [stopi = line stopi]) and: [(spaces = line spaces and:
[padwidth = line padwidth])]
slide: delta | |
    starti ← starti + delta.
    stopi ← stopi + delta
_!
SystemOrganization classify: #CharLine under: 'Text Objects'._!
```

"Dispframe"

```
Class new title: #Dispframe
   subclassof: Stream
   fields: 'text'
   declare: 'doit prompt ';
   asFollows_]

I am a dialog window

Initialization
classInit | |
   prompt ← '_' o 1.
   doit ← '_]' o 1
frame← r | |
   text para: nil frame: r
init | |
   text ← Textframe new.
   self of: (String new: 16)
rect: r | |
   self init.
   self frame← r.
   self clear

Scheduler
eachtime | t |
   (text window has: user mp)
     ifTrue:
      [user kbck
        ifTrue:
         [(t ← self kbd)
           ifTrue:
            [t ≡ nil
              ifFalse:
               [self space.
                self print: nil 's t].
             self prompt]]
        ifFalse:
         [user bluebug
           ifTrue: [↑false]]]
     ifFalse:
      [user anybug
        ifTrue: [↑false]]
firsttime | |
   (text window has: user mp)
     ifTrue:
      [self outline.
      self prompt]
     ifFalse: [↑false]
lasttime | |
   self last = prompt
     ifTrue:
      [self skip: ¯2.
      self show].
   ↑user bluebug ≡ false
leave | |
```

```
Dialog
ev | t |
   [self cr.
   t ← self request: ''] whileTrueDo:
      [self space.
      self print: nil 's t].
   ↑false
kbd | n t |
    "false if user pauses, nil if ctrl-d, all input since prompt if _"
   [user kbck] whileTrueDo:
      [t ← user kbd.
      t = 132
        ifTrue:
         [self append: 'done.'.
         self show.
         ↑nil].
       "ctl-d for done"
      t = 8
        ifTrue:
         [self last = prompt
           ifFalse: [self skip: ¯1]]
        ifFalse:
         [ "backspace"
         t = 30
           ifTrue:
            [n ← array o (position to: 1 by: ¯1) find: prompt.
            n = 0
              ifTrue:
               [self append: 'lost beginning'.
               self prompt]
              ifFalse:
               [t ← self last: n - 1.
               self next← doit.
               self show.
               ↑t]]
           ifFalse:
            [ "do-it (LF)"
            t = 145
              ifTrue:
               [self last = prompt
                 ifFalse:
                  [self skip: ¯1 "ctl-w for backspace word".
                  [position > 0 and: [self last tokenish]] whileTrueDo: [self skip: ¯1]]]
              ifFalse:
               [t = 151
                 ifTrue:
                  [self reset.
                  self prompt]
                 ifFalse: [ "ctl-x clears frame"
                  self next← t]]]]].
   self show.
   ↑false
prompt | |
   self cr.
   self next← prompt.
   self show
read | t |
```

```
    "false if ctrl-d, all input since prompt if _J"
  self next← prompt.
  self show.
  [user kbck
    ifTrue: [t ← self kbd]
    ifFalse: [[false]] whileFalseDo: [].
  t ≡ nil
    ifTrue: [↑false].
  ↑t
request: s | |
    "false if ctrl-d, all input since prompt if _J"
  self append: s.
  ↑self read

Image
clear | |
  self reset.
  self show
moveto: pt | |
  (text window inset: ¯2 ⊙ ¯2) dragto: pt - (¯2 ⊙ ¯2)
outline | |
  text window outline: 2
show | t |
  text show: self contents asParagraph.
  [text lastshown ≥ position] whileFalseDo:
    [position < (t ← text scrolln: 1)
      ifFalse:
        [t ← array copy: t + 1 to: position.
        text show: t asParagraph.
        position ← 0.
        self append: t "self dequeue: (text scrolln: 1).
      text show: self contents"]]

Access to Parts
frame | |
  ↑text frame
text | |
  ↑text
_J
SystemOrganization classify: #Dispframe under: 'Text Objects'._J
Dispframe classInit_J
```

```
                                                                    "Font"
Class new title: #Font
   subclassof: Object
   fields: 'name minascii maxascii maxwidth length ascent descent xoffset raster
glyphs xtable'
   declare: '';
   asFollows_!

"This class models the strike format for fonts"

Access to Parts
ascent | |
   ↑ascent
descent | |
   ↑descent
glyphs | |
   ↑glyphs
height | |
   ↑ascent + descent "height of font"
maxascii | |
   ↑maxascii
minascii | |
   ↑minascii
name | |
   ↑name
raster | |
   ↑raster
spacewidth | w |
   w ← self widthof: 32.
   w = 0
     ifTrue: [↑4].
   ↑w
widthof: glyph | |
   ↑xtable o (glyph + 2) - (xtable o (glyph + 1))
xtable | |
   ↑xtable
_!
SystemOrganization classify: #Font under: 'Text Objects'._!
```

"FontSet"

```
Class new title: #FontSet
   subclassof: Object
   fields: 'fonts names height ascent descent'
   declare: '';
   asFollows_!

Structure for holding a set of fonts

Initialization
init | |
   fonts ← Vector new: 16.
   names ← Vector new: 16.
   height ← ascent ← descent ← 0

Access
o n | |
   (n > (fonts length - 1) or: [n < 0])
     ifTrue: [user notify: 'fontset offset < 0 or > 15 illegal']
     ifFalse:
      [(fonts o (n + 1) ls: Font)
        ifTrue: [↑fonts o (n + 1)].
       (fonts o 1 ls: Font)
        ifTrue: [↑fonts o 1].
       user notify: 'No valid fonts in this FontSet']
o n ← font | |
   (n > (fonts length - 1) or: [n < 0])
     ifTrue: [user notify: 'fontset offset < 0 or > 15 illegal']
     ifFalse:
      [names o (n + 1) ← font name asUppercase.
       fonts o (n + 1) ← font.
       ascent < font ascent
        ifTrue: [ascent ← font ascent].
       descent < font descent
        ifTrue: [descent ← font descent].
       height ← ascent + descent]
ascent | |
   ↑ascent
baseline | |
   ↑ascent
descent | |
   ↑descent
height | |
   ↑height
size: n | s c size |
    "return size from fontname"
   names o n ≡ nil
     ifTrue: [n ← 1].
   size ← 0.
   s ← (names o n) asStream.
   [(c ← s next) isletter] whileTrueDo: [].
   [size ← size * 10 + (c - 48).
   c ← s next] whileTrueDo: [].
   ↑size
_!
SystemOrganization classify: #FontSet under: 'Text Objects'._!
```

"Paragraph"

```
Class new title: #Paragraph
   subclassof: Array
   fields: 'text runs alignment'
   declare: '';
   asFollows_[

Paragraphs implement pretty text.
   text is a String of the ascii characters.
   alignment specifies how the paragraph should be justified.
   runs is a String of run-coded format information.
      odd byte is run length (≤255)
      following byte is 16*format number +
         1*bold 2*italic 4*underline 8*strikeout
      longer runs are made from several of length 255.

Initialization of parts
copy | |
   ↑self class new text: text runs: runs alignment: alignment
text: t1 alignment: t2 | |
   text ← t1.
   alignment ← t2
text: t1 runs: t2 alignment: t3 | |
   text ← t1.
   runs ← t2.
   alignment ← t3

Normal access
o x | |
   ↑text o x
asParagraph | |
   ↑self
asStream | |
   ↑text asStream
asVector | |
   ↑text asVector
copy: a to: b | |
    "Return a copied subrange of this paragraph"
   ↑self class new text: (text copy: a to: b) runs: (self run: a to: b) alignment:
alignment
findString: str startingAt: start | |
   ↑text findString: str startingAt: start
length | |
   ↑text length
replace: a to: b by: c | |
    "alters self - doesnt copy"
   (runs ≡ nil and: [(c isnt: self class)])
     ifFalse: [runs ← self runcat: (self run: 1 to: a - 1) and: [((c is: self class)
              ifTrue: [c runs]
              ifFalse: [self makerun: c length val: (runs empty
                 ifTrue: [0]
                 ifFalse: [runs o ((self runfind: b) o 1 + 1)])])] and: [(self run: b + 1 to:
text length)]].
   text ← text replace: a to: b by: ((c is: self class)
          ifTrue: [c text]
          ifFalse: [c])
subst: x for: y | |
```

```
    "runs are not supported yet here"
    ↑text subst: x for: y
text | |
    ↑text
textStyle | |
    ↑DefaultTextStyle

Text alignment
alignment | |
    ↑alignment
alignment← t1 | |
    alignment ← t1
center | |
    alignment ← 2
justify | |
    alignment ← 1

Manipulation of format runs
allBold | |
    self maskrunsunder: 1 to: 1
makeBoldPattern | s i c |
    s ← text asStream.
    i ← 0.
    [(c ← s next)
      ifTrue:
       [ " scan to bracket, bar or comment "
       c = 91
         ifTrue: [true]
         ifFalse:
          [c = 124
            ifTrue: [true]
            ifFalse:
             [c = 34
               ifTrue: [true]
               ifFalse:
                [c = 25
                  ifTrue: [true]
                  ifFalse: [false]]]]]
      ifFalse: [true]] whileFalseDo:  [ "end"
       i ← i + 1].
    self maskrun: 1 to: i under: 1 to: 1
makerun: len val: val | str i |
    "Make up a solid run of value val"
    len = 0
      ifTrue: [↑nullString].
    str ← String new: len - 1 / 255 + 1 * 2.
    (1 to: str length by: 2) do:
       [:i | str o i ← (len > 255
         ifTrue: [255]
         ifFalse: [len]).
       str o (i + 1) ← val.
       len ← len - 255].
    ↑str
maskrun: i to: j under: m to: val | r k |
    "Alter my runs so that the bits selected by m become val." "Maybe merge this with
mergestyle"
    r ← self run: i to: j.
```

```
    (2 to: r length by: 2) do: [:k | r o k ← (r o k land: 255 - m) + val].
    runs ← self runcat: (self run: 1 to: i - 1) and: [r] and: [(self run: j + 1 to: text
length)]
maskrunsunder: m to: val | |
    self maskrun: 1 to: text length under: m to: val
run: a to: b | c |
    "subrange of run"
    a > b
      ifTrue: [↑nullString].
    runs ≡ nil
      ifTrue: [↑self makerun: 1 + b - a val: 0].
    a ← self runfind: a.
    b ← self runfind: b.
    c ← runs copy: a o 1 to: b o 1 + 1 "copy the sub-run".
    a o 1 = (b o 1)
      ifTrue: [c o 1 ← 1 + (b o 2) - (a o 2)]
      ifFalse:
       [c o 1 ← 1 + (runs o (a o 1)) - (a o 2) "trim the end lengths".
       c o (c length - 1) ← b o 2].
    ↑c
runAndVal: index | run t |
    "length and value at index"
    runs ≡ nil
      ifTrue: [↑{text length - index + 1 , 0}].
    run ← 1.
    [(t ← index - (runs o run)) > 0] whileTrueDo:
       [index ← t.
       run ← run + 2].
    ↑{runs o run - index + 1 , (runs o (run + 1))}
runcat: r1 and: r2 and: r3 | i r olen len oc c nr |
    nr ← Stream new "concatenate and compact 3 runs" of: (String new: 30).
    oc ← false.
    (1 to: 3) do:
       [:i | r ← (i = 1
              ifTrue: [r1]
              ifFalse:
               [i = 2
                 ifTrue: [r2]
                 ifFalse: [r3]]).
       r length = 0
         ifFalse:
          [r ← r asStream.
          [len ← r next] whileTrueDo:
             [c ← r next.
             len = 0
               ifFalse:
                [ "ignore empty runs (shouldn't be any)"
                oc = c
                  ifTrue:
                   [(olen ← olen + len) ≤ 255
                     ifFalse:
                      [nr next← 255.
                      nr next← oc.
                      olen ← olen - 255]]
                  ifFalse:
                   [oc
                     ifTrue:
```

```
                    [nr next← olen.
                    nr next← oc] "first time thru".
                olen ← len.
                oc ← c]]]]].
    oc
      ifTrue:
       [nr next← olen "leftovers".
       nr next← oc].
    ↑nr contents
runfind: index | run t |
    "index into run"
    run ← 1.
    [(t ← index - (runs o run)) > 0] whileTrueDo:
       [index ← t.
       run ← run + 2].
    ↑{run , index}
runs | |
    "return runs or default if none"
    runs ≡ nil
      ifTrue: [↑self makerun: text length val: 0].
    ↑runs

Bravo conversions
applyBravo: s at: i to: j | v ch t bslash cr |
    s ← s asStream "Alter runs of characters i through j according to trailer.
    see Ibis<Bravo>Trailer.Memo for further info.
    some functions may not be implemented, thus parsed and ignored.

    paragraph looks.
    implemented: justification (j), centering (c).
    ignored: left margin (l), first line left margin (d), right margin (z),
    line leading (x), paragraph leading (e), vertical tab (y), keep (k), profile (q),
    tab tables ( () )".
    cr ← 13.
    bslash ← '\' o 1.
    [(ch ← s next) = bslash] whileFalseDo:
       [(ch ≡ false or: [ch = cr])
         ifTrue: [↑self].
        "no more"
       (t ← 'jcq' find: ch) > 0
         ifTrue:
          [t = 1
            ifTrue: [self justify]
            ifFalse:
             [t = 2
               ifTrue: [self center]]]
         ifFalse:
          [(t ← '(ldzxeyk' find: ch) > 0
            ifTrue:
             [t = 1
               ifTrue: [s skipTo: ')' o 1]
               ifFalse: [s integerScan]]] "character looks.
    implemented: font (f), bold (bB), italic (iI), underline (uU).
    ignored: graphic (g), visible (v), overstrike (s), superscript (o), tabcolor (t)"].
    [(ch ← s next) and: [ch ≠ cr]] whileTrueDo:
       [ "run length"
       ((ch ≥ 48 and: [ch ≤ 57 "isdigit"])
```

```
        ifTrue: [s skip: ¯1]
        ifFalse: [ch = 32])
        ifTrue: [i ← i + s integerScan]
        ifFalse:
         [(t ← 'bBiIuU' find: ch) > 0
           ifTrue: [self maskrun: i to: j under: #(1 1 2 2 4 4 ) o t to: #(1 0 2 0 4 0 ) o t]
           ifFalse:
            [(t ← 'fot' find: ch) > 0
              ifTrue:
               [v ← s integerScan "new value follows".
               t = 1
                 ifTrue: [self maskrun: i to: j under: 240 to: (v lshift: 4)]]]]]
bravoRuns: s | i old len dif new bit bits |
   "Encode the runs in a Bravo paragraph trailer onto a Stream"
  s append: "assume Ctrl-Z is already there"
  (alignment = 1
    ifTrue: ['j\g']
    ifFalse:
     [alignment = 2
       ifTrue: ['c\g']
       ifFalse: ['\g']]).
  runs ≡ nil
    ifFalse:
     [len ← 0.
     old ← 256.
     bits ← #(1 2 4 ).
     (1 to: runs length by: 2) do:
        [:i | dif ← old lxor: (new ← runs o (i + 1)).
        (dif land: 247) = 0
          ifTrue: [ "No changes"
           len ← len + (runs o i)]
          ifFalse:
           [i = 1
             ifFalse: [len printOn: s].
           (1 to: 3) do:
              [:bit | (dif land: bits o bit) = 0
                ifFalse: [s next← ((new land: bits o bit) ≠ 0
                   ifTrue: ['biu']
                   ifFalse: ['BIU']) o bit]].
           (dif land: 240) ≠ 0
             ifTrue:
              [ "Font change"
              s append: 'f'.
              s print: (new lshift: ¯4).
              s space].
           old ← new.
           len ← runs o i]]].
  s cr
fromBravo | newpara newtext loc i j |
   "Find Bravo trailers and return a copy of self with them applied"
  newpara ← self copy.
  loc ← 1.
  [(i ← (newtext ← newpara text) find: 26) ≠ 0] whileTrueDo:
     [j ← newtext o (i + 1 to: newtext length) find: 13.
     newpara applyBravo: newtext o (i + 1 to: i + j) at: loc to: i - 1.
     newpara replace: i to: (i + j = newtext length
       ifTrue: [i + j]
```

```
        ifFalse: [i + j - 1]) by: ''.
      loc ← i + 1].
  ↑newpara
toBravo | s |
  s ← (String new: text length * 2) asStream.
  s append: text.
  s next← 26.
  self bravoRuns: s.
  ↑s contents asParagraph

Press printing
presson: press in: r | |
  ↑self presson: press in: r style: self textStyle
presson: press in: r style: style | char pos s3 y chop |
  (text length "Output paragraph inside rectangle (page coordinates)" "probably
ParagraphScanner should handle this" > 0 and: [text o 1 = 12])
    ifTrue: [↑self copy: 2 to: text length "formfeed --> page break"].
  y ← r corner y "We change corner y later".
  s3 ← ParagraphScanner new of: self to: press style: style.
  s3 init in: r.
  pos ← s3 position.
  chop ← (alignment = 1
        ifTrue: [0]
        ifFalse: [alignment]).
  [y and: [(char ← s3 scan)]] whileTrueDo:
    [char = 9
      ifTrue: [s3 tab]
      ifFalse:
       [(char = 32 or: [char = 13])
         ifTrue:
          [(y ← s3 printfrom: pos aligned:  "carriage return or exceeded max width
and backed up to blank"
                (char = 32
                  ifTrue: [alignment]
                  ifFalse: [chop]) skip: 1)
            ifTrue:
             [r corner y← y.
              s3 init in: r.
              pos ← s3 position]]
         ifFalse:
          [char ≡ true
            ifTrue:
             [s3 backup "exceeded max width with no blanks in line".
              (y ← s3 printfrom: pos aligned: 0 skip: 0)
                ifTrue:
                 [r corner y← y.
                  s3 init in: r.
                  pos ← s3 position]] "user notify: 'unimplemented control char'"]] "Put
out trailing text if any"].
  (y and: [(pos = s3 position or: [(y ← s3 printfrom: pos aligned: chop skip: 0)])])
    ifTrue:
     [press append: text.
      ↑y].
  press append: text o (1 to: pos).
  ↑self copy: pos + 1 to: text length

Filing
```

```
readFrom: file | |
    text ← file nextString.
    runs ← file nextString.
    alignment ← file next.
    runs empty
      ifTrue: [runs ← nil]

SystemOrganization classify: #Paragraph under: 'Text Objects'.
```

"StyleSheet"

```
Class new title: #StyleSheet
   subclassof: Object
   fields: 'fontset linelead lineheight baseline tabsize spacesize effect'
   declare: '';
   asFollows_!
```

This class has not yet been commented

```
Access to Parts
baseline | |
   ↑baseline
font: fontindex | |
   ↑fontset o fontindex
lineheight | |
   ↑lineheight
tabsize | |
   ↑tabsize
!
SystemOrganization classify: #StyleSheet under: 'Text Objects'._!
```

"Textframe"

```
Class new title: #Textframe
   subclassof: Object
   fields: 'frame window para style lines lastline'
   declare: 'tab space cr ';
   asFollows_]

I display a paragraph on the screen in a frame clipped by a window

Initialization
classInit | |
   space ← 32.
   cr ← 13.
   tab ← 9
frame: t1 window: t2 para: t3 style: t4 | |
   frame ← t1.
   window ← t2.
   para ← t3.
   style ← t4.
   frame ≡ nil ifFalse: [frame ← window ← frame intersect: user screenrect].
   lines ← Vector new: 32.
   lastline ← 0.
   (para ≡ nil or: [frame ≡ nil])
     ifFalse: [self measureall]
para: t1 frame: t2 | |
   para ← t1.
   frame ← t2.
   self frame: frame window: frame para: para style: DefaultTextStyle
para: t1 frame: t2 style: t3 | |
   para ← t1.
   frame ← t2.
   style ← t3.
   self frame: frame window: frame para: para style: style

Scheduling
aboutToFrame | |
    "My frame is about to change.  I dont care."
takeCursor | |
   user cursorloc← window center "Move the cursor to the center of my window."

Access to Parts
frame | |
   ↑frame
frame← t1 | |
   frame ← t1.
   window ← frame "Change my frame and window."
lineheight | |
   ↑style lineheight
para | |
   ↑para
window | |
   ↑window

Measuring
charofpoint: point andrects: rect | linei leftx rightx y chari P orig r |
   "Return character index of character at point" "TD enter."
   linei ← 0 max: (self lineofy: point y).
```

```
    y ← self yofline: linei.
    linei = 0
      ifTrue:
       [chari ← 1.
       orig ← frame minX ⊙ (y + style lineheight)]
      ifFalse:
       [point y > (y + style lineheight)
         ifTrue:
          [self pointofchar: para length + 1 andrect: [:r | r].
          rect value← r.
          ↑para length + 1].
       P ← self printer: false.
       P scanline: lines ∘ linei at: y stopx: point x stopchar: (lines ∘ linei) stopi.
       chari ← P chari.
       rightx ← P rightx.
       leftx ← P leftx.
       (point x ≤ (leftx + rightx / 2) or: [para ∘ chari = cr])
         ifTrue: [orig ← leftx ⊙ y]
         ifFalse:
          [chari ← chari + 1.
          orig ← rightx ⊙ y]].
    rect value← Rectangle new origin: orig extent: 0 ⊙ style lineheight.
    ↑chari
lastshown | ignored |
    ↑self charofpoint: window corner-1 andrect: [:ignored | ignored]
maxx: char | i rect maxx |
    self measureall.
    maxx ← 0.
    (1 to: lastline) do:
       [:i | self pointofchar: (lines ∘ i) stopi andrect: [:rect | rect].
       rect corner x > maxx
         ifTrue: [maxx ← rect corner x]].
    ↑maxx ⊙ rect corner y
measureall | starti stopi linei |
     "Set up lines for text displayer"
    starti ← linei ← 1.
    stopi ← para length.
    [starti > stopi] whileFalseDo:
       [self line: linei ← self measurefrom: starti to: stopi.
       starti ← (lines ∘ linei) stopi + 1.
       linei ← linei + 1].
    self trimLinesTo: linei - 1
measurefrom: starti to: stopi | line chari spacecount rightbits P stopx |
     "measure and return a line"
    line ← CharLine new starti: starti stopi: stopi spaces: 0 padwidth: 0.
    P ← self printer: false.
    P scanline: line at: 0 stopx: (stopx ← frame maxX) stopchar: stopi.
    chari ← P chari.
    spacecount ← P spacecount.
    (chari = stopi or: [para ∘ chari = cr])
      ifTrue:
       [ "CR or string end"
       rightbits ← stopx - P rightx.
       spacecount ← 0]
      ifFalse:
       [spacecount > 0
         ifTrue:
```

```
            [ "Back up to space"
            chari ← P spacei.
            rightbits ← stopx - P spacex.
            (chari > starti and: [para o (chari - 1) = space])
              ifTrue:
                [spacecount ← spacecount - 1.
                rightbits ← rightbits + P rightx - P leftx]
              ifFalse:
                [(chari < stopi and: [para o (chari + 1) = space])
                  ifTrue: [chari ← chari + 1]] "double spaces".
            spacecount ← 1 max: spacecount - 1]
          ifFalse:
            [chari ← chari - 1.
            rightbits ← stopx - P leftx]] "Back 1 char".
      ↑line starti: starti stopi: chari spaces: spacecount padwidth: rightbits
pointofchar: index andrect: rect | linei y P orig |
      "Return origin of character index"
      linei ← self lineofchar: index.
      y ← self yofline: linei.
      index > para length
        ifTrue:
          [(para length = 0 or: [para o para length = cr])
            ifTrue: [orig ← frame minX ⊙ (y + style lineheight)]
            ifFalse:
              [P ← self printer: false.
              P scanline: lines o linei at: y stopx: 9999 stopchar: index - 1.
              orig ← P rightx ⊙ y]]
        ifFalse:
          [.
          P ← self printer: false.
          P scanline: lines o linei at: y stopx: 9999 stopchar: index.
          orig ← P leftx ⊙ y].
      rect value← Rectangle new origin: orig extent: 0 ⊙ style lineheight.
      ↑orig
scrollN: n | ignored |
      ↑self charofpoint: frame corner x ⊙ (frame origin y + (n-1 * style lineheight))
andrect: [:ignored | ignored]

Displaying
clear | |
      (frame intersect: window) clear
clearfrom: y | |
      ((frame origin x ⊙ y rect: frame corner) intersect: window) clear: white
comp | |
      window comp
displayall | liney |
      "display all the lines in window"
      liney ← self displaylines: 1 to: lastline.
      self clearfrom: liney "self outline"
displaylines: startline to: stopline | liney line P lineheight boty leftx |
      "display a range, windowed"
      lineheight ← style lineheight.
      startline ← startline max: (self lineofy: window minY).
      stopline ← stopline min: (self lineofy: window maxY - lineheight).
      liney ← self yofline: startline.
      boty ← liney + lineheight.
      P ← self printer: true "P should do left and right clearing, since it has clip rect".
```

```
    (startline to: stopline) do:
      [:startline | line ← lines o startline.
      para alignment > 1
        ifTrue:
         [leftx ← frame minX + (para alignment = 4
                ifTrue: [line padwidth]
                ifFalse: [line padwidth / 2]).
          (window minX ⊙ liney rect: leftx ⊙ boty) clear: white].
      P scanline: line at: liney stopx: window maxX stopchar: line stopi.
      (para alignment nomask: 1)
        ifTrue: [(P rightx ⊙ liney rect: window maxX ⊙ boty) clear: white].
      liney ← boty.
      boty ← boty + lineheight].
    ↑liney
erase | |
    (window inset: ¯2 ⊙ ¯2) clear
outline | |
     "from TextFrame"
    window border: 2 color: black
printer: printing | |
    ↑TextScanner new frame: frame window: window para: para style: style printing:
printing
put: t1 at: pt | |
    para ← t1.
    self put: para at: pt centered: false
put: t1 at: pt centered: center | rect |
    para ← t1.
    window ← frame ← pt rect: 1000 ⊙ 1000.
    para ← para asParagraph.
    self measureall.
    self pointofchar: para length + 1 andrect: [:rect | rect].
    window growto: rect corner.
    center
      ifTrue: [window moveby: pt - window center].
    window ← window inset: ¯3 ⊙ ¯2.
    window clear: white.
    self show
put: t1 centered: pt | |
    para ← t1.
    self put: para at: pt centered: true
show | |
    self measureall.
    self displayall
show: t1 | |
    para ← t1.
    self measureall.
    self displayall

Editing
replace: starti to: stopi with: insert | i j k startline stopline oldlastline newline range
stopped oldlines begini oldy newy moveRect |
     "cut or paste"
    para replace: starti to: stopi by: insert "do the replacement in the string".
    lastline = 0
      ifTrue:
       [self measureall.
        self displayall]
```

```
    ifFalse:
     [ "if lines have never been set up,
                        measure them and display all the
                        lines falling in the window"
     oldlines ← lines copy.
     oldlastline ← lastline.
     i ← startline ← self lineofchar: starti "find the starting and stopping lines".
     stopline ← self lineofchar: stopi.
     range ← insert length - (stopi - starti + 1) "how many characters being
                              inserted or deleted" "If the starting line is not also the first
line, then measuring must commence from line preceding the one in which starti
appears. For example, deleting a line with only a carriage return may move
characters following the deleted portion of text into the line preceding the deleted
line.".
     begini ← (lines o i) starti.
     startline > 1
       ifTrue:
        [newline ← self measurefrom: (lines o (startline - 1)) starti to: para length.
        lines o (startline - 1) = newline
          ifFalse:
           [ "no backwrap" "start in line preceding that containing the starting
character"
           startline ← startline - 1.
           self line: startline ← newline.
           begini ← newline stopi + 1]].
     begini > para length
       ifTrue:
        [self trimLinesTo: (i - 1 max: 0) "nil lines after cut--remeasure last line
below".
        para length = 0
          ifTrue:
           [ "cut entire paragraph--clear and return"
           window clear: white.
           ↑self]].
     stopped ← false.
     j ← stopline.
     [stopped or: [begini > para length]] whileFalseDo:
        [self line: i ← newline ← self measurefrom: begini to: para length.
        [(j > oldlastline or: [newline stopi "no more old line to compare with" < (stopi
← (oldlines o j) stopi + range)]) or: [stopped]] whileFalseDo:
          [stopi = newline stopi
            ifTrue:
             [ "got the match"
             oldy ← self yofline: j + "get source and destination y's for moving the
unchanged lines"
                    (range < 0
                      ifTrue: [0]
                      ifFalse: [1]).
             newy ← self yofline: i + (range < 0
                      ifTrue: [0]
                      ifFalse: [1]).
             stopline ← i.
             stopped ← true "fill in the new line vector with the old unchanged lines.
          Update their starting and stopping indices on the way.".
             ((j ← j + 1) to: oldlastline) do: [:k | self line: (i ← i + 1) ← oldlines o k
slide: range].
             self trimLinesTo: i "trim off obsolete lines, if any"]
```

```
            ifFalse: [j ← j + 1]].
        begini ← newline stopi + 1.
        i ← i + 1 "Now the lines are up to date.  What remains is the move the
'unchanged' lines and display those which have changed."].
      begini > para length
        ifTrue: [ "If at the end of previous lines simply display lines
                  from the line in which the first character of the
                  replacement occured through the end of the paragraph."
        self displaylines: startline to: (stopline ← i min: lastline)]
        ifFalse:
        [ "Otherwise prepare to move the unchanged lines.  moveRect defines the
portion of the window containing the lines which may be move en masse." "Deletion
-- moving 'up' the screen or ..."
        moveRect ← window minX ⊙ (oldy max: window minY) rect: window corner
"Insertion -- moving 'down' the screen.
              Shorten moveRect by height of insertion or ...".
        oldy ≤ newy
          ifTrue: [moveRect corner← window corner + (0 ⊙ (oldy - newy))]
          ifFalse:
           [ "Deletion, and top of moveRect will fall above top of window.
      Increase the origin of moveRect by the amount that would fall above
      (hence outside) the window."
            newy < window minY
              ifTrue: [moveRect origin← window minX ⊙ (oldy + window minY -
newy)]] "Move it.".
        moveRect blt: window minX ⊙ (newy max: window minY) mode: storing
"Display the new lines.".
        self displaylines: startline to: stopline "A deletion may have 'pulled' previously
undisplayed lines into the window.
      If so, display them.".
        (newy < oldy and: [(self yofline: oldlastline + 1) > window maxY])
          ifTrue: [self displaylines: (self lineofy: window maxY - (oldy - newy)) to:
(stopline ← self lineofy: window maxY)]] "If we have done a deletion, obsolete material
may remain at the bottom of the window.  If so, clear it out.".
      oldlastline ≥ lastline
        ifTrue: [self clearfrom: (self yofline: lastline + 1)]]

Line management
line: i ← line | |
    "store a line, track last, and grow lines if necessary"
  i > lastline
    ifTrue: [lastline ← i].
  lastline > lines length
    ifTrue: [lines ← lines grow].
  ↑lines o i ← line
lineofchar: index | i |
    "Return line number (may be 0)"
  (1 to: lastline) do:
    [:i | index ≤ (lines o i) stopi
      ifTrue: [↑i]].
  ↑lastline
lineofy: y | |
    "Return line number for a given y (may be out of range!)"
  ↑y - frame minY / style lineheight + 1 min: lastline
trimLinesTo: last | i |
  (last + 1 to: lastline) do: [:i | lines o i ← nil].
  (lastline ← last) < (lines length / 2)
```

```
    ifTrue: [lines ← lines growby: 0 - (lines length / 2)]
yofline: line | |
    "top y of given line"
    ↑frame minY + (line - 1 * style lineheight)

Conversion
makeParagraph | |
    para ≡ nil "simulate ListPane for hardcopy"
      ifTrue: [para ← 'NIL !' asParagraph]

Printing
hardcopy | pf |
    user displayoffwhile: [
      (pf ← dpo pressfile: 'frame.press'.
      window hardcopy: pf.
      self hardcopy: pf.
      pf close.
      pf toPrinter)]
hardcopy: pf | first last lasty len parag left right top bottom rect |
    para ≡ nil
      ifTrue: [self makeParagraph].
    parag ← para asParagraph.
    frame = window
      ifTrue: [parag presson: pf in: (pf transrect: window) style: style]
      ifFalse:
       [left ← frame minX max: window minX.
       right ← window maxX min: frame maxX.
       bottom ← window maxY min: frame maxY.
       top ← window minY max: frame minY.
       lasty ← top + 4 "slop for char finding and making print rect larger".
       first ← self charofpoint: left + 1 ⊙ lasty andrect: [:rect | rect].
       len ← parag length.
       (frame minX ≥ left and: [frame maxX ≤ right])
         ifTrue: [(parag copy: first to: len) "paragraph is inset and may be scrolled"
presson: pf in: (pf transrect: (left ⊙ top rect: right ⊙ (bottom + 4))) style: style]
         ifFalse: [ "yuk, frame extends left or right so do it a line at a time for clipping"
          [first < len and: [lasty < bottom]] whileTrueDo:
             [last ← (self charofpoint: right - 1 ⊙ lasty andrect: [:rect | rect]) min: len.
             lasty ← lasty + rect height.
             (parag copy: first to: last) presson: pf in: (pf transrect: (left ⊙ rect minY
rect: right ⊙ lasty)) style: style.
             first ← last + 1]]]

Image
style | |
    ↑style
]
SystemOrganization classify: #Textframe under: 'Text Objects'.]
Textframe classInit]
```

"TextImage"

```
Class new title: #TextImage
   subclassof: Textframe
   fields: 'c1 c2 begintypein superimage oldEntity sel'
   declare: 'cut scrap paragraphmenu ctlw paste Deletion on esc off Scrap bs ';
   asFollows_]
```

This class has not yet been commented

```
INIT
classInit | |
   bs ← 8.
   ctlw ← 145.
   esc ← 160.
   cut ← 173.
   paste ← 158.
   Scrap ← Deletion ← nullString.
   on ← 1.
   off ← 0.
   paragraphmenu ← Menu new string: 'again
copy
cut
paste
doit
compile
undo
cancel
align
fit
resize
'
close | |
   superimage ← nil
paragraph: t1 frame: t2 style: t3 | |
   para ← t1.
   frame ← t2.
   style ← t3.
   nil ≡ para
     ifTrue: [para ← nullString].
   c1 ← c2 ← begintypein ← 1.
   self para: para frame: frame style: style

EDITING
align | |
   para alignment← #(1 2 4 0 0 ) o (1 + para alignment).
   self displayall.
   self reversefrom: c1 to: c2
checklooks | t val mask range |
   t ← #(166 150 137 151 230 214 201 215 135 159 144 143 128 127 129 131 180 149 199 223
208 207 192 191 240 226 ) find: user kbck.
   t = 0
     ifTrue: [↑false].
   user kbd.
   oldEntity
     ifFalse:
      [oldEntity ← para.
      para ← para copy].
```

```
    t = 25
      ifTrue: [para ← para toBravo]
      ifFalse:
       [ "ctl-T"
       t = 26
         ifTrue: [para ← para fromBravo]
         ifFalse:
          [ "ctl-F"
          .
          val ← #(1 2 4 256 ¯1 ¯2 ¯4 256 0 16 32 48 64 80 96 112 128 144 160 176 192 208
224 240 ) o t "ctl-b i - x  B I ¯ X" "ctl-0 1 ... 9" "ctl-shift-0 1 ... 5".
          val = 256
            ifTrue:
             [mask ← 255.
             val ← 0]
            ifFalse:
             [ "reset all"
             val < 0
               ifTrue:
                [mask ← 0 - val.
                val ← 0]
               ifFalse:
                [ "reset emphasis"
                (val > 0 and: [val < 16])
                  ifTrue: [mask ← val]
                  ifFalse: [ "set emphasis"
                    mask ← 240]]] "set font".
          para maskrun: c1 to: c2 - 1 under: mask to: val "fix paragraph".
          mask = 240
            ifTrue: [self replace: c1 to: c2 - 1 "remeasure if changing fonts" with: (para
copy: c1 to: c2 - 1)]
            ifFalse: [ "otherwise just redisplay lines"
              self displaylines: (self lineofchar: c1) to: (self lineofchar: c2)].
          self select]]
copyselection | |
    Scrap ← self selection "copy the current selection and store it in the Scrap."
cut | |
    self fintype "cut out the current selection and redisplay the paragraph.".
    self replace: nullString.
    self selectAndScroll.
    Scrap ← Deletion
fintype | |
    begintypein
      ifTrue:
       [begintypein < c1
         ifTrue:
          [Scrap ← para copy: begintypein to: c1 - 1.
          c1 ← begintypein].
       begintypein ← false]
      ifFalse: [↑false]
fixframe: f | dy |
    dy ← (frame ≡ nil
           ifTrue: [0]
           ifFalse: [self frameoffset]).
    window ← f copy.
    frame ← Rectangle new origin: window origin + (2 ⊙ dy) extent: window width - 4
⊙ 9999.
```

```
   self measureall.
   ↑window
paste | |
   self fintype "paste the Scrap over the current selection and redisplay the
paragraph.".
   self unselect.
   self replace: Scrap.
   self selectAndScroll
realign | |
   self align.
   sel ← on
replace: t | |
   oldEntity
     ifFalse:
      [oldEntity ← para.
      para ← para copy].
   begintypein
     ifFalse: [Deletion ← self selection].
   self replace: c1 to: c2 - 1 with: t.
   c2 ← c1 + t length
undo | |
   self fintype.
   self replace: Deletion.
   self complement

SELECTION
complement | |
   self complement: on
complement: nsel | |
   nsel = sel
     ifFalse:
      [ "already that way"
      (nsel = on and: [(user rawkbck or: [user redbug])])
        ifFalse:
         [ "slippage"
         sel ← nsel.
         self reversefrom: c1 to: c2]]
complementfrom: hair1 to: hair2 | temprect |
   hair1 minY "Complement the screen dots corresponding to the lines and part-lines
of the paragraph between hair1 inclusive and hair2 exclusive. If hair1 = hair2, this is
a no-op. If hair1 > hair2, they are reversed. This complementing happens in three
parts, A, B, and C, between points 1 and 2, according to the following illustration:
                1AAA
                BBBB
                BBBB
                BBBB
                CCC2
unless there is just one line involved, as in:
                1DD2
" "one line case" = hair2 minY
     ifTrue: [(((hair1 minX ≤ hair2 minX
        ifTrue: [hair1 origin rect: hair2 corner]
        ifFalse: [hair2 origin rect: hair1 corner]) intersect: frame) intersect: window)
comp]
     ifFalse:
      [hair1 minY > hair2 minY
        ifTrue:
```

```
        [temprect ← hair1.
        hair1 ← hair2.
        hair2 ← temprect].
      temprect ← frame minX ⊙ hair1 maxY rect: frame maxX ⊙ hair2 minY.
      (((hair1 origin rect: temprect maxX ⊙ temprect minY) intersect: frame)
intersect: window) comp.
      ((temprect intersect: frame) intersect: window) comp.
      (((temprect minX ⊙ temprect maxY rect: hair2 corner) intersect: frame)
intersect: window) comp]
reversefrom: char1 to: char2 | h1 h2 rect |
  self pointofchar: char1 andrects [:rect | rect] "Complement the dots corresponding to
the the lines and part-lines of the paragraph between the left edge of char1 and the
left edge of char2. If char1 = char2, this is sort of a no-op. If char1 > char2, this is
undefined.".
  h1 ← rect.
  char2 = char1
    ifTrue: [h2 ← h1 + (1 ⊙ 0)]
    ifFalse:
     [self pointofchar: char2 andrects [:rect | rect].
     h2 ← rect].
  self complementfrom: h1 to: h2
select | |
  sel ← off.
  c1 ≡ nil
    ifTrue: [c1 ← c2 ← 1].
  self complement: on
select: t | |
  self complement: off.
  c1 ← c2 ← t.
  self selectAndScroll
selectAndScroll | l dy c1y ignored |
  l ← self lineheight.
  self select.
  c1y ← (self pointofchar: c1 andrects [:ignored | ignored]) y.
  dy ← c1y - window minY.
  dy ≥ 0
    ifTrue: [dy ← c1y + l - 1 - window maxY max: 0].
  dy ≠ 0
    ifTrue: [self scrollby: dy abs + l - 1 / l * dy sign]
selecting | pt t h1 h2 c h drag2 selection |
  t ← self charofpoint: (pt ← user mp) andrects [:h1 | h1].
  self complement: off.
  self fintype.
  (t = c1 and: [c1 = c2])
    ifTrue:
     [[ "bugged hairline - maybe double-bug"
     user redbug and: [t = (self charofpoint: user mp andrects [:h | h])]] whileTrueDo:
[ "wait for unclick or drawing selection"
        ].
      user redbug ≡ false
        ifTrue:
         [self selectword.
         self select.
         ↑true]].
  sel ← on "draw out and record (c1 and c2) a selection".
  c1 ← c2 ← t.
  h2 ← h1 + (1 ⊙ 0).
```

```
    self complementfrom: h1 to: h2.
    selection ← true.
    [pt ← user mpnext] whileTrueDo:
      [c ← self charofpoint: pt andrect: [:h | h].
      c1 = c2
        ifTrue: [drag2 ← c ≥ c2].
      drag2
        ifTrue:
         [c < c1
           ifTrue: [self pointofchar: (c ← c1) andrect: [:h | h]].
          self complementfrom: h to: h2.
          c2 ← c.
          h2 ← h]
        ifFalse:
         [c > c2
           ifTrue: [self pointofchar: (c ← c2) andrect: [:h | h]].
          self complementfrom: h1 to: h.
          c1 ← c.
          h1 ← h].
      h1 = h2
        ifTrue: [self complementfrom: h1 to: (h2 ← h1 + (1 ⊙ 0))]].
    drag2
      ifFalse: [ "get rid of extra line in backwards select"
       self complementfrom: h2 - (1 ⊙ 0) to: h2]
selection | |
    para text empty
      ifTrue: [↑para copy].
    ↑para copy: c1 to: c2 - 1
selectionAsStream | |
    ↑Stream new of: para text from: c1 to: c2 - 1
selectRange | |
    ↑c1 to: c2 - 1
selectRange: r | |
    self complement: off.
    c1 ← r start.
    c2 ← r stop.
    self complement: on
selectword | a b dir t level open close s slen t10 |
    a ← b ← dir ← ¯1 "Select bracketed or word range, as a result of double-bug.".
    s ← para text.
    slen ← s length.
    level ← 1.
    open ← '([{<''"
'.
    close ← ')]}>''"
'.
    c1 ≤ 1
      ifTrue:
       [dir ← 1.
       t ← c1]
      ifFalse:
       [c1 > slen
         ifTrue: [t ← c1 - 1]
         ifFalse:
          [t ← open find: (a ← para o (c1 - 1)).
          t > 0
            ifTrue:
```

```
            [ "delim on left"
            dir ← 1.
            b ← close o t.
            t ← c1 - 1]
           ifFalse:
            [ "match to the right"
            t ← close find: (a ← para o c1).
            t > 0
              ifTrue:
               [ "delim on right"
               dir ← ¯1.
               b ← open o t.
               t ← c1]
              ifFalse:
               [ "match to the left"
               a ← ¯1.
               t ← c1]]]] "no delims - select a token".
    [level = 0 or: [(dir = 1
      ifTrue: [t ≥ slen]
      ifFalse: [t ≤ 1])]] whileFalseDo:
       [(t10 ← s o (t ← t + dir)) = b
         ifTrue: [level ← level - 1]
         ifFalse:
          [ "leaving nest"
          t10 = a
            ifTrue: [level ← level + 1]
            ifFalse:
             [ "entering nest"
             a = ¯1
               ifTrue:
                [(s o t) tokenish
                  ifTrue:
                   [ "token check goes left "
                   t = 1
                     ifTrue:
                      [c1 ← dir ← 1.
                      t ← c2]]
                  ifFalse:
                   [dir = ¯1
                     ifTrue:
                      [c1 ← t + 1.
                      dir ← 1.
                      t ← c2 - 1]
                     ifFalse: [ "then right"
                      level ← 0]]]]]].
    level ≠ 0
      ifTrue: [t ← t + dir].
    dir = 1
      ifTrue: [c2 ← t min: slen + 1]
      ifFalse: [c1 ← t + 1]
unselect | |
    self complement: off

SYSTEM
copy | t |
    t ← TextImage new paragraph: para copy frame: frame copy style: style copy.
```

```
    t c1← c1.
    t c2← c2.
    t begintypein← begintypein.
    ↑t
presson: press in: r | |
    ↑para presson: press in: r

ACCESS TO PARTS
begintypein← t1 | |
    begintypein ← t1
c1← t1 | |
    c1 ← t1
c2← t1 | |
    c2 ← t1
contents | |
    ↑para
Deletion← s | |
    Deletion ← s
formerly | |
    ↑oldEntity
formerly: t1 | |
    oldEntity ← t1
frameoffset | |
    ↑frame minY "a useful number" - window minY
frame← f | |
    self fixframe: f
height | rect |
    self pointofchar: para length + 1 andrect: [:rect | rect].
    ↑rect corner y - frame minY
Scrap← s | |
    Scrap ← s

DISPLAY
enter | |
    begintypein ← false.
    self show.
    self select
leave | |
    self complement: off
show | |
    lastline = 0
      ifTrue: [super show]
      ifFalse: [self displayall].
    sel ← off

PARAGRAPH EDITOR
scrollPos | t |
    t ← self height - self lineheight.
    t = 0
      ifTrue: [↑0.0].
    ↑0.0 - self frameoffset / t
scrollTo: f | |
    self scrollUp: self frameoffset + (f * self height) asInteger - 4
scrollUp: n | |
    self scrollby: n / self lineheight

SCROLLING
```

```
scrollby: n | oldw topline |
   (n > 0 and: [n + (topline ← self lineofy: window minY) > lastline])
     ifTrue: [n ← lastline - topline].
   n ← n * self lineheight max: self frameoffset.
   frame moveby: 0 ⊙ (0 - n).
   n abs ≥ window height
     ifTrue:
      [self show.
      self select]
     ifFalse:
      [ "need only to reshow part of window"
      oldw ← window.
      window ← (n < 0
            ifTrue: [window inset: 0 ⊙ 0 and: [0 ⊙ (0 - n)]]
            ifFalse: [window inset: 0 ⊙ n and: [0 ⊙ 0]]).
      window blt: window origin - (0 ⊙ n) mode: storing.
      n < 0
        ifTrue: [window corner y← window origin y - n]
        ifFalse: [window origin y← self yofline: (self lineofy: window corner y - n)].
      self displayall.
      self select.
      window ← oldw]

Editing
again | many |
   many ← user leftShiftKey.
   self fintype
     ifTrue:
      [Scrap ← Scrap text.
      self select].
   many
     ifTrue: [[self againOnce] whileTrueDo: []]
     ifFalse:
      [self againOnce
        ifFalse: [frame flash]]
againOnce | t |
   t ← para findString: Deletion startingAt: c2.
   t = 0
     ifTrue: [↑false].
   self unselect.
   c1 ← t.
   c2 ← c1 + Deletion length.
   self replace: Scrap.
   self selectAndScroll

Public Messages
kbd | more char |
    "key struck on the keyboard"
   (c1 < c2 and: [self checklooks])
     ifTrue: [↑self show complement].
   more ← (String new: 128) asStream.
   begintypein
     ifFalse:
      [Deletion ← self selection.
      begintypein ← c1].
   [char ← user kbdnext] whileTrueDo:
      [char = bs
```

```
        ifTrue:
         [more empty "backspace"
           ifTrue: [begintypein ← begintypein min: (c1 ← 1 max: c1 - 1)]
           ifFalse: [more skip: ¯1]]
        ifFalse:
         [char = cut
           ifTrue:
            [self fintype.
            c1 = c2
              ifTrue: [c2 ← c1 + 1 min: para length + 1].
            self replace: nullString.
            self complement.
            Scrap ← Deletion.
            ↑self].
          char = paste
           ifTrue: [↑self paste].
          char = ctlw
           ifTrue:
            [ "ctl-w for backspace word"
            more empty
              ifFalse:
               [self replace: more.
               more reset.
               c1 ← c2].
            c1 ← 1 max: c1 - 1.
            [c1 > 1 and: [(para o (c1 - 1)) tokenish]] whileTrueDo: [c1 ← c1 - 1].
            begintypein ← begintypein min: c1]
           ifFalse:
            [char = esc
              ifTrue:
               [ "select previous type-in"
               more empty
                 ifTrue: [self unselect]
                 ifFalse:
                  [self replace: more contents.
                  c1 ← c2].
               self fintype.
               c1 ← c2 - Scrap length.
               ↑self complement].
             "just a normal character"
            (char between: 11 and: 12)
              ifTrue: [more append: (char=11 ifTrue: ['ifTrue: '] ifFalse: ['ifFalse: '])]
              ifFalse: [more next← char]]]].
    self replace: more contents.
    c1 ← c2.
    self selectAndScroll
]
SystemOrganization classify: #TextImage under: 'Text Objects'.]
TextImage classInit]
```

"TextScanner"

```
Class new title: #TextScanner
    subclassof: BitBlt
    fields: 'printing chari stopx xtable exceptions spacecount spacei spacex charpad text
spacesize style para font fontno minascii maxascii glyphs frame looktype'
    declare: 'bs defaultExceptions tab space cr ';
    asFollows_]

A TextScanner is created to scan a paragraph, or just one line, and is then released.
It carries most of the state of the line being scanned, to answer questions at the end.
As a subclass of BitBlt, it can paste characters and spaces if printing is desired.
TextScanners will use less gas if driven over the whole paragraph, rather than being
created and destroyed for every line.

Initialization
classInit | |
    "TextScanner classInit."
    space ← 32.
    tab ← 9.
    cr ← 13.
    (defaultExceptions ← Vector new: 256) all← 0.
    defaultExceptions o (space + 1) ← 12.
    defaultExceptions o (tab + 1) ← 13.
    defaultExceptions o (cr + 1) ← 4
frame: t1 window: w para: t3 style: t4 printing: t5 | |
    frame ← t1.
    para ← t3.
    style ← t4.
    printing ← t5.
    super init.
    self screen: white.
    self toDisplay.
    exceptions ← defaultExceptions.
    fontno ← ¯1 "indicates font not set".
    text ← (para ← para asParagraph) text.
    self window: w
setfont: newfont liney: liney | |
    newfont = fontno
      ifTrue: [ "just bump y if same font"
        destY ← liney + style baseline - font ascent]
      ifFalse:
        [font ← style font: (fontno ← newfont).
        xtable ← font xtable.
        spacesize ← font spacewidth.
        minascii ← font minascii.
        maxascii ← font maxascii.
        glyphs ← font glyphs.
        printing
          ifTrue:
            [sourceRaster ← font raster.
            destY ← liney + style baseline - font ascent.
            height ← font height.
            source ← glyphs]]

Access to parts
chari | |
    ↑chari "index of last char scanned"
```

```
leftx | |
   ↑destX - width
rightx | |
   ↑destX
spacecount | |
   ↑0 - spacecount "because counted down from zero"
spacei | |
   ↑spacei
spacex | |
   ↑spacex

Scanning
emphasize: emphasis leftx: leftx rightx: rightx liney: liney | y y2 |
   "Should eventually use self callBLT, to get windowing and go faster"
   (font ≡ nil or: [font height = style lineheight])
    ifFalse:
     [font ascent < style baseline
       ifTrue:
        [y ← liney + (style baseline - font ascent).
        (leftx ⊙ liney rect: rightx ⊙ y) clear: white].
      font descent < (style lineheight - style baseline)
       ifTrue:
        [y ← liney + (style baseline + font descent).
        y2 ← y + (style lineheight - (style baseline + font descent)).
        (leftx ⊙ y rect: rightx ⊙ y2) clear: white]].
   emphasis = 0
    ifFalse:
     [emphasis ≥ 8
       ifTrue:
        [emphasis ← emphasis - 8 "Strike-out".
        y ← liney + (style baseline / 2).
        (leftx ⊙ y rect: rightx ⊙ (y + 1)) clear: black].
      emphasis ≥ 4
       ifTrue:
        [emphasis ← emphasis - 4 "Underline".
        y ← liney + style baseline.
        (leftx ⊙ y rect: rightx ⊙ (y + 1)) clear: black].
      emphasis ≥ 2
       ifTrue:
        [emphasis ← emphasis - 2 "Italic".
        y ← liney + style lineheight - 4.
        [y ≤ liney] whileFalseDo:
           [(leftx ⊙ liney rect: rightx - 1 ⊙ y) blt: leftx + 1 ⊙ liney mode: storing.
           (leftx ⊙ liney rect: leftx + 1 ⊙ y) clear: white.
           y ← y - 4]].
      emphasis ≥ 1
       ifTrue:
        [emphasis ← emphasis - 1 "Bold".
        (leftx ⊙ liney rect: rightx ⊙ (liney + style lineheight)) blt: leftx + 1 ⊙ liney
mode: oring]]
scanline: line at: y stopx: t3 stopchar: stopchar | ascii runx reason emphasis newrun
endrun padwidth relx tabsize spacepad t15 |
   stopx ← t3.
   chari ← line starti - 1.
   spacecount ← line spaces.
   padwidth ← line padwidth.
   destX ← spacex ← runx ← frame minX.
```

```
    emphasis ← charpad ← spacepad ← 0.
    tabsize ← style tabsize.
    spacei ← ¯1.
    (t15 ← para alignment) = 0
      ifFalse:
       [ "LeftFlush"
       t15 = 1
         ifTrue:
          [spacecount > 0
            ifTrue:
             [ "justified"
             charpad ← padwidth / (line stopi - chari + spacecount).
             charpad > 0
               ifTrue:
                [padwidth ← padwidth - (charpad * (line stopi - chari - 2 -
spacecount)).
                 printing
                   ifTrue: [(frame minX ⊙ y rect: frame maxX ⊙ (y + style lineheight))
clear: white]].
             spacepad ← padwidth / spacecount.
             spacecount ← spacecount - (padwidth \ spacecount)]]
         ifFalse:
          [t15 = 2
            ifTrue: [destX ← destX + (padwidth / 2)]
            ifFalse:
             [ "Centered"
             t15 = 4
               ifTrue: [destX ← destX + padwidth]]]] "RightFlush".
    reason ← 10.
    [reason < 10] whileFalseDo:
      [ "2=>stopx, 3=>stopchar, 4=>CR,
         10=>end of run, 11=>ascii out of range, 12-254=>exception chars"
       reason = 10
         ifTrue:
          [reason ← 255.
          chari ≥ stopchar
            ifTrue: [reason ← 3]
            ifFalse:
             [ "new run"
             printing
               ifTrue: [self emphasize: emphasis leftx: runx rightx: destX liney: y].
             runx ← destX.
             chari ← chari + 1.
             newrun ← para runAndVal: chari.
             emphasis ← newrun ⊙ 2 land: 15.
             endrun ← newrun ⊙ 1 + chari - 1 min: stopchar.
             self setfont: (newrun ⊙ 2 lshift: ¯4) liney: y]]
         ifFalse:
          [reason ← self scanword: endrun.
          reason > 10
            ifTrue:
             [reason = 12
               ifTrue:
                [spacecount ← spacecount - 1 "space".
                spacecount = 0
                  ifTrue: [spacepad ← spacepad + 1].
                spacei ← chari.
```

```
                spacex ← destX.
                width ← spacesize + spacepad.
                printing
                  ifTrue:
                   [super effect: 28.
                   self callBLT]]
                ifFalse:
                 [reason = 13
                   ifTrue:
                    [relx ← destX - frame minX "tab".
                    width ← (para alignment > 0
                           ifTrue: [tabsize + charpad]
                           ifFalse: [(relx + tabsize truncateTo: tabsize) - relx]).
                    printing
                      ifTrue:
                       [super effect: 28.
                       self callBLT]]
                   ifFalse:
                    [reason = 11
                      ifTrue: [ascii ← maxascii + 1].
                    sourceX ← xtable o (ascii + 1).
                    width ← xtable o (ascii + 2) - sourceX.
                    printing
                      ifTrue:
                       [super effect: 16.
                       self callBLT].
                    width ← width + charpad]].
              (destX ← destX + width) > stopx
                ifTrue: [reason ← 2]
                ifFalse:
                 [chari ≥ endrun
                   ifTrue: [reason ← 10]
                   ifFalse: [chari ← chari + 1]]]
            ifFalse:
             [reason = 4
               ifTrue: [width ← 0] "CR"]]].
    printing
      ifTrue: [self emphasize: emphasis leftx: runx rightx: destX liney: y]
scanword: endrun | ascii |
    <primitive: 76>
    "Scan or print text until terminated by x-value, special character or new format
run.
    Returns an Integer as follows:
       2=>stopx, 10=>end of run, 11=>ascii out of range,
       else exceptionsochari (4=CR, 12=space, 13=tab)"
    "exceptions and xtable are Vectors of Integers, printing is Boolean,
       rest are Integers"
    effect← 35.
    [chari > endrun] whileFalseDo:
       [ascii ← text o chari.
       exceptions o (ascii + 1) ≠ 0
         ifTrue: [↑exceptions o (ascii + 1)].
       (ascii < minascii or: [ascii > maxascii])
         ifTrue: [↑11].
       sourceX ← xtable o (ascii + 1).
       width ← xtable o (ascii + 2) - sourceX.
       printing
```

```
        ifTrue: [self callBLT].
      width ← width + charpad.
      destX ← destX + width.
      destX > stopx
        ifTrue: [↑2].
      chari ← chari + 1].
    chari ← chari - 1.
    ↑0
]
SystemOrganization classify: #TextScanner under: 'Text Objects'.]
TextScanner classInit]
```

Windows

```
'From Smalltalk-80 of February 8, 1981 on 17 February 1981 at 2:30:21 pm.'_!
                                                        "Window"
Class new title: #Window
   subclassof: Object
   fields: 'frame collapsed titlepara growing exitflag '
   declare: 'titleloc titleframe windowmenu titlerun border ';
   asFollows_!

This is a superclass for presenting windows on the screen.  Besides outlining and
scheduling the frame, it includes the distribution of user events which will someday
be driven by interrupts.

Initialization
classInit | |
    "Window classInit"
   border ← 2 ⊙ 2.
   titleframe ← Textframe new para: nil frame: nil.
   titleloc ← 3 ⊙ (¯4 - titleframe lineheight).
   titlerun ← String new: 2.
   titlerun word: 1 ← ¯255.
   windowmenu ← Menu new string: 'under
frame
close
print
printbits

reset | |
   exitflag ← true.
   growing ← false

Scheduling
eachtime | |
   (frame has: user mp)
    ifTrue:
     [user kbck
       ifTrue: [↑self kbd].
      user anybug
       ifTrue:
        [user redbug
          ifTrue: [↑self redbug].
         user yellowbug
          ifTrue: [↑self yellowbug].
         user bluebug
          ifTrue: [↑self bluebug]]
       ifFalse:
        [user anykeys
          ifTrue: [↑self keyset]]]
     ifFalse:
      [self outside
       ifFalse:
        [user anybug
          ifTrue:
           [(frame has: user mp)
             ifFalse: [↑false]]
          ifFalse:
           [user kbck
             ifTrue:
```

```
                [user kbd.
                frame flash] "flush typing outside"]]]
firsttime | |
   (frame has: user mp)
     ifTrue:
      [self reset.
      ↑self enter].
   ↑false
lasttime | |
   self leave.
   ↑exitflag

Framing
clearTitle: color | |
   (titleframe window inset: ¯2 ⊙ ¯2) clear: color
editTitle | pared w |
   pared ← TextImage new para: titlepara frame: nil.
   pared formerly: false.
   pared fixframe: titleframe window + (1 ⊙ 2).
   pared enter.
   w ← titleframe window.
   [user anybug and: [(w has: user mp) ≡ false]] whileFalseDo:
      [user kbck
        ifTrue: [pared kbd]
        ifFalse:
         [user redbug
           ifTrue:
            [(w has: user mp)
              ifTrue: [pared selecting]]
           ifFalse:
            [user yellowbug
              ifTrue:
               [(w has: user mp)
                 ifTrue: [w flash]]]]].
   titlepara ← pared contents.
   self showtitle
erase | |
   (frame inset: ¯2 ⊙ ¯2) clear.
   self clearTitle: background
fixframe: f | |
   ↑Rectangle new origin: f origin extent: (f extent max: 32 ⊙ 32)
frame | |
   ↑frame
frame: f | |
   frame ← self fixframe: f
newframe | a oldframe |
   user waitnobug.
   user restoredisplay.
   frame ≡ nil
     ifFalse:
      [self aboutToFrame.
      self erase].
   a ← OriginCursor showwhile: [user waitbug].
   growing ← true.
   frame ← self fixframe: (a rect: a + 32).
   frame outline.
   CornerCursor showwhile: [([a ← user mpnext] whileTrueDo:
```

```
        [oldframe ≡ nil
          ifTrue: [user cursorloc← a max: frame corner].
        oldframe ← frame copy.
        frame ← self fixframe: (frame growto: a).
        (oldframe inset: ¯2) clear.
        frame outline])].
    self frame: frame.
    growing ← false.
    self takeCursor
outline | |
    frame outline "Clear and outline me."
show | |
    self outline.
    growing
      ifFalse: [self showtitle]
showtitle | |
    titlepara ≡ nil
      ifTrue: [titlepara ← Paragraph new text: self title runs: titlerun alignment: 0].
    titleframe put: titlepara at: frame origin + titleloc.
    titleframe outline
takeCursor | |
    user cursorloc← frame center "Move the cursor to my center."
title | |
    ↑'Untitled'

Default Event responses
aboutToFrame | |
    "My frame is about to change.  I dont care."
bluebug | t1 |
    (t1 ← windowmenu bug) = 1
      ifTrue: [↑exitflag ← false].
    t1 = 2
      ifTrue:
       [self newframe.
       self enter]
      ifFalse:
       [t1 = 3
         ifTrue:
          [self close.
          self erase.
          user unschedule: self.
          ↑false].
        t1 = 4
          ifTrue: [self hardcopy]
          ifFalse:
           [t1 = 5
             ifTrue: [self print]]]
close | |

enter | |
    self show
hardcopy | |
    frame flash
kbd | |
    user kbd.
    frame flash
keyset | |
```

```
    frame flash
leave | |

outside | |
    (titleframe window has: user mp)
      ifTrue:
       [user anybug
         ifTrue: [self editTitle]
         ifFalse: [↑false]]
      ifFalse: [↑false]
print | t1 |
    (t1 ← dp0 pressfile: (self title + '.press.') asFileName) screenout: frame scale:
PressScale.
    t1 toPrinter
redbug | |
    frame flash
yellowbug | |
    frame flash
!
SystemOrganization classify: #Window under: 'Windows'.!
Window classInit!
```

"PanedWindow"

```
Class new title: #PanedWindow
   subclassof: Window
   fields: 'panes templates title'
   declare: '';
   asFollows_]
```

A paned window is a Window that has subwindows (panes) that are awakened and resized in unison. The instance variable templates is a set of Rectangles for the frames of the panes normalized such that the whole PanedWindow is a frame of 0⊙0 rect: 36⊙36.

```
Initialization
title: t1 with: t2 at: t3 | pane |
   "The instance variable templates is a set of Rectangles for the frames of the panes
normalized such that the whole PanedWindow is a frame of 0⊙0 rect: 36⊙36."
   title ← t1.
   panes ← t2.
   templates ← t3.
   self reset.
   panes do: [:pane | pane init]

Window protocol
close | pane |
   panes do: [:pane | pane close]
eachtime | pane |
   (frame has: user mp)
    ifTrue:
     [user bluebug
       ifTrue: [↑self bluebug].
      panes do: [:pane | pane startup]]
    ifFalse:
     [self outside
       ifFalse:
        [user anybug
          ifTrue:
           [(frame has: user mp)
             ifFalse: [↑false]]
          ifFalse:
           [user kbck
             ifTrue:
              [user kbd.
               frame flash] "flush typing outside"]]]
enter | pane |
   super show.
   panes do: [:pane | pane windowenter]
erase | |
   self titlerect clear.
   super erase
fixframe: f | |
   ↑Rectangle new origin: f origin extent: (f extent max: 160 ⊙ 80)
frame: t1 | templateStream template pane orig ext |
   "(Re)initialize my frame, and tell my panes their locations."
   frame ← t1.
   templateStream ← templates asStream.
   orig ← frame origin - 1.
   ext ← frame extent + 2.
```

```
    panes do:
        [:pane | template ← templateStream next "It would be nice to have parallel fors
as in MLISP.".
        pane frame← template * ext / 36 + orig inset: 1]
hardcopy | p |
    user displayoffwhile: [
        (p ← dpO pressfile: (self title + '.press') asFileName.
        self hardcopy: p.
        p close.
        p toPrinter)]
hardcopy: pf | pane |
    self hardcopyTitle: pf "print frame rectangle".
    frame hardcopy: pf "print all panes".
    panes do: [:pane | pane hardcopy: pf "print cursor if it's inside"].
    (frame has: user mp)
      ifTrue: [user currentCursor hardcopy: pf]
hardcopyTitle: pf | |
    self showtitle "refresh title (since it's a class var)" "draw title rectangle".
    titleframe window hardcopy: pf "print title text (make frame larger)".
    titleframe para presson: pf in: (pf transrect: (titleframe frame origin rect:
titleframe frame corner + (999 ⊙ 2))) style: titleframe style
kbd | pane |
    (pane ← self pickedpane)
      ifTrue: [↑pane kbd]
keyset | pane |
    (pane ← self pickedpane)
      ifTrue: [↑pane keyset]
leave | pane |
    panes do: [:pane | pane windowleave]
pickedpane | pane |
    panes do:
        [:pane | pane picked
          ifTrue: [↑pane]].
    frame flash.
    ↑false
redbug | pane |
    (pane ← self pickedpane)
      ifTrue: [↑pane redbug]
show | pane |
    super show.
    panes do: [:pane | pane outline]
takeCursor | |
    (panes o 1) takeCursor
title | |
    ↑title
yellowbug | pane |
    (pane ← self pickedpane)
      ifTrue: [↑pane yellowbug]

Private
titlerect | |
    ↑frame origin - (2 ⊙ (DefaultTextStyle lineheight + 4)) rect: frame corner x ⊙
frame origin y + (2 ⊙ 0)
_]
SystemOrganization classify: #PanedWindow under: 'Windows'._]
```

"BrowseWindow"

```
Class new title: #BrowseWindow
    subclassof: PanedWindow
    fields: ''
    declare: 'stdTemplates ';
    asFollows_!

"I am a five-paned window to browse through classes.  My panes are...
    system pane: categories of classes in the system
    class pane: classes in the selected category
    organization pane: categories of methods in the selected class
    selector pane: method selectors in the selected category
    code pane: source code of the selected method, if any, else other useful info"

"Initialization
classInit | |
    stdTemplates ← {(0 ⊙ 0 rect: 10 ⊙ 14) , (10 ⊙ 0 rect: 18 ⊙ 14) , (18 ⊙ 0 rect: 28 ⊙
14) , (28 ⊙ 0 rect: 36 ⊙ 14) , (0 ⊙ 14 rect: 36 ⊙ 36)}
default | systemPane classPane orgPane selectorPane codePane |
    "Let the user draw a five-paned window to browse through classes."
    systemPane ← SystemPane new "Create the panes.".
    classPane ← ClassPane new.
    orgPane ← OrganizationPane new.
    selectorPane ← SelectorPane new.
    codePane ← CodePane new "Acquire them.".
    self title: 'Classes' with: {systemPane , classPane , orgPane , selectorPane ,
codePane} at: stdTemplates.
    self newframe.
    self show "Interconnect them.".
    systemPane to: classPane.
    classPane from: systemPane to: orgPane.
    orgPane from: classPane to: selectorPane.
    selectorPane from: orgPane to: codePane.
    codePane from: selectorPane "Display them.".
    systemPane update
_!
SystemOrganization classify: #BrowseWindow under: 'Windows'._!
BrowseWindow classInit_!
```

```
                                                    "CodeWindow"
Class new title: #CodeWindow
   subclassof: PanedWindow
   fields: ''
   declare: 'stdTemplates ';
   asFollows_]

I am a paned window with a code pane to edit a method or a file.

Initialization
class: class selector: selector para: para formerly: oldpara | codePane |
   codePane ← CodePane new class: class selector: selector para: nil.
   self title: class title + ' ' + selector with: codePane inVector at: stdTemplates.
   self newframe.
   self show.
   codePane showing: para.
   codePane formerly: oldpara.
   codePane from: codePane
classInit | |
   stdTemplates ← (0 ⊙ 0 rect: 36 ⊙ 36) inVector
editTitle | |
   titleframe window flash
file: file | filePane |
   filePane ← FilePane new file: file.
   self title: file name with: filePane inVector at: stdTemplates.
   self newframe.
   self show.
   filePane showing: file contents asParagraph.
   filePane from: filePane
hardcopy: p | pane |
   panes do: [:pane | pane hardcopy: p]
_]
SystemOrganization classify: #CodeWindow under: 'Windows'._]
CodeWindow classInit_]
```

"ProjectWindow"

```
Class new title: #ProjectWindow
   subclassof: Window
   fields: 'userview parent changes'
   declare: 'actionMenu ';
   asFollows_!
```

A ProjectWindow represents its userview as a window to provide access to many UserViews, each for a different "project". Besides the state in the userview, they also carry their own hashset for changes, so that such changes can be maintained on a per-project basis. parent specifies another ProjectWindow to which control is given when the user leaves the current userview

```
Changing views
install | |
    "Establish this project and its userview as the current screen view"
   Changes ← changes.
   (user ← userview) install.
   self putTitle.
   user restart
putTitle | |
   titlepara ≡ nil
     ifTrue: [titlepara ← 'Top View' asParagraph allBold].
   titleframe put: titlepara centered: user screenrect extent x / 3 ⊙ 8.
   titleframe outline
runParent | |
    "leave this view by installing the one above"
   parent install

Window behavior
close | |
    "break circular links"
   userview ← parent ← nil
yellowbug | |
   actionMenu bug = 1
     ifTrue: [self install]

Initialization
classInit | |
   actionMenu ← Menu new string: 'enter'
init | |
    "a new window"
   self userview: (user copyIn: self) changes: HashSet init parent: user projectWindow.
   self newframe.
   self show
userview: t1 changes: t2 parent: t3 | |
    "load state"
   userview ← t1.
   changes ← t2.
   parent ← t3
_!
SystemOrganization classify: #ProjectWindow under: 'Windows'._!
ProjectWindow classInit_!
```

Panes and Menus

"ListPane"

```
Class new title: #ListPane
   subclassof: Textframe
   fields: 'list firstShown lastShown selection scrollBar'
   declare: '';
   asFollows_]
```

A list pane displays a vertical list of one-line items. The list can be scrolled slow or fast, and any item can be selected. When an item is selected (or deselected), a dependent pane can be told to display appropriate material.

```
initialization
of: t1 | |
    "Acquire the specified list and show me scrolled to the top"
   list ← t1.
   selection ← 0.
   self makeParagraph.
   self frame← window.
   self outline.
   self displayall.
   self deselected
revise: newlist with: sel | changing |
    "Acquire newlist. Do not change firstShown. Select sel if in list."
   (changing ← list ≠ newlist)
     ifTrue:
      [list ← newlist.
      window clear: white.
      self makeParagraph.
      self fill.
      self displayall]
     ifFalse:
      [selection > 0
        ifTrue:
         [(changing ← list o selection ≠ sel)
           ifTrue: [self compselection]]
        ifFalse: [changing ← true]].
   changing
     ifTrue:
      [selection ← ¯1.
      self select: (list find: sel)]
select: lineNum | oldSel |
   oldSel ← selection "Select my non-dummy displayed entry whose subscript is
lineNum; highlight it; if it is different from selection, tell me to select. If there is no
such entry, set selection to 0 and if it wasnt 0 before, tell me to deselect.".
   ((1 max: firstShown - 1) ≤ lineNum and: [lineNum ≤ (list length min: lastShown -
1)])
     ifTrue:
      [selection ← lineNum.
      self compselection.
      oldSel ≠ selection
        ifTrue: [self selected]]
     ifFalse:
      [selection ← 0.
      oldSel ≠ selection
        ifTrue: [self deselected]]

Pane protocol
```

```
close | |
    "Zero my selection so it wont be grayed when I close.  Break cycles."
  selection ← 0.
  scrollBar close
eachtime | |
  (window has: user mp)
    ifTrue:
     [user kbck
       ifTrue: [↑self kbd].
      user anybug
       ifTrue:
        [user redbug
          ifTrue: [↑self redbug].
         user yellowbug
          ifTrue: [↑self yellowbug].
         user bluebug
          ifTrue: [↑false]]
       ifFalse:
        [user anykeys
          ifTrue: [↑self keyset]]]
    ifFalse: [↑self outside]
enter | |
  scrollBar show
firsttime | |
  (window has: user mp)
    ifTrue: [self enter]
    ifFalse: [↑false]
frame← t1 | |
    "(Re)initialize my window"
  window ← t1.
  (frame ← window inset: 2 ⊙ 0 and: [0 ⊙ 0]) width← 999.
  self fill.
  scrollBar ← (scrollBar ≡ nil
         ifTrue: [ScrollBar new]
         ifFalse: [scrollBar]) on: window from: self
hardcopy: pf | t cr first last lasty lineNum parag left right lineheight |
  window hardcopy: pf thickness: 1.
  para ≡ nil
    ifTrue: [self makeParagraph].
  parag ← para asParagraph.
  t ← para asStream.
  last ← 0.
  cr ← 13.
  left ← frame minX.
  right ← window maxX.
  lasty ← frame minY.
  lineheight ← self lineheight.
  (firstShown to: lastShown) do:
     [:lineNum | first ← last.
     ((t skipTo: cr) or: [lineNum = lastShown])
       ifTrue: [last ← t position]
       ifFalse: [user notify: 'not enough lines'].
     (lineNum = selection and: [selection > 0])
       ifTrue: [(self selectionRect "outline selection; complementing doesn't look good"
- (0 ⊙ 1) inset: 0 ⊙ 1) hardcopy: pf thickness: 1].
     (parag copy: first + 1 to: last - 1) presson: pf in: (pf transrect: (left ⊙ lasty rect:
right ⊙ (lasty + lineheight + 4))) style: style.
```

```
      lasty ← lasty + lineheight]
kbd | |
   window flash.
   user kbd
keyset | c t2 |
   c ← user currentCursor "As long as any keyset keys are down, react to keys 2 and
8 down by scrolling up or down a line at a time. If key 4 is down as well, scroll
faster.".
   self scrollControl: [
      ((t2 ← user keyset) = 6
        ifTrue: [2]
        ifFalse:
         [t2 = 12
           ifTrue: [-2]
           ifFalse:
            [t2 = 2
              ifTrue: [1]
              ifFalse:
               [t2 = 8
                 ifTrue: [-1]
                 ifFalse: [0]]]])].
   c show
lasttime | |
   self leave
leave | |
   scrollBar hide
outline | |
   window outline: 1
outside | |
   ↑scrollBar startup
picked | |
   ↑window has: user mp
redbug | newSel f |
    "Deselect selection and select cursor item, if any"
   (f ← self locked)
    ifFalse:
     [self compselection.
     newSel ← ((self lineofy: user mp y) max: 1) - 1.
     XeqCursor showwhile: [(self select: (newSel = selection
          ifTrue: [0]
          ifFalse: [newSel]))]].
   [user redbug and: [(window has: user mp)]] whileTrueDo:
     [f
       ifTrue:
        [f flash.
        self compselection.
        self compselection]]
scrollPos | |
   ((firstShown ≡ nil or: [list ≡ nil]) or: [list length = 0])
    ifTrue: [↑0.0].
   ↑(firstShown - 1) asFloat / list length
scrollTo: f | place y |
   ↑self scrollUp: (f * lastline) asInteger - firstShown + 1 * style lineheight
windowenter | |
    "Refresh my image. Reaffirm selection."
   self outline.
   self displayall.
```

```
    self compselection
windowleave | |
    self compselection.
    self grayselection
yellowbug | |
    window flash

Subclass defaults
deselected | |
    "I just lost my selection.  I dont care, but my subclasses might."
dirty | |
    "My subclasses may want to prohibit a change of selection"
    ↑false
locked | |
    "My subclasses may want to prohibit a change of selection"
    ↑selection = 0
      ifTrue: [false]
      ifFalse: [self dirty]
selected | |
    "A new selection is highlighted.  I dont care, but my subclasses might"

Private
compselection | |
    "If I have a selection, complement its image."
    selection ≠ 0
      ifTrue: [self selectionRect comp]
displayall | |
    self displaylines: firstShown to: lastShown
dummy | |
    ↑'~~~~~~~~~~~~~~~'
fill | |
    firstShown ≡ nil
      ifTrue: [self makeParagraph]
      ifFalse: [lastShown ← self lineofy: window maxY - (style lineheight - 1)].
    (self locked or: [selection > 0])
      ifTrue:
       [selection < firstShown
         ifTrue: [frame origin y← window minY - (selection * style lineheight) + style
lineheight]
         ifFalse:
          [selection > lastShown
            ifTrue: [frame origin y← window minY - (selection * style lineheight + style
lineheight) + (window maxY - window minY truncateTo: style lineheight) min:
window minY]]].
    firstShown ← self lineofy: window minY + (style lineheight - 1).
    lastShown ← self lineofy: window maxY - (style lineheight - 1)
firstShown: new | delta deltaPt |
    delta ← new - firstShown.
    deltaPt ← 0 ⊙ (0 - delta * style lineheight) "negative".
    firstShown ← new.
    lastShown ← lastShown + delta.
    frame moveby: deltaPt.
    (window origin - deltaPt rect: window corner) blt: window origin mode: storing.
    self displaylines: (firstShown max: lastShown - delta + 1) to: lastShown
grayselection | |
    selection ≠ 0
      ifTrue: [self selectionRect color: ltgray mode: oring]
```

```
init | |
    self para: nil frame: nil
lastShown: new | delta deltaPt |
    delta ← new - lastShown "negative".
    deltaPt ← 0 ⊙ (0 - delta * style lineheight) "positive".
    lastShown ← new.
    firstShown ← firstShown + delta.
    frame moveby: deltaPt.
    (Rectangle new origin: window origin extent: window width ⊙ (window height -
deltaPt y truncateTo: style lineheight)) blt: window origin + deltaPt mode: storing.
    self displaylines: firstShown to: (firstShown - delta - 1 min: lastShown)
makeParagraph | i len s |
    "Given firstShown, compute lastShown."
    firstShown ≡ nil
      ifTrue:
       [firstShown ← 1.
       selection ← 0].
    s ← (String new: 200) asStream.
    s append: self dummy.
    s cr.
    list ≡ nil
      ifFalse: [(1 to: list length) do:
          [:i | list o i printOn: s.
          s cr]].
    s append: self dummy.
    s cr.
    para ← s contents asParagraph.
    self measureall.
    lastShown ← self lineofy: window maxY - (style lineheight - 1).
    frame corner y← (self yofline: lastline) + style lineheight.
    selection > lastShown
      ifTrue: [selection ← 0]
scrollUp: amount | linetomove |
    self compselection.
    amount ← amount truncateTo: style lineheight.
    linetomove ← (self lineofy: window minY + amount "abs") max: 1.
    amount > 0
      ifTrue:
       [lastline = lastShown
         ifFalse:
          [self locked
            ifTrue:
             [selection + 1 = firstShown
               ifTrue: [window flash]
               ifFalse: [self firstShown: (linetomove min: selection + 1)]]
            ifFalse: [self firstShown: (linetomove min: lastline - (window height / style
lineheight) + 1)]]]
      ifFalse:
       [firstShown = 1
         ifFalse:
          [self locked
            ifTrue:
             [selection + 1 = lastShown
               ifTrue: [window flash]
               ifFalse: [self lastShown: (lastShown - (firstShown - linetomove) max:
selection + 1)]]
            ifFalse:
```

```
        [.
        self lastShown: (lastShown - (firstShown - linetomove) max: (lastline min:
window height / style lineheight))]]].
    (window origin x ⊙ ((self yofline: lastline) + style lineheight) rect: window corner)
clear: white.
    ↑self select: selection
selectionRect | newy |
    ↑(Rectangle new "I have a selection. Return its highlighting rectangle." origin:
frame minX ⊙ (newy ← selection * style lineheight + frame minY) corner: frame
maxX ⊙ (newy + style lineheight)) intersect: window
]
SystemOrganization classify: #ListPane under: 'Panes and Menus'.]
```

Interpreter

"ContextPart"

```
Class new title: #ContextPart
   subclassof: Object
   fields: 'sender pc stackp'
   declare: '';
   asFollows_]
```

This class has not yet been commented

```
As yet unclassified
erase | i |
    "release frames to break cycles"
   (1 to: stackp) do: [:i | self o i ← nil]
mclass | mclass |
   self receiver class invertMethod: self method inClass: [:mclass].
   ↑mclass
pop | val |
   val ← self o stackp.
   stackp ← stackp - 1.
   ↑val
printOn: strm | mclass selector class |
   selector← (class← self receiver class) invertMethod: self method inClass: [:mclass].
   strm append: class title.
   mclass ≡ class
     ifFalse: [strm append: '('; append: mclass title; append: ')'].
   strm append: '>>'.
   strm append: selector
push: val | |
   ↑self o (stackp ← stackp + 1) ← val
release | | self releaseTo: nil
releaseFully | | self releaseTo: nil
releaseTo: caller | c | "release frames to break cycles"
   c ← self.
   [c ≡ nil or: [c≡caller]] whileFalseDo:
      [c erase.
      c ← c sender]
selector | |
   ↑self receiver class invertMethod: self method inClass: [:ignored]
sender | |
   ↑sender
sender← t1 | |
   sender ← t1.
   ↑sender
sourceCode | mclass selector |
   selector← self receiver class invertMethod: self method inClass: [:mclass].
   ↑(mclass canunderstand: selector)
      ifTrue: [mclass code: selector]
      ifFalse: [(LADDecompiler new decompile: selector
            in: self receiver class method: self method) asString]
stack | a strm |
   strm ← (Vector new: 20) asStream.
   strm next← a ← self.
   [(a ← a sender) ≡ nil] whileFalseDo:  [strm next← a].
   ↑strm contents
tempframe | i |
   ↑(1 to: self method numTemps) collect: [:i | self tempAt: i]
tempNames | |
```

```
↑LADParser new parseArgsAndTemps: self sourceCode asStream notifying: false
SystemOrganization classify: #ContextPart under: 'Interpreter'.
```

"BlockContext"

```
Class new title: #BlockContext
   subclassof: ContextPart
   fields: 'nargs startpc home'
   declare: '';
   bytesize: true;
   asFollows_]
```

This class has not yet been commented

```
As yet unclassified
blockCopy: t1 | |
   nargs ← t1.
   ↑(BlockContext new: self length) home: home startpc: pc + 2 nargs: nargs
erase | |
   home ← nil.
   super erase
eval | save val |
   nargs = 0
     ifTrue: [↑self value].
   nargs = 1
     ifTrue:
      [save ← startpc "amazing crock for reading ST76 remote variable".
      startpc ← startpc + (self method o startpc < 112
            ifTrue: [1]
            ifFalse: [2]).
      nargs ← 0.
      val ← self value.
      startpc ← save.
      nargs ← 1.
      ↑val].
   self valueError
home || ↑home
home: t1 startpc: t2 nargs: t3 | |
   home ← t1.
   startpc ← t2.
   nargs ← t3
method | |
   ↑home method
printOn: strm ||
   strm append: '[] in '.
   super printOn: strm
receiver | |
   ↑home receiver
tempAt: index | |
   ↑home o index
tempAt: index put: value | |
   ↑home o index ← value
value | |<primitive: 26>
   self valueError
value: arg | |<primitive: 27>
   self valueError
valueError | |
   user notify: 'Incompatible number of args'
value← val | |
    "for compatibility with ST76 remote variables"
   ↑self value: val
```

```
SystemOrganization classify: #BlockContext under: 'Interpreter'.
```

"Message"

```
Class new title: #Message
   subclassof: Object
   fields: 'selector args'
   declare: '';
   asFollows_!
```

This class has not yet been commented

```
As yet unclassified
args | |
   ↑args
selector | |
   ↑selector
!
SystemOrganization classify: #Message under: 'Interpreter'._!
```

"MethodContext"

```
Class new title: #MethodContext
    subclassof: ContextPart
    fields: 'method receiverMap receiver'
    declare: '';
    bytesize: true;
    asFollows_!

This class has not yet been commented

Initialization
goBaby | width height |
    width ← 640.
    height ← 480.
    user currentDisplay: (DisplayForm new extent: width@height bits:
        (DisplayBitmap new: width/16 * height) offset: nil).
    user currentCursor: NormalCursor.
    [true] whileTrueDo: [self run]
restart | |
    pc ← method initialPC.
    stackp ← method numArgs + method numTemps
run | |
    self setRestart: [(↑nil)].
    user restore.
    user run
setRestart: code | |
    RestartCode ← code
systemRestart | |
    thisContext releaseTo: RestartCode home.
    NormalCursor show.
    RestartCode eval

Access to Parts
method | |
    ↑method
receiver | |
    ↑receiver
tempAt: index | |
    ↑self ○ index
tempAt: index put: value | |
    ↑self ○ index ← value

Blocks
blockCopy: nargs | |
    ↑(BlockContext new: self length) home: self startpc: pc + 2 nargs: nargs
remoteCopy | |
    ↑self blockCopy: 0

Debugger
erase | |
    receiver ← nil.
    super erase
_!
SystemOrganization classify: #MethodContext under: 'Interpreter'._!
```