"kernel classes"
"Version 3.5 6 Sept 77"

"Class"

```
Class new title: 'Class';
  fields: 'title "<String> for identification, printing"
       myinstvars "<String> partnames for compiling, printing"
       instsize "<Integer> for storage management"
       messagedict "<MessageDict> for communication, compiling"
       classvars "<Dictionary/nil> compiler checks here"
       superclass "<Class> for execution of inherited behavior"
       environment "<Vector of SymbolTables> for external refs"
       fieldtype';
  asFollows!
```

Classes are the molecules of Smalltalk. The instance fields specify the number and naming of fields for each instance, and the messages define the protocol with which these objects may be communicated. Classes inherit the fields and message protocol of their superclass. Locally defined messages will override inherited ones of the same name, and overriden ones may be accessed through the use of super in place of self. A typical class definition looks like:

```
Class new title: 'CodeEditor';
    subclassof: Window;
    fields: 'pared class selector';
    declare: 'editmenu'
```

This ordering is required, though the subclassof: and declare: messages are optional. A class definition may be re-executed but, if the fields: clause has changed, all instances of the old class will become obsolete (they will fail to respond to any messages).

Initialization

```
abstract
    [self fields: nullString]
bytesize: n "non-pointer declaration"
    [self≠self realself⇒[self realself bytesize: n]
    fieldtype ← 32+ [n=8⇒ [8] 16]]
declare: v
    [self≠self realself⇒[self realself declare: v]
        [classvars@nil⇒[classvars ← SymbolTable init]].
    classvars declare: [v is: String⇒[v asVector] v]]
environment ← environment [] "for resetting to reread sharing clauses"
fields: myinstvars        "list of instance variables"
    [  [self≠self realself⇒
          [self instvars≠self realself instvars⇒
              [user notify: 'All '+title+'s become obsolete if you proceed...'.
              self realself obsolete.      "if he chooses to proceed"
              Smalltalk·title unique ← self]
          self realself environment← nil; subclassof: superclass.
          ⇑self]].
    fieldtype ← 16.
    instsize ← self instvars length.
    instsize>16⇒
        [user notify: 'too many instance variables']
    messagedict ← MessageDict init.
    self organization]
obsolete     "invalidate further communication"
    [title ← 'AnObsolete'+title.
    classvars ← nil.      "recycle class variables"
    messagedict init.   "invalidate and recycle local messages"
    environment ← self.   "keep me around for old instances"
    "Note that superclass messages will continue to function;
    superclass←nil would inhibit that, but would also prevent default
    printing, which the debugger may invoke"]

realself [⇑Smalltalk·title unique]    "as opposed to possible filin ghost"
sharing: table
    [self≠self realself⇒[self realself sharing: table]
    environment ← environment asVector , table]
subclassof: superclass
title: title
```

```
    [self title: title unique insystem: Smalltalk]
title: name insystem: system
    [superclass ← Object.
    system has: name⇒[]
    system define: name as: self.
    AllClassNames ← AllClassNames insertSorted: name.
    SystemOrganization classify: name under: 'As yet unclassified']
veryspecial: n      "inaccessible fields"
    [instsize ← instsize+n]
```

Access to parts

```
'x [⇑classvars'x]
'x ← val [⇑classvars'x ← val]
asName  "Return my title, which is assumed to be 99 characters or fewer without any CRs."
    [⇑title]
instsize
    ["Return the number of instance words that can be accessed by instfield:."
    ⇑[fieldtype≥32⇒ [0] self@Class or⁸ self@VariableLengthClass⇒ [instsize-1] instsize]]
instvars
    [superclass@nil ⇒ [⇑myinstvars asVector]
    ⇑superclass instvars concat: myinstvars asVector]
md [⇑messagedict]
superclass [⇑superclass]
title [⇑title]
```

Organization

```
classvars [⇑classvars]
environment
    [⇑classvars asVector concat: environment asVector]
organization | o
    [   [classvars @ nil⇒[self declare: ☞ClassOrganization]].
    o ← classvars'☞ClassOrganization.
    o is: ClassOrganizer⇒[⇑o]
    o ← ClassOrganizer new init: messagedict contents sort.
    classvars'☞ClassOrganization ← o.  ⇑o]
```

Editing

```
ed: selector | c s
    [c← self code: selector. user clearshow: c.
    while⁸ (s← user request: 'substitute: ') do⁸
        [c ← c subst: s for: (user request: 'for: ').
        user clearshow: c]
    self understands: c]
edit: selector | para s v
    [para ←
        [selector=☞ClassOrganization⇒
            [self organization asParagraph]
        messagedict has: selector⇒[self code: selector]
        nullString asParagraph].
    user schedule:
        (CodeEditor new class: self selector: selector para: para)]
execute: code       "disposable methods"
    [self understands: 'doit [⇑' + code + ']'.
    ⇑self new doit]
fixcode | selector t        "make patterns bold and compress paragraphs"
    [   [myinstvars has: 032⇒
            [myinstvars ← myinstvars asParagraph fromBravo text]].
    for⁸ selector from: messagedict do⁸
        [t ← self code: selector.
        t text has: 032⇒
            [messagedict code: selector ←
            t makeBoldPattern packIntoString]]]
```

Message access

```
bytesof: sel
    [⇑(messagedict method: sel) asBytes]
canunderstand: selector
    [⇑messagedict has: selector]
```

```
code: selector
    [selector=☞ClassOrganization⇒
            [⇑self organization asParagraph]
    ⇑Paragraph new unpackFromString: (messagedict code: selector)]
compileall | s
    [for☺ s from: messagedict do☺
        [self understands: (messagedict code: s)]]
derstands: selector | c        "overstands?  undersits? - forget it"
    [messagedict ← messagedict delete: selector.
    self organization delete: selector.
    Changes has: (c←title+' '+selector)⇒ [Changes delete: c]]
install: name method: method literals: literals
        code: code backpointers: backpointers
    [messagedict ← messagedict insert: name method: method
        literals: literals code: code asParagraph packIntoString
        backpointers: backpointers.
    name=☞doit⇒[]; =☞debugEval:⇒[]
    Changes insert: title+' '+name]
messages [⇑messagedict contents , ☞ClassOrganization]
selectors        "Return a Vector of all my selectors."
    [⇑self messages]
shrink [messagedict ← messagedict shrink]
space | a s
    [s ← 0. for☺ a from: messagedict do☺
        [s ← s + (messagedict method: a) length]
    ⇑s]
understands: code | selector old        "install method"
    [old ← messagedict.
    selector ← user displayoffwhile☺ [Compiler new compile: code in: self].
    self organization classify: selector under: 'As yet unclassified'.
    ⇑selector]
understands: code classified: heading | selector        "install method"
    [selector ← user displayoffwhile☺ [Compiler new compile: code in: self].
    self organization classify: selector under: heading.  ⇑selector]
whosends: selector | s l a
    [s ← Stream default.
    for☺ a from: messagedict do☺
        [for☺ l from: (messagedict literals: a) do☺
            [selector@l⇒[s append: a; space]]]
    ⇑s contents]
```

Instance access

```
allinstances [user croak] primitive: 60
copy: inst | t i
    [t ← self new.
    for☺ i to: instsize do☺
        [t instfield: i ← inst instfield: i]
    ⇑t]
default
    [⇑self new default]
init        "init and default get propagated to instances"
    [⇑self new init]
new [user croak] primitive: 28
print: inst on: strm | ivars i
    [ivars ← self instvars.
    strm append: '('; append: title; append: ' new '.
    for☺ i to: instsize do☺
        [strm append: ivars*i; append: ': ';
            print: (inst instfield: i); space]
    strm append: ')']
printon: strm
    [strm append: 'Class ' + title]
recopy: inst | t i
    [t ← self new.
    for☺ i to: instsize do☺
        [t instfield: i ← (inst instfield: i) recopy]
    ⇑t]
```

Filin and Filout

```
asFollows | s heading trailer
    [self≠self realself⇒[self realself asFollows]
    FilinSource upto: 015; ≪ 015.
    until§ FilinSource≪036 do§
        [s ← FilinSource upto: 032.
        trailer ← FilinSource upto: 015.
        FilinSource ≪ 015.
        trailer = '\ig'⇒[self organization globalComment ← s];
                = '\f5bg'⇒[heading ← s]
        s ← s asParagraph applyBravo: trailer at: 1 to: s length.
        user show: [heading @ nil⇒[self understands: s]
                self understands: s classified: heading].
        user space]
    FilinSource upto: 015]
definition | strm       "return a string that defines me (Class new title etc.)"
    [strm ← (String new: 50) asStream.
    self printdefon: strm.
    ⇑strm contents]
filout | f
    [user displayoffwhile§
        [f ← dp0 file: title+'.st.'.
        self fullprinton: f.
        f shorten; close]]
filoutOrganization | f       "So we can merge separate work on organization"
    [user show: title; cr.
    user displayoffwhile§
        [f ← dp0 file: title+'.org.'.
        f append: title+' organization fromParagraph:'; cr.
        f append: self organization asParagraph text asString.
        f append: 'asParagraph!'; shorten; close]]
fullprinton: strm | s heading org
    [user show: title; cr.
    strm append: '"'; append: title; append: '"'; ctlz: 'z20000115000\f5bg'.
    self printdefon: strm.
    strm semicrtab; append: 'asFollows'; next← 036; ctlz: '\f5bg'.
    org ← self organization.
    strm cr; append: org globalComment; ctlz: '\ig'.
    for§ heading from: org categories do§
        [strm cr; append: heading; ctlz: '\f5bg'.
        for§ s from: (org category: heading) do§
            [s=☞doit⇒[]; =☞debugEval:⇒[]
            strm append: (self code: s) makeBoldPattern toBravo text]]
    [self canunderstand: ☞classInit⇒[strm append: '('+title.
            strm append: [self is: Class⇒[' new'] ' new: 1']+') classInit!'; ctlz: '\g']].
    strm next← 036; ctlz: '\g']
printdefon: strm | s v       "print my definition on strm"
    [strm append: 'Class new title: ' + title asString.
        [superclass=Object⇒[]
        strm semicrtab; append: 'subclassof: ' +
            [superclass@nil⇒['nil'] superclass title]].
    strm semicrtab; append: 'fields: ' + myinstvars asString.
        [classvars@nil⇒[]
        (v ← classvars contents) = ☞(ClassOrganization)⇒[]
        strm semicrtab; append: 'declare: '".
        for§ s from: v do§
            [s=☞ClassOrganization⇒[]
            strm append: s; space]
        strm append: '"'].
        [environment@nil⇒[]
        for§ s from: environment do§
            [strm semicrtab; append: 'sharing: '+ (Smalltalk invert: s)]]]
!
```

"ClassOrganizer"

```
Class new title: 'ClassOrganizer';
   fields: 'globalComment commentVector groupVector';
   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

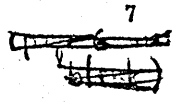
```
init: sortedVec
   [globalComment ← 'This class has not yet been commented'.
   commentVector ← 'As yet unclassified' inVector.
   groupVector ← sortedVec inVector]
```

Access to parts

```
categories [⇑commentVector]
category: str | i
   [i ← commentVector find: str.
   i=0⇒[user notify: 'No such category: '+str]
   ⇑groupVector•i]
classify: selector under: heading | s
   [selector is: Vector⇒
         [for: s from: selector do:
            [self classify: s under: heading]]
   [heading='As yet unclassified'⇒
      [selector=☞doit⇒[⇑self]; =☞debugEval:⇒[⇑self]
      self invert: selector⇒[⇑self]]]].
   s ← commentVector find: heading.
   s=0⇒[commentVector ← commentVector , heading.
         groupVector ← groupVector , selector inVector]
   groupVector•s has: selector⇒[]
   groupVector•s ← groupVector•s insertNonDescending: selector]
delete: selector | i          "delete this from all categories"
   [for: i to: groupVector length do:
      [groupVector•i has: selector⇒
         [groupVector•i ← (groupVector•i) delete: selector]
      ]]
globalComment [⇑globalComment]
globalComment ← globalComment [⇑self]
invert: selector | i
   [for: i to: groupVector length do:
      [groupVector•i has: selector⇒[⇑commentVector•i]]
   ⇑false]
```

Conversion to text

```
asParagraph | s i
   [s ← Stream default.
   s print: globalComment.
   for: i to: commentVector length do:
      [s cr; print: ((commentVector•i) inVector concat: groupVector•i)]
   ⇑s contents asParagraph]
fromParagraph: para | t i j g
   [user displayoffwhile:
   [t ← para asVector.
   globalComment ← t•1.
   commentVector ← Vector new: t length-1.
   groupVector ← Vector new: t length-1.
   for: i to: t length-1 do:
      [g ← t•(i+1).
      commentVector•i ← g•1.
      until: 0=(j← g find: ☞←) do:    "reconstitute ← suffixes"
         [g ← g replace: j-1 to: j by: (g•(j-1)+'←') unique inVector]
      groupVector•i ← (g copy: 2 to: g length) sort]
   ]]
!
```



"Context"

Class new title: 'Context';
fields: 'sender "<Context> from which this message was sent"
receiver "<Object> to which this message was sent"
mclass "<Class> in which a method was found"
method "<String>, the encoded method"
tempframe "<Vector> to hold temporaries and a stack"
pc "<Integer> marks progress of execution in method"
stackptr "<Integer> offset of stack top in tempframe"';
asFollows!

This class has not yet been commented

As yet unclassified

```
cleancopy | i
    [⇑Context new
        sender: sender
        receiver: receiver
        mclass: mclass
        method: method
        tempframe: tempframe copy
        pc: pc
        stackptr: stackptr]
debug | t
    [self print.
    while☺ [user cr. t ← user request: '*'] do☺
        [t ← [t@nil⇒[self doit] self execute: t].
        ☞debugret=t⇒[self print] t print]
    ⇑☞debugret]
eval [user croak] primitive: 30
execute: code | vars var strm
    [
    code length=0⇒ [⇑nil]
    (code˙1)=('#'˙1)⇒ [⇑self's(code˙(2 to: code length))]
    vars ← self variables.
    strm ← Stream default.
    for☺ var from: vars do☺
        [strm append: var; space].
    receiver class understands: 'debugEval: debugVars | debugRipt debugAns ' + strm contents +
        '
        [
        for☺ debugRipt to: debugVars length do☺
            [(thisContext tempframe)˙(debugRipt+3) ← debugVars˙debugRipt.].
        debugAns ← [' + code + '].
        for☺ debugRipt to: debugVars length do☺
            [debugVars˙debugRipt ← (thisContext tempframe)˙(debugRipt+3).].
        ⇑debugAns
]'.
    ⇑receiver debugEval: tempframe˙(1 to: vars length).]
litof: a
    [⇑(method word: a+4) asObject]
mclass [⇑mclass]
printon: strm
    [strm append: mclass title. sender@nil⇒ []
    [receiver is: mclass⇒[] strm append: '('+receiver class title+')'].
    strm append: '↠'; print: sender thisop]
receiver
    ["Return my receiver."
    ⇑receiver]
release        "release frames to break cycles"
    [tempframe ← nil.
    sender@nil⇒[] sender release]
remoteCopy
    [⇑Context new sender: sender receiver: receiver mclass: mclass
        method: method tempframe: tempframe pc: pc+2 stackptr: stackptr]
sender [⇑sender]
```

```
sender: sender receiver: receiver mclass: mclass
        method: method tempframe: tempframe pc: pc stackptr: stackptr
sender← sender []
specialops
    [⇑☞(+ - < > ≤ ≥ = ≠
        * / \ | min: max: land: lor:
        ' →('←' atomize) next →('next←' atomize) length @ nil nil
        class and: or: new new: to: oneToMeAsStream asStream)]
stack | a strm
    ["Return a Vector of me and all my derivative contexts."
    strm ← (Vector new: 20) asStream.
    strm next ← a ← self.
    until: (a←a sender)@nil do: [strm next ← a].
    ⇑strm contents.]
tempframe
    ["Return my tempframe."
    ⇑tempframe]
thisop | a
    [a ← method'pc.
    a≥0320⇒ [⇑self litof: a-0320]
    a≥0260⇒ [⇑self specialops'(1+a-0260)]
    ⇑☞something]
trace | strm a
    [strm ← Stream default. self printon: strm.
    a ← sender. until: a@nil do:
        [strm cr. a printon: strm. a ← a sender]
    ⇑strm contents]
variables | sourceCode vars selector var
    ["Return a Vector of all my method variable names."
    "This should be done by the compiler"
    sourceCode ← mclass code: sender thisop.
    sourceCode ← (sourceCode'(1 to: (sourceCode find: ('['' 1))-1)) asStream asVector asStream.
    vars ← (Vector new: 10) asStream.
    while: [(selector ← sourceCode next) and: selector≠☞|] do:
        [
        var ← sourceCode next.
        [var=☞←⇒ [var ← sourceCode next]].
        var⇒
            [
var=☞|⇒ [sourceCode skip: ¯1].
            vars next ← var.
]
        ].
    while: [var ← sourceCode next] do: [vars next ← var.].
    ⇑vars contents]
!
```

"Object"

```
Class new title: 'Object';
   subclassof: nil;
   fields: '';
   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 [user croak] primitive: 4
≠ x [⇑self=x@false]
≥ x [⇑self<x@false]
= x [⇑self@x]
and° x [self⇒[⇑x eval] ⇑false]
and: x [self⇒[⇑x] ⇑false]
eqv: x [x⇒[⇑self] ⇑self@false]
is: x [⇑self class@x]
isnt: x [⇑(self class@x) @ false]
or° x [self⇒[⇑true] ⇑x eval]
or: x [self⇒[⇑true] ⇑x]
xor: x [x⇒[⇑self@false] ⇑self]
```

Construction

```
, x | v
   [v ← Vector new: 2.
   v˙1 ← self. v˙2 ← x. ⇑v]
asStream [⇑self asVector asStream]
asVector | v
   [self@nil⇒[⇑Vector new: 0]
   v ← Vector new: 1. v˙1 ← self. ⇑v]
copy      "create new copy of self"
   [self is: Object⇒[⇑self]
   ⇑self class copy: self]
inVector | vec
   ["Return me as the sole element of a new Vector."
   vec ← Vector new: 1.
   vec˙1 ← self.
   ⇑vec]
recopy   "recursively copy whole structure"
   [⇑self class recopy: self]
```

Aspects

```
's code
   [self class understands: 'doit [⇑[' + code + ']]'.
   ⇑self doit]
asOop [user croak] primitive: 46
canunderstand: selector
   [⇑self class canunderstand: selector]
class [user croak] primitive: 27
fields
   ["Return an Array of all my field names or many of my subscripts."
   self class is: VariableLengthClass⇒
      [
      self length < 30⇒ [⇑1 to: self length]
      ⇑ (1 to: 10) concat: (self length-10 to: self length)
      ]
   ⇑self class instvars]
hash [user croak] primitive: 46
inspect | instanceVarPane instanceValuePane panedWindow safeVec
   [instanceVarPane ← VariablePane init. instanceValuePane ← CodePane init.
   panedWindow ← PanedWindow new title: self class title
      with: (instanceVarPane, instanceValuePane) at: InspectPaneFrames.
   panedWindow newframe; show.
```

```
    instanceVarPane to: instanceValuePane. instanceValuePane from: instanceVarPane.
    safeVec ← Vector new: 2. safeVec all ← self.
    instanceVarPane names: (↑(self) concat: self fields) values: safeVec wrt: false.
    user restartup: panedWindow]
inspectfield: n    "used by variable panes"
    [self class is: VariableLengthClass⇒ [↑self˙(self fields˙n)]
    ↑self instfield: n]
instfield: n [user croak] primitive: 38
instfield: n ← val [user croak] primitive: 39
instfields | vec size i
    ["Return an Array of all my field values or many of my elements."
    self class is: VariableLengthClass⇒ [↑self˙self fields]
    size ← self class instsize.
    vec ← Vector new: size.
    for: i to: size do: [vec˙i ← self instfield: i].
    ↑vec]
ref: index
    [↑FieldReference new object: self offset: index]
```

Printing

```
asString | strm
    [strm ← (String new: 8) asStream.
    self printon: strm. ↑strm contents]
fullprint | strm
    [strm ← Stream default. self fullprinton: strm.
    user show: strm contents]
fullprinton: strm
    [self@nil⇒ [strm append: 'nil']
    self@false⇒ [strm append: 'false']
    self@true⇒ [strm append: 'true']
    self class print: self on: strm]
print
    [user show: self asString]
printon: strm
    [self@nil⇒ [strm append: 'nil']
    self@false⇒ [strm append: 'false']
    self@true⇒ [strm append: 'true']
    strm append: 'a ' + self class title]
```

System Primitives

```
error
    [user notify: 'Message not understood: '+thisContext sender thisop]
PTR [] primitive: 46
purge [] primitive: 44
refct [user croak] primitive: 45
startup    "loopless scheduling"
    [self firsttime⇒
        [while: self eachtime do: [].
        ↑self lasttime]
    ↑false]
systemNoHistory [] primitive: 82
```

Cloning

```
specialLocs [] primitive: 84
systemFlush | bacpur i chk a
    [user clearshow: 'Dont forget Top top! in the new system to enable interrupts. '.
    i ← nil specialLocs.
    bacpur ← CoreLocs new base: (i˙11)-3 length: 4.
    014764≠(bacpur˙3)⇒[user notify: 'bad systemFlush constants']
    a ← (bacpur base+2), (bacpur˙2). "bacpur pcl"
    (bacpur˙1 "purge pcl" -1)≠(a˙2 "bacpur pcl")⇒[↑false]
    chk ← nil systemTableWrite: (dp0 file: 'Table.Vmem').
    chk next← a. chk next← (i˙4 +1), ¯1. "lox empty"
    chk reset. a ← (0,0,0,0,0).
    for: i from: chk do: [(i˙2)≠(mem˙(i˙1))⇒[↑false]].
    nil systemVmemOut. user overlay: a]
systemTableWrite: f | m i PMbase a
    [i ← nil specialLocs.
```

```
    PMbase ← CoreLocs new base: (i*2)-3 length: 1024+3.
    0174000≠(PMbase*0)⇒[user notify: 'bad systemFlush constants']
    f reset.
    i ← 1023+3-1. while§ PMbase*i ≠0 do§ [i ← i-2].
    f nextword← (0 field: 151 "PCL" ← (i+2-(3-1))/2). "pmatm"
    a ← (PMbase base+2), (PMbase*2). "pmend"
    for§ i from: (2 to: 1023+3) do§ [f nextword← PMbase*i]. "pmend+PM"
    SpecialOops*1 ← Object md method: ☞error.
    f nextword← SpecialOops asOop. " page 0 oops "
    cur ← Context new sender: nil receiver: user mclass: UserView
        method: (m ← UserView md method: ☞restart)
        tempframe: (Vector new: (m*3 max: m*4)) pc: m*6
        stackptr: m*5 -1.
    f nextword← cur asOop.
    i ← nil systemZipWrite: f. i next← a. ↑i
    ]
systemVmemOut [] primitive: 83
systemZipWrite: f | zip zadd i a
    [i ← nil specialLocs.
    zip ← CoreLocs new base: (i*3)-3 length: 512+3.
    512≠(zip*1)⇒[user notify: 'bad systemFlush constants']
    zadd ← CoreLocs new base: (i*12)-3 length: 4.
    0100≠(zadd*0)⇒[user notify: 'bad systemFlush constants']
    a ← (Vector new: 1) asStream. a next← (zadd base+1), (zadd*1).
    f settopage: 1 char: 2*(1024+4).
    f nextword← zip*0 -(i*3). " zfree "
    f nextword← zadd*1. " zfused "
    for§ i from: (3 to: 511+3) do§ [f nextword← zip*i].
    f close. ↑a
    ]
!
```

"UserView"

```
Class new title: 'UserView';
  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"
    screenfile "<File> for saving current bitmap"
    disp "<dispframe> default message stream"
    sched "<PriorityScheduler> tasks for this view"';
  asFollows
```

This is a melting-pot, incorporating the notions of user interaction and display context

Terminal

```
anybug [↑self buttons>0]
anykeys [↑self keyset>0]
bluebug [↑self buttons=2]
buttons
   [↑7-(mem'0177030 land: 7)]
kbck | t
   [t ← self rawkbck→[↑kbMap't] self purgealittle. ↑false ]
kbd [until° self rawkbck do° [self purgealittle]
   ↑kbMap'self rawkbd]
kbd: char [user croak] primitive: 100    "actually stuff char into the keyboard queue"
keyset
   ["Fix a bug in sysdefs."
   ↑037 - ((mem'0177030 lshift: ~3) land: 037)]
mp
   [↑Point new x: mem'0424-htab y: mem'0425-vtab]
nobug
   [↑self buttons = 0]
rawkbck [] primitive: 64
rawkbd [] primitive: 63
redbug [↑self buttons=4]
waitbug
   [until° self anybug do° [] ↑self mp]
waitnobug
   [while° self anybug do° [] ↑self mp]
waitnokey [until° self keyset=0 do° [self rawkbck]]
yellowbug [↑self buttons=1]
```

Dialog Window

```
clearshow: str
   [disp clear; append: str; show]
cr [disp cr]
croak
   [self notify: 'A primitive has failed.']
ev
   [disp ev]
read [↑disp read]
request: s[↑disp request: s]
show [disp show]
show: str
   [disp append: str; show]
space
   [disp space]
```

Display

```
cursorloc ← pt
   [mem'0424 ← pt x+htab. mem'0425 ← pt y+vtab]
displayoffwhile° expr | t v
   [t ← mem'067. mem'067 ← 56.
   v ← expr eval. mem'067 ← t. ↑v]
```

```
dowithdisplayoff expr | v
    [mem˙067 ← 30.  v ← expr eval.
    self restoredisplay. ⇑v]
loadcursorbitsfrom: bitstr|i
[for i to: 16 do
        [mem˙(0430+i) ← bitstr word: i]]
reconfigure [] primitive: 62
restoredisplay
    [mem˙067 ← screenrect height/2]
screenextent: extent tab: tab
    [mem˙065 ← (0400*(tab x/16))+(extent x/16|2).
    mem˙067 ← extent y/2. mem˙063 ← tab y/2.
    htab ← tab x|16. vtab ← tab y|2.
    screenrect ← 0@0 rect: (extent x|32)@(extent y|2).
    self reconfigure]
screenrect [⇑screenrect]
```

Display Overlays (unfinished)

```
close
    [screenfile open.   "write out the old bitmap"
    screenfile write: screenrect height*(screenrect width+15/16)/256
        from:   mem˙066.
    screenfile close]
install
    [user close.  self reconfigure.  self open.  user ← self]
open
    [screenfile open.   "read in the new bitmap"
    screenfile read: screenrect height*(screenrect width+15/16)/256
        to:   mem˙066.
    screenfile close]
```

System quit/restart

```
InLd: fileid
    [user notify: 'file problem'] primitive: 86
OutLd [] primitive: 85
overlay: fileid
    [dp0 close. self OutLd⇒[self InLd: fileid]
    while user keyset>0 do [user show: 'The keyset is stuck'; cr]]
quit
    [self overlay: 0,0,0,0,0]
restart
    [NormalCursor topage1.
    self restart [user run]]
restart code
    [thisContext sender release.
    code sender ← nil. thisContext sender ← nil.      "release caller chain"
    CodeKeeper ← (Vector new: 10) asStream.    "release held code"
    disp cr; append: ' ** restart'; show.
    while true do [code eval]]
Swat [] primitive: 90
```

Window Scheduling

```
browse      "Let the user draw a five-paned window to browse through classes."
        | panedWindow systemPane classPane orgPane selectorPane codePane
    [
    "Create the panes."
    systemPane ← SystemPane init. classPane ← ClassPane init.
    orgPane ← OrganizationPane init. selectorPane ← SelectorPane init.
    codePane ← CodePane init.
    "Create the paned window."
    panedWindow ← PanedWindow new
        title: 'Classes'
        with: (systemPane, classPane, orgPane, selectorPane, codePane)
        at: BrowsePaneFrames.
    panedWindow newframe; show.
    "Interconnect the panes."
    systemPane to: classPane. classPane from: systemPane to: orgPane.
    orgPane from: classPane to: selectorPane. selectorPane from: orgPane to: codePane.
```

```
    codePane from: selectorPane.
    "Display the panes."
    systemPane update.
    "Schedule and activate the three-paned window."
    thisContext sender release. self restartup: panedWindow]

notify: errorString
        | panedWindow stackPane
    ["Restore the full display.  Schedule a one-paned window to notify the user that errorString
happened."
    self restoredisplay.
    NotifyFlag@false⇒
        [disp append: errorString; cr;
        append: 'ctrl-d to proceed, else #user restart or #sender debug'; cr; show.
        thisContext sender debug]
    NotifyFlag ← false. stackPane ← StackPane init.
    panedWindow ← NotifyWindow new title: errorString with: stackPane inVector at: SinglePaneFrames.
    panedWindow frame: (panedWindow fixframe: NotifyFrame); show.
    stackPane from: panedWindow context: false instance: false code: false.
    stackPane of: thisContext sender inVector.
    NotifyFlag ← true. self restartup: panedWindow]
restartup: window
    [ "Equivalent to schedule window, restart, and redbug in window, except firsttime is already
done."
    NormalCursor topage1.
    thisContext sender ← nil.
    CodeKeeper ← (Vector new: 10) asStream.
    self schedule: window.
    window takeCursor.
    while⁰ window eachtime do⁰ [].
    window lasttime.
    while⁰ true do⁰ [self run].]
restore | w
    [screenrect clear.
    for⁰ w from: (sched length to: 1 by: ¯1) do⁰
        [(sched•w) show]]
run | n t [while⁰ true do⁰
    [n ← 0.
    until⁰ (n← n+1)>sched length do⁰
        [(t← sched•n) startup⇒
            [sched promote: t. n ← 0]]]]
sched [⇑sched]
schedule: window
    [sched@nil⇒[sched ← window asVector]
    sched ← window asVector concat: sched]
unschedule: window | t
    [0<(t← sched find: window)⇒
        [sched ← sched•(1 to: t-1) concat: sched•(t+1 to: sched length)]]
```

Misc System Stuff

```
oopsToFile | a c i t s cs f
    [f ← dp0 file: 'oops'.
    cs ← (Vector new: 10) asStream.
    for⁰ a from: Smalltalk do⁰
        [c ← Smalltalk•a.
        (c is: Class) or: (c is: VariableLengthClass)⇒[cs next ← c]]
    cs ← cs contents. t ← cs copy.
    for⁰ i to: t length do⁰ [t•i ← (t•i) asOop]
    for⁰ i from: t sorter do⁰
        [f append: (t•i) base8; tab; append: (cs•i) title; cr]
    for⁰ c from: cs•t sorter do⁰
        [f cr; append: c title; cr.
        s ← c md contents. "selectors"
        t ← s copy.
        for⁰ i to: t length do⁰
            [t•i ← (c md method: t•i) asOop]  "method oops"
        for⁰ i from: t sorter do⁰
            [f tab; append: (t•i) base8; tab; append: s•i; cr]
        user show: c title; cr]
    f close]
```

```
purgealittle [] primitive: 89
test | a c s f l v i   "user test"
   ["f ← dp0 file: 'classes.list'."
   for: a from: Smalltalk contents sort do:
      [c ← Smalltalk˙a.
      (c is: Class) or: (c is: VariableLengthClass)⇒
         [user show: c title; cr.
         user displayoffwhile:
            [for: s from: c md do:
               [(c md method: s)˙2=41⇒[c understands: (c code: s)]]
            ]
         ]
      "f shorten; close"]
   ]

time: expr | t
   [t ← mem˙280. expr eval. ↑mem˙280-t]
Version [↑'3.5 September 6']
   "
   growing enabled again.
   mouse scrolling inverted
      goes up from bottom, down from top
      in text, adjacent line will jump to middle
   proceed now works in debugger (when NotifyFlag is true)
   # now works in debugger (when NotifyFlag is false)
   - - -
   Keyset cursor bug fixed
   Class.filoutOrganization fixed
   Undeclared cleaned out
      (Array.replace, Dispframe.eachtime and UserView.unschedule
      were all sharing the undeclared variable t!)
   - - -
   Scroll bars controlled by mouse (keyset still works)
      redbug does it, dist from center sets speed
   Several debugger bugs fixed.
   Faster cursor show
   Filin prints selector names again
   Group filout puts headings at page tops as originally intended
   "
```

As yet unclassified

```
classNames     "an alphabetized Vector of all Smalltalk class titles uniqued"
      | classes x c
   [
   AllClassNames@nil⇒
      [
      classes ← (Vector new: 20) asStream.
      for: x from: Smalltalk do:
         [
         c ← Smalltalk ˙ x.
         (c is: Class) or: (c is: VariableLengthClass)⇒ [classes next ← x].
         ].
      AllClassNames ← classes contents sort.
      ].
   ↑AllClassNames
   ]
!
```

"VariableLengthClass"

```
Class new title: 'VariableLengthClass';
   subclassof: Class;
   fields: '';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
copy: inst | t i
   [t ← self new: inst length.
   for i to: inst length do
      [t'i ← inst'i]
   ↑t]
new
   [user notify: 'use new: <Integer=length> here.']
new: length
   [length ≥ 020000 ⇒[length print. user notify: ' is too large']
   length < 0 ⇒[length print. user notify: ' is too small']
   ↑self new: length asInteger] primitive: 29
recopy: inst | t i
   [t ← self new: inst length.
   for i to: inst length do
      [t'i ← (inst'i) recopy]
   ↑t]
!
```

"numbers"
"Version 3.5 6 Sept 77"

"Float"

```
Class new title: 'Float';
   subclassof: Number;
   fields: '';
   declare: 'pi halfpi twopi ';
   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. The printing routine should be cleaned up someday (volunteers?). Here are some valid floating-point examples:
8.0 13.3 0.3 2.5e6 1.27e-300 -12.987654e2412
Mainly: use shift-minus, no imbedded blanks, little e for tens power, and a digit on both sides of the decimal point.

Arithmetic

```
≤ arg
   [↑self≤arg asFloat] primitive: 73
≠ arg
   [↑self≠arg asFloat] primitive: 76
≥ arg
   [↑self≥arg asFloat] primitive: 75
* arg
   [↑self*arg asFloat] primitive: 69
+ arg
   [↑self+arg asFloat] primitive: 67
- arg
   [↑self-arg asFloat] primitive: 68
/ arg
   [0.0=arg⇒[user notify: 'Attempt to divide by 0.0']
   ↑self/arg asFloat] primitive: 70
< arg
   [↑self<arg asFloat] primitive: 71
= arg
   [↑self=arg asFloat] primitive: 72
> arg
   [↑self>arg asFloat] primitive: 74
```

Conversion

```
asFloat
asInteger       "Return an Integer = self ipart"
   [user notify: 'Value out of range'] primitive: 78
fpart [user croak] primitive: 77
ipart     "Returns a Float with zero fractional part"
   [↑self-self fpart]
recopy [↑self]
```

Math functions

```
cos [↑(self+halfpi) sin]
degreesAsRadians [↑self/90.0*halfpi]
init
   [pi ← 3.1415926536.
   halfpi ← pi/2.0.
   twopi ← 2.0*pi]
ipow: x       "fixed powers in log n steps"
   [x=0⇒ [↑1.0]
   x=1⇒ [↑self]
   x>1⇒ [↑((self*self) ipow: x/2)*(self ipow: x\2)]
   ↑1.0/(self ipow: 0-x)]
sin | x x2 sum const sign  "for angles in radians"
   [sign ← 1.0.      "Normalize to 0≤x≤pi/2"
   x ← (self/twopi) fpart*twopi.
      [x<0.0⇒[x ← x+twopi]].
      [x>pi⇒[x ← x-pi.  sign ← -1.0]].
      [x>halfpi⇒[x ← pi-x]].
   sum ← x.
   x2 ← x*x.
```

```
    for§ const from:    "Now compute the series"
        ☞(¯0.166666666 0.0083333315 ¯1.98409e¯4 2.7526e¯6 ¯2.39e¯8)
        do§ [sum ← const*(x ← x*x2)+sum]
    ⇑sign*sum]
sqrt | guess i
    [self≤0.0⇒[self=0.0⇒[⇑0.0] user notify: 'sqrt invalid for x<0.']
    guess ← self+0.0.    "copy x"
    guess instfield: 1 ← (guess instfield: 1)/4*2."and halve expt for first guess"
    for§ i to: 5 do§
        [guess ← (self-(guess*guess)) / (guess*2.0) + guess]
    ⇑guess]
```

Printing

```
absprinton: strm | x y q i fuzz
    [fuzz ← 1.0e¯9.   x ← fuzz+1*self.        "round up"
    y ← [x<1⇒ [0-(10.0/x epart: 10.0)] self epart: 10.0].
    x ← x/(10.0 ipow: y).          "normalize x, y←exponent"
    [x≥10.0⇒[y← y+1. x← x/10.0]]. "shouldnt be necessary"
    fuzz ← fuzz*x*0.2.       "now fuzz tracks significance"
    [y<6 and: y>¯4⇒
        [q ← 0.            "decimal notation"
        y<0⇒ [strm append: '0.0000'•(1 to: 1-y)]].
    q ← y. y ← 0].         "scientific notation"
    until§ ((y←y-1)<¯2 and: x<(fuzz←fuzz*10.0)) do§
        [strm next ← 060+x ipart.
        x ← 10.0 * x fpart.
        y=¯1⇒ [strm append: '.']]
    q≠0⇒[strm append: 'e'; print: q]]
epart: base | x "gives floor log.base self"
    [self<base⇒ [⇑0]   "self assumed positive"
    self<(base*base)⇒ [⇑1]
    x ← 2*(self epart: base*base)."binary recursion like ipow"
    ⇑x + ((self/(base ipow: x)) epart: base)]
printon: strm
    [self>0.0⇒[self absprinton: strm]
    self=0.0⇒[strm append: '0.0']
    strm append: '¯'. (0.0-self) absprinton: strm]
!
```

"Integer"

```
Class new title: 'Integer';
   subclassof: Number;
   fields: '';
   sharing: BitMasks;
   asFollows!
```

Integers are 16-bit numbers, stored in two-s complement form. The allowable range is from ¯32768 to +32767. You can type them in octal by typing a leading zero, as in 0377.

Arithmetic

```
≤ arg
   [↑self ≤ arg asInteger] primitive: 11
≠ arg
   [↑self ≠ arg asInteger] primitive: 12
≥ arg
   [↑self ≥ arg asInteger] primitive: 13
* arg
   [↑self * arg asInteger] primitive: 21
+ arg
   [↑self + arg asInteger] primitive: 6
- arg
   [↑self - arg asInteger] primitive: 7
/ arg
   [0=arg⇒[user notify: 'Attempt to divide by 0']
   ↑self / arg asInteger] primitive: 22
< arg
   [↑self < arg asInteger] primitive: 8
= arg
   [↑self = arg asInteger] primitive: 9
> arg
   [↑self > arg asInteger] primitive: 10
\ arg "mod"
   [↑self \ arg asInteger] primitive: 26
| arg  "truncate"
   [↑self/arg*arg]
```

Bit Manipulation

```
allmask: b [↑b = (self land: b)]
anymask: b [↑0 ≠ (self land: b)]
field: fld
   [↑self field: fld asInteger] primitive: 36
field: fld ← val
   [↑self field: fld asInteger ← val asInteger] primitive: 37
hibit | i
   [for: i from: (16 to: 1 by: ¯1) do:
      [0=(self land: (biton:i))⇒[] ↑i] ↑ 0]
land: arg
   [↑self land: arg asInteger] primitive: 23
lor: arg
   [↑self lor: arg asInteger] primitive: 24
lshift: arg
   [↑self lshift: arg asInteger] primitive: 25
nomask: b [↑0 = (self land: b)]
xor: arg
   [↑self xor: arg asInteger] primitive: 35
```

Conversion

```
asFloat [user croak] primitive: 34
asInteger [↑self]
asLI
      | d i
   [self<0⇒[↑(0-self) asLI negated]
   d ← self asString reverse copy.
   for: i to: d length do: [d:i ← d:i-060]
   ↑LongInteger new digits: d neg: false]
```

```
asObject [user croak] primitive: 81
oneToMeAsStream        "used by for-loops"
    [⇑Stream new of: (Interval new from: 1 to: self by: 1)]
```

Subscripts

```
subscripts: a
    [user notify: 'Subscript out of bounds: ' + self asString]
subscripts: a ← val
    [self≥1 and: self≤a length⇒[user notify: 'Improper store (non-char into String?)']
    user notify: 'Subscript out of bounds: ' + self asString]
```

Printing

```
absprinton: strm | rem
    [rem ← self\10.
    [self>9⇒ [self/10 absprinton: strm]].
    strm next ← rem+060]
base8 | s
    [s ← Stream default. s append: '0'.
    self printon: s base: 8. ⇑s contents]
base: b | s
    [s ← Stream default.
    self printon: s base: b. ⇑s contents]
printon: strm
    [self<0⇒[strm append:'-'. (0-self) printon: strm base: 10]
    self printon: strm base: 10]
printon: strm base: b | rem
    [self<0⇒[rem ← 040000\b*2+self-0100000\b. "get it?"
            (040000/b*2+(self-0100000/b)) printon: strm base: b.
            strm next ← 060+rem]
    rem ← self\b.
    [self≥b⇒[self/b printon: strm base: b]].
    strm next ← 060+rem]
```

Characters

```
asLowercase
    [0101 ≤ self⇒ [
        self ≤ 0132⇒ [⇑self + 040]]]
asUppercase
    [0141 ≤ self⇒ [
        self ≤ 0172⇒ [⇑self - 040]]]
compareChar: c | a
    ["⇑self asLowercase compare: c asLowercase"
    a← self.    "written in-line for speed"
    [0101≤a⇒[a≤0132⇒[a←a+040]]].
    [0101≤c⇒[c≤0132⇒[c←c+040]]].
    a<c⇒[⇑1] a=c⇒[⇑2] ⇑3]
isalphanumeric
    [self isletter⇒[⇑true] "lower-case"
    ⇑self isdigit]
isdigit
    [self ≥ 060⇒ " 0 "
        [⇑self ≤ 071]   " 9 "
    ⇑false]
isletter
    [self ≥ 0141⇒   " a "
        [⇑self ≤ 0172]  " z "
    self ≥ 0101⇒ " A "
        [⇑self ≤ 0132]  " Z "
    ⇑false]
tokenish        "test for token-chars"
    [self isletter⇒[⇑true] "lower-case"
     self isdigit⇒[⇑true]   "digits"
     ⇑'-.:$' has: self]
```

Copying and Purging

```
copy [⇑self]
purge [user croak] primitive: 44
recopy [⇑self]
```

"LongInteger"

Class new title: 'LongInteger';
subclassof: Number;
fields: 'digits neg';
asFollows!

Long integers are integers of arbirary size. The digits field is a string that holds the integer in base 10. Division has not yet been implemented. You can create them by sending the message asLI to an Integer or a String.

Access to parts

```
digits [⇑digits]
digits: digits neg: neg [⇑self]
neg [⇑neg]
```

Arithmetic

```
* arg
        | prod prodin prodout mcand i x carry plier p
    [ [arg is: LongInteger⇒[] arg←arg asLI].
    prod ← String new: digits length+arg digits length.
    prodin ← prod asStream. prodout ← prod asStream.
    prod all← 0. mcand ← arg digits.
    for: i to: digits length do:
        [prodin reset; skip: i-1. prodout reset; skip: i-1.
        plier ← digits˙i. carry ← 0.
        for: x from: mcand do:
            [p ← x*plier+carry+prodin next.
            carry ← p/10. prodout next← p-(carry*10)]
        until: carry=0 do:
            [p ← carry+prodin next.
            carry ← p/10. prodout next← p-(carry*10)]]
    [prod last=0⇒[prod ← prod˙(1 to: prod length-1) copy]].
    ⇑LongInteger new digits: prod neg: neg≠arg neg]
+ arg
        | shorter longer sl s sum i carry
    [ [arg is: LongInteger⇒[] arg←arg asLI].
    neg⇒[arg neg⇒[⇑(self negated + arg negated) negated]
        ⇑arg - self negated]
    arg neg⇒[⇑self - arg negated]
    longer ← digits. shorter ← arg digits.
    [shorter length>longer length⇒[longer ← shorter. shorter ← digits]].
    sum ← (String new: longer length) asStream.
    sl ← shorter length. carry ← 0.
    for: i to: longer length do:
        [s ← longer˙i+carry+[i>sl⇒[0] shorter˙i].
        carry ← [s>9⇒[s←s-10. 1] 0].
        sum next← s]
    [carry>0⇒[sum next← 1]].
    ⇑LongInteger new digits: sum contents neg: false]
- arg
        | shorter longer sl s sum i borrow
    [ [arg is: LongInteger⇒[] arg←arg asLI].
    neg⇒[arg neg⇒[⇑arg negated - self negated]
        ⇑(arg + self negated) negated]
    arg neg⇒[⇑self + arg negated]
    arg>self⇒[⇑(arg - self) negated]
    longer ← digits. shorter ← arg digits.
    sum ← (String new: longer length) asStream.
    sl ← shorter length. borrow ← 0.
    for: i to: longer length do:
        [s ← longer˙i+borrow-[i>sl⇒[0] shorter˙i].
        borrow ← [s<0⇒[s←s+10. ¯1] 0].
        sum next← s]
    while: sum last=0 do: "trim leading zeroes"
        [sum skip: ¯1.
        sum empty⇒[⇑0 asLI]]
    ⇑LongInteger new digits: sum contents neg: false]
```

```
/ arg []
< arg [⇑(self compare: arg)=1]
= arg [⇑(self compare: arg)=2]
> arg [⇑(self compare: arg)=3]
compare: arg | i a
    [ [arg is: LongInteger⇒[] arg←arg asLI].
    neg⇒
        [arg neg⇒[⇑arg greatermag: self] ⇑1]
    arg neg⇒[⇑3]
    a ← arg digits.
    digits length>a length⇒[⇑3]; <a length⇒[⇑1]
    for: i from: (digits length to: 1 by: ¯1) do:
        [(digits•i)>(a•i)⇒[⇑3];
                <(a•i)⇒[⇑1]]
    ⇑2]
negated
    [⇑LongInteger new digits: digits neg: neg@false]
```

Printing

```
printon: strm | i
    [ [neg⇒[strm append: '¯']].
    for: i from: (digits length to: 1 by: ¯1) do:
        [strm next← digits•i+060]]
!
```

"Number"

```
Class new title: 'Number';
   fields: '';
   asFollows!
```

Numbers in general

Arithmetic

```
abs [self<0⇒[↑self * ¯1]]
compare: i
   [self < i⇒ [↑1]
   self = i⇒ [↑2]
   ↑3]
factorial  "I only work for positive integer values"
   [self=0⇒[↑1]
   ↑self * (self-1) factorial]
max: arg
   [self<arg⇒[↑arg]]
min: arg
   [self>arg⇒[↑arg]]
sign
   [↑[self=0⇒ [0]; <0⇒ [¯1] 1]]
```

Conversions

```
asPoint
   ["Return a Point with me as both coordinates."
   ↑self @ self]
asPtX "pretend to be a Point for Point +-*/"
asPtY "pretend to be a Point for Point +-*/"
asRectangle
   ["Return a Rectangle with me as all coordinates."
   ↑self @ self rect: self @ self]
asRectCorner "pretend to be a Rectangle for Rectangle +-*/"
asRectOrigin "pretend to be a Rectangle for Rectangle +-*/"
```

Subscripts

```
subscripts: a
   [↑a'self asInteger]
subscripts: a ← val
   [↑a'self asInteger ← val]
```

Intervals, Points

```
@ y
   [↑Point new x: self y: y]
to: x
   [↑Interval new from: self to: x by: 1]
to: x by: y
   [↑Interval new from: self to: x by: y]
!
```

"basic data structures"
"Version 3.5 6 Sept 77"

"Array"

Class new title: 'Array';
fields: '';
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 a'(1 to: 3).

Reading and Writing

```
' x
    [↑x subscripts: self]
' x ← val
    [↑x subscripts: self ← val].
= arg | x
    [self length ≠ arg length⇒ [↑false]
    for: x to: self length do:
        [(self'x) = (arg'x)⇒ [] ↑false]
    ↑true]
all ← val | i
    [for: i to: self length do:
        [self'i ← val]]
last
    [↑self'self length]
last ← val
    [↑self'self length ← val]
length  [user notify: 'message not understood.']
```

Copying and Altering

```
+ arg [↑self concat: arg]
concat: arg | x
    [x ← self species new: self length + arg length.
    self copyto: x.
    arg copyto: (Stream new of: x from: self length+1 to: x length).
    ↑x]
copy
    [↑self copyto: (self species new: self length)]
copy: a to: b [↑(self'(a to: b)) copy]
copyto: t | i s me
    [s ← t asStream.
    me ← self asStream.
    for: i to: self length do:      "general code wont stop at false"
        [s next ← me next]
    ↑t]
delete: obj | s each
    [s ← (self species new: self length) asStream.
    for: each from: self do:
        [obj=each⇒[] s next← each]
    ↑ s contents]
grow
    [↑self copyto: (self species new: (4 max: self length*2))]
growby: n
    [↑self copyto: (self species new: self length+n)]
replace: a to: b by: s | x xs
    [x ← self species new: self length+s length -(1+b-a).
    xs ← x asStream.
    self'(1 to: a-1) copyto: xs.
    s copyto: xs.
    self'(b+1 to: self length) copyto: xs.
    ↑x]
swap: i with: j | t
    [t ← self'i. self'i ← self'j. self'j ← t]
```

Sorting and Searching

```
find: x | i
    [for: i to: self length do:
```

```
        [self·i=x⇒ [⇑i]].
    ⇑0]
findnon: x | i
    [for: i to: self length do:
        [self·i≠x⇒ [⇑i]].
    ⇑0]
has: x
    [⇑0≠(self find: x)]
insertNonDescending: x    | a c lo mid hi "self is assumed to be sorted"
    [
    c ← (a ← self species new: (hi ← self length + (lo ← 1))) asStream.
    while: lo < hi do: [self·(mid←lo+hi/2) > x⇒ [hi ← mid] lo ← mid+1].
    self·(1 to: hi-1) copyto: c. c next ← x. self·(hi to: self length) copyto: c.
    ⇑a
    ]
insertSorted: x    | a c lo mid hi    "self is assumed to be sorted"
    [c ← (a ← self species new: (hi ← self length + (lo ← 1))) asStream.
    while: lo < hi do:    "binary search"
        [self·(mid←lo+hi/2) > x⇒[hi ← mid]  lo ← mid+1].
    self·(1 to: hi-1) copyto: c. c next ← x. self·(hi to: self length) copyto: c.
    ⇑a]
permutationToSort
    ["Return a Vector, permutation, such that self·permutation is sorted nondescending. Do not
alter self."
    ⇑((self·((1 to: self length) copy)) sort: 1 to: self length) map.]
promote: t | n
    [n ← self find: t. n=0⇒ []
    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
    ["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."
    self sort: 1 to: self length.]
sort: i to: j | di dij dj tt ij k l n
    ["Sort elements i through j of self to be nondescending."

    "The prefix d means the data at."
    (n←j+1-i)≤1⇒ ["Nothing to sort."]
    "Sort di,dj."
    di ← self·i. dj ← self·j.
    [di>dj⇒ [self swap: i with: j. tt←di. di←dj. dj←tt]].
    n=2⇒ ["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·ij.
    [di>dij⇒ [self swap: i with: ij. dij←di] dj<dij⇒ [self swap: j with: ij. dij←dj]].
    n=3⇒ ["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.
    while:
        [
        while: self·(l←l-1) > dij do: [].
        while: self·(k←k+1) < dij do: [].
        k≤l
        ]
    do:
        [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.]
```

Mapping

```
asStream
    [⇑Stream new of: self]
subscripts: x          "subarrays"
    [⇑Substring new data: x map: self]
```

```
subscripts: x ← val        "subrange replacement"
    [self length≠val length⇒
        [user notify: 'lengths not commensurate']
    val copyto: (Substring new data: x map: self).
    ⇑val]
```

Compatibility

```
isIntervalBy1
    [⇑false]
species
    [⇑Vector]
!
```

"Interval"

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

This class has not yet been commented

As yet unclassified

```
• x
   [x is: Integer⇒[x<1⇒ [⇑nil]
      x>length⇒ [⇑nil]
      ⇑start+(step*(x-1))]
   ⇑super•x]
• x ← val
   [user notify: 'Intervals are not for writing into']
from: start to: stop by: step
   [length ← 1+(stop-start/step)]
isIntervalBy1
   [⇑step=1]
length [⇑length]
start [⇑start]
stop [⇑stop]
!
```

"Stream"

Class new title: 'Stream'; fields: 'array position limit'; asFollows

This class has not yet been commented

As yet unclassified

```
x
   [↑array.x]
x ← val
   [↑array.x ← val]
x | y
   [y← self next⇒      "peek for matching element"
      [x=y⇒ [↑y]       "gobble it if found"
      position ← position-1. ↑false]
   ↑false]
append: x | i    "Array arg"
   [for: i from: x do:
      [self next ← i].
   ↑x]
asStream
asVector
   [↑(Reader new of: self) read]
contents
   [↑(array.(1 to: position)) copy]
cr
   [self next ← 015]
ctlz: chars [self next← 032; append: chars; next← 015]
default
   [self of: (String new: 8)]
dequeue      "use it as a FIFO"
   [↑self dequeue: 1]
dequeue: n | t
   [position<n⇒ [↑false]
   t ← (array.(1 to: n)) copy.
   array.(1 to: position-n) ← array.(n+1 to: position).
   position ← position-n. ↑t]
empty        "for".
   [↑position=0]
end
   [↑position≥limit]
into: x | i        "Array result"
   [for: i to: x length do:
      [x.i ← self next].
   ↑x]
last
   [↑array.position]
last: n
   [↑(array.(position-n+1 to: position)) copy]
loc    "synonym for compiler"
   [↑position]
myend
   [↑position≥limit]
next  "simple result"
   [self myend⇒ [↑self pastend]
   ↑array.(position ← position+1)] primitive: 17
next ← x      "simple arg"
   [self myend⇒ [↑self pastend ← x]
   ↑array.(position ← position+1) ← x] primitive: 18
of: array
   [position ← 0. limit ← array length]
of: array from: position to: limit
   [position ← position-1]
pastend
   [↑false]
pastend ← x
   [array ← array grow. limit ← array length.
```

```
    ↑self next ← x]
peek | x
    [x← self next⇒ [position ← position-1.  ↑x]   "peek at next element"
    ↑false]
pop         "use it as a LIFO"
    [position<1⇒ [↑false]
    position ← position-1. ↑array'(position+1)]
pop: n | t
    [position<n⇒ [↑false]
    t ← self last: n.
    position ← position-n. ↑t]
position
    [↑position]
print: obj
    [obj printon: self]
reset
    [position ← 0]
semicrtab [self append: ';
    ']
skip: x
    [position ← position+x]
space
    [self next ← 040]
tab
    [self next ← 011]
upto: x | y s
    [s ← Stream default.
    for° y from: self do°
        [y=x⇒[↑s contents]
        s next ← y]
    ↑s contents]
!
```

"String"

Class new title: 'String';
subclassof: Array;
fields: '';
asFollows!

This class has not yet been commented

As yet unclassified

```
+ arg
    [↑self concat: arg]
- s | i c   "Return a negative, zero, or positive integer as I compare < = or > s"
    ["The collation sequence is ascii with case differences ignored."
    for: i to: (self length min: s length) do:
        [(c←(UpperCase'(self'i + 1))-(UpperCase'(s'i + 1)))≠0⇒ [↑c*2]].
    ↑self length-s length*2]
< s
    ["Return true iff I collate before s.  The collation sequence is ascii with case differences
ignored."
    ↑(self-s)<0]
> s
    ["Return true iff I collate after s.  The collation sequence is ascii with case differences
ignored."
    ↑(self-s)>0]
asBytes | s c
    [s ← Stream default.
    for: c from: self do:
        [s append: c base8; space]
    ↑s contents]
asLI
    | d i
    [d ← self reverse copy.
    for: i to: d length do: [d'i ← d'i-060]
    ↑LongInteger new digits: d neg: false]
asParagraph
    [↑Paragraph new text: self alignment: 0]
asUppercase | s c
    [s ← Stream default.
    for: c from: self do:
        [s next ← UpperCase'(c+1)]
    ↑s contents]
asVector
    [↑self asStream asVector]
compare: s | i v
    [for: i to: [self length<s length⇒[self] s] length do:
        [self'i = (s'i)⇒ []
        v ← self'i compareChar: s'i.
        v ≠ 2⇒ [↑v]]
    ↑self length compare: s length]
hash | x h        "not great, but compatible with FT atom hashing"
    [h ← 13131.
    for: x to: self length do:
        [h ← self'x * h.
        h ← (h lshift: ¯1)+(h lshift: 15)]
    ↑h]
lock [] primitive: 31
match: s | pattern match x y star pound
    [star ← 052. pound ← 043.
    pattern ← self asStream.  match ← s asStream.
    x ← pattern next. y ← match next.
    while: [x⇒[y] false] do:
        [x=star⇒[y isalphanumeric⇒[y←match next]
                x←pattern next]
        x=pound⇒[y isalphanumeric⇒[x←pattern next. y←match next]
                ↑false]
        (x compareChar: y)=2⇒[x←pattern next. y←match next]
        ↑false]
    x⇒[↑x=star]
```

```
    ⇑y@false]
printon: strm | x "print inside string quotes"
    [strm next← 047.
    for° x from: self do°
        [strm next← x.
        x=047⇒[strm next← x]]   "imbedded quotes get doubled"
    strm next← 047]
recopy
    [⇑self copy]
species
    [⇑String]
subst: repl for: key | key1 i nskip result
    [nskip ← 0. key1 ← key'1. result ← Stream default.
    for° i to: self length do°        " the Boyer Slow string replacement "
        [nskip>0⇒ [nskip ← nskip-1]
        self'i = key1⇒
            [self'(i to: (self length min: i+key length-1)) = key⇒
                [result append: repl. nskip ← key length-1]
            result next← self'i]
        result next← self'i]
    ⇑result contents]
unique | u        "copy and intern"
    [   [self length=1⇒[UST1@nil⇒[]
            self'1<128⇒[⇑UST1'(self'1+1)]]].
    ⇑☞a intern: self]
unlock [] primitive: 32
word: x        "read word in String"
    [⇑self'(x+x) + (self'(x+x-1) lshift: 8)]
word: x ← y      "write word in String"
    [self'(x+x-1) ← y lshift: ¯8.
    self'(x+x) ← y land: 0377. ⇑y]
!
```

"Substring"

```
Class new title: 'Substring';
   subclassof: Array;
   fields: 'data map';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
• x
   [⇑data•(map•x)]
• x ← val
   [⇑data•(map•x) ← val]
asStream
   [map isIntervalBy1⇒ "direct stream for simple substrings"
      [⇑Stream new of: data from: map start to: map stop]
   ⇑Stream new of: self from: 1 to: map length]
data: data map: map
length
   [⇑map length]
map
   ["Return my map."
   ⇑map]
species
   [⇑data species]
swap: i with: j | t
   ["By permuting my map (a writable Array), swap elements i and j."
   t ← map•i. map•i ← map•j. map•j ← t.]
!
```

"UniqueString"

```
Class new title: 'UniqueString';
   subclassof: String;
   fields: '';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
'x ← val
   [user notify: 'UniqueStrings are not for writing into']
= x [↑self@x]
asName
[↑self copy]
hash [] primitive: 46
intern: s | i a v
   [a ← self intern: s hash: (i←s hash)⇒ [↑a]
   i ← 1+(i lshift: ¯8).
   v ← USTable'i.
   USTable'i ← Vector new: 2*v length.  "grow that hash bucket"
   for@ a from: v do@      "copy all its contents"
      [a@nil⇒ []
      self intern: a hash: a stringhash]
   ↑self intern: s]
intern: s hash: h | i j v n
   [v ← USTable'(1+(h lshift: ¯8)).
   for@ i to: v length do@     "interning compatible with FT atoms - change it soon"
      [h ← h\v length+1.
      v'h@nil⇒     "empty slot"
         [s is: UniqueString⇒ [↑v'h ← s]     "(when growing)"
         n← ¯4.  for@ j from: v do@
            [j@nil⇒ [n ← n+4]]     "count # empty slots"
         n < v length⇒[↑false] "grow if not 1/4 "
         ↑v'h ← (UniqueString new: s length) str: s]      "new atom"
      s length=(v'h) length⇒ [s=(v'h)⇒[↑v'h]]]
   user notify: 'USTable jammed (UniqueString)']
isinfix | x
   [self length≠1⇒ [↑false]  ↑(self'1) isletter@false]
iskeyword | x  "ends with colon"
   [self length≤1⇒ [↑false]
   x ← self'self length.
   x=072⇒[↑true] ↑x=03]
isuneval | x     "ends with open colon"
   [↑self'self length=03]
makeUST1 | i a v
   [v ← Vector new: 128. a ← String new: 1.
   for@ i to: 128 do@
      [a'1 ← i-1. v'i ← a unique]
   UST1 ← v]
printon: strm
   [strm append: self]
species
   [↑String]
str: s | j
   [for@ j to: s length do@
      [super'j ← s'j]
   ↑self]
stringhash
   [↑super hash]
unique
!
```

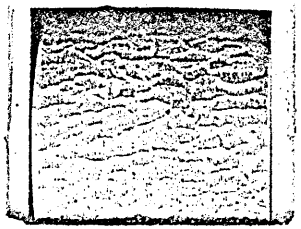
"Vector"

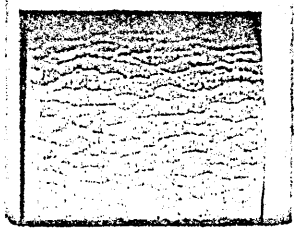
```
Class new title: 'Vector';
   subclassof: Array;
   fields: '';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
, x | v
   [v ← self growby: 1.    "use a stream if youre in a hurry"
   v last ← x. ⇑v]
asVector
nail [user croak] primitive: 31      "Nail me in core and return my core address"
printon: strm | i
   [strm append: '('.
   for i to: self length do
      [strm print: self i; space]
   strm append: ')' ]
unNail [user croak] primitive: 32      "Release me from being nailed"
!
```





"graphical objects"
"Version 3.5 6Sept 77"

"BitRect"

Class new title: 'BitRect';
subclassof: Rectangle;
fields: 'title stripheight data';
asFollows!

This class has not yet been commented

Showing the Bits

```
bitsFromString | s i
    [s ← self slices.
    [data class @ Vector⇒[data length = s length⇒[i←false] i←true]i←true].
    [i⇒[user notify: 'a BitRect lost its format.']].
    for i to: s length do
        [s·i bitsFromString: data·i. ((data·i)asOop) purge].
    ]
bitsFromString: s
    [user notify: 'BitRect already knows its bits.']
bitsIntoString | s i
    [self onscreen @ false ⇒[user notify: 'cannot load from off screen.']
    s ← self slices.
    [data class @ Vector⇒[data length = s length⇒[i←false] i←true]i←true].
    [i⇒[data ← Vector new: s length]].
    for i to: s length do
        [data·i ← (s·i) bitsIntoString. ((data·i) asOop) purge].
    ↑data]
bitsIntoString: s
    [user notify: 'use bitsIntoString instead']
onscreen | a
    [self = (user screenrect intersect: self) ⇒[↑true]
    self moveto: ((user screenrect recopy growby: origin-corner)
        nearest: origin).
    ↑ false]
show [self bitsFromString]
slices | i s a
    [a ← self extent x ☺ stripheight.
    s ← Vector new: (self extent y + stripheight-1) /stripheight.
    for i to: s length do
        [s·i ← Rectangle new origin← origin +
            (0☺stripheight * (i-1)).
        (s·i) extent ← a].
    (s·s length) extent← a x ☺ (self extent y \ stripheight).
    ↑s]
```

How to Use

```
AComment
    ["BitRect is a Rectangle that remembers the bits within it.
    To create and edit one, say:
        (BitRect new fromuser) edit.
    (This installs a BitRectEd in the scheduler)
    "]
edit | a
    [user schedule: (a ← BitRectEditor new picture: self). ↑a]
filin: title | f i s  "read bits from a file"
    [f ← dp0 file: (title concat: '.pic.').
    i ← f nextword.
    self extent← i ☺ f nextword.
    stripheight ← f nextword.
    s ← self slices. data ← Vector new: s length.
    for i to: s length do [data·i ← String new: (s·i) bitStringLength.
        f into: data·i.
        ((data·i)asOop) purge].
    f close]
filout | f i  "write bits on a file"
    [f ← dp0 file: (title concat: '.pic.').
    f nextword ← self extent x.
    f nextword ← self extent y.
```

```
    f nextword ← stripheight.
    for i from: data do [f append: i].
    f close]
fromuser
    [self title: 'BitRect' in: (Rectangle new fromuser)]
title: title  in: rect
    [origin ← rect origin.
    corner ← rect corner.
    stripheight ← 1023/((self extent x + 15)/16).
    self onscreen; bitsIntoString]
```

HouseKeeping

```
printon: strm
    [strm append: 'a BitRect']
title [↑title]
workspace
 ["(cc ← BitRectEditor new) init.
(aa ← BitRect new) fromuser; filin: (☞actionpic asString); show.
cc actionpic: aa.
(aa ← BitRect new) fromuser; filin: (☞toolpic asString); show.
cc toolpic: aa.
(aa ← BitRect new) fromuser; filin: (☞sample asString); edit.
"]
!
```

"BitRectEditor"

```
Class new title: 'BitRectEditor';
  subclassof: Window;
  fields: 'tool picture dirty';
  declare: 'actionpic actionbuts picframe toolpic windowmenu tools
';
  asFollows!
```

This class has not yet been commented

As yet unclassified

```
AComment
   ["BitRectEditor edits BitRects.
   To create, say:
      (BitRect new fromuser) edit.
   (This installs a BitRectEditor in the scheduler)
   To edit the two menu pictures, say:
      aa ← (BitRect new fromuser) edit.
      aa editActionpic.
   or
      aa editToolpic.
   (To see your picture in the menu, leave the window and come back in).
   "]
actionpic: a [actionpic ← a]
bluebug | a oldf i
   [windowmenu bug
    =1 ⇒[self leave. self erase. ⇑false];  "under"
   =2 ⇒[picture bitsIntoString. a ← picture origin. " move "
      oldf ← frame.
      user waitnobug. user waitbug.
      while☺ user anybug do☺ [picture dragto: user mp].
      picture onscreen.
      frame moveby: picture origin - a.
      for☺ i from: (oldf minus: frame) do☺ [i clear].
      self enter];
   =3 ⇒[a ← picture corner.  " grow "
      user waitnobug. user waitbug.
      while☺ user anybug do☺ [picture corner← (user mp); color: 51
         mode: xoring; color: 51 mode: xoring].
      picture title: picture title in: picture.
      frame growby: picture corner - a];
   =4 ⇒[self leave. self erase.   " close "
       user unschedule: self. ⇑false];
   =5 ⇒[self leave. picture filout]  "filout"
   ]
editActionpic | pic
   [pic ← actionpic copy.
   pic moveto: picture origin.
   self picture: pic]
editToolpic | pic
   [pic ← toolpic copy.
   pic moveto: picture origin.
   self picture: pic]
enter | pt p
   [super show.
   pt ← p ← frame origin.
   actionpic moveto: pt; show.
   pt ← actionbuts•1 moveto: pt. "action"
   pt ← actionbuts•3 moveto: pt. "mode"
   pt ← actionbuts•4 moveto: pt. "width"
   tool brushpt: pt + (0 ☺ 40).
   pt ← p + (0☺20).
   pt ← actionbuts•2 moveto: pt. "tone"
   pt ← actionbuts•5 moveto: pt. "grid"
   tool graypenpt: pt.
   toolpic moveto: p+(0☺40); show.
   tools moveto: p+(0☺40); setvalue: tool. "tools"
```

```
    picture show.
    picframe ← picture inset: (¯2@¯2).
    tool showon: actionbuts.
    dirty ← false]
fixframe: r
    [picture moveto: (r origin + (20@40)); onscreen.
    r origin← picture origin - (20@40).
    actionpic moveto: r origin.
    toolpic moveto: r origin + (0@40).
    r corner← ((picture corner max: actionpic corner) max: toolpic corner).
    ↑r]
init | t i
    [t ← Vector new: 6.
    for: i to: t length do: [t•i ← BitRectTool new init].
    tools ← (RadioButtons new) vec: t at: 0@0 width: 20.
    windowmenu ← Menu new string: 'under
move
grow
close
filout'.
    self initmenu1]
initmenu1 | s z
    [s ← Vector new: 7. z ← 20.
    s•1 ← (RadioButtons new) vec: ☞(setbrush paint block draw line blowup) at: 0@0 height: z.
"action"
    s•2 ← (RadioButtons new) vec: (black, dkgray, gray, ltgray, white) at: 0@0 height: z. "tone"
    s•3 ← (RadioButtons new) vec: (0, 1, 2, 3) at: 0@0 height: z. "mode"
    s•4 ← (RadioButtons new) vec: (1, 2, 4, 8) at: 0@0 height: z. "width"
    s•5 ← (RadioButtons new) vec: (1, 2, 4, 8, 16, 32) at: 0@0 height: z. "grid"
    actionbuts ← s.]
leave | i
    [[dirty⇒[picture bitsIntoString. dirty ← false]].
    for: i from: (frame minus: picframe) do: [i clear] ]
picture: picture
    [tool ← tools•1.
    self frame: (picture origin - (20@40) rect: 0@0)]
redbug
    [picframe has: (dirty ← user mp) ⇒[tool redbug]
    toolpic has: dirty ⇒
        [tool ← tools bug: dirty.
        tool showon: actionbuts]
    actionpic has: dirty ⇒
        [tool setfrom: actionbuts]
    ]
title [↑picture title]
toolpic: a [toolpic ← a]
yellowbug [self redbug]
!
```

"BitRectTool"

Class new title: 'BitRectTool';
fields: 'action pencil brush mode tone grid';
declare: 'rblock brushpt gpenrect graypens blowqmenu ';
asFollows!

This class has not yet been commented

As yet unclassified

```
bitpaint: brect on: srect | r a i
    [pencil width: 1. r ← 0@0 rect: 3@3.
    while☐ true do☐
        [while☐ (brect has: (a ← user mp)) do☐
            [r moveto: brect origin + (i ← (a-brect origin)|4).
            pencil place: srect origin + (i/4).
            user redbug⇒[r color: black mode: storing.
                pencil black; go: 0]
            user yellowbug⇒[r color: white mode: storing.
                pencil white; go: 0]
            user bluebug⇒[1=blowqmenu bug⇒[↑nil] ]].
        user anybug ⇒[srect comp; comp. brect comp; comp]
        ]
    ]
blowup | svbits svrect brect
    [rblock origin ← user mp.
    while☐ user anybug do☐ [rblock corner← user mp; comp; comp].
    [rblock extent < (100@100)⇒[rblock empty ⇒[] ] ↑nil].
    (svrect ← rblock corner rect: 0@0) extent← 4@4 + (rblock extent *4).
    [user screenrect has: svrect corner ⇒[]
        svrect moveto: user screenrect corner/4].
    svbits ← svrect bitsIntoString.
    brect ← svrect inset: 2@2.
    rblock blowup: brect origin by: 4.
    self bitpaint: brect on: rblock.
    svrect bitsFromString: svbits.
    ]
brushpt: a [brushpt← a]
graypenpt: pt
    [gpenrect moveto: pt.
    ↑pt+ (gpenrect extent x @ 0)]
init | i pt t
    [(pencil ← Turtle new) init; black; width: 2.
    (brush ← BitRect new) title: 'brush' in: (0@0 rect: 16@16).
    tone ← black. mode ← 0. grid ← 1. action ← ☞draw.
    graypens @ nil ⇒
        [rblock ← 0@50 rect: 9@59.
        (t ← Turtle new init) black.
        graypens ← Vector new: 8.
        for☐ i to: 8 do☐ [t width: i. pt ← i@i.
            rblock clear: white.
            t place: rblock origin + (pt/2); go: 0.
            graypens'i ← rblock bitsIntoString].
        rblock clear: background.
        gpenrect ← rblock recopy.
        brushpt ← 0@0.
        blowqmenu ← Menu new string: ' quit ']
    ]
maybegray
    [action @ ☞draw or: action @ ☞drawgray ⇒
        [tone = white ⇒[action ← ☞draw]; = black ⇒[action ← ☞draw]
        gpenrect bitsFromString: graypens' pencil width.
        action ← ☞drawgray] ]
redbug
    [action @ ☞paint ⇒[while☐ user redbug do☐
        [brush brush: (user mp | grid) mode: mode color: tone]];
    @ ☞block ⇒[rblock origin← user mp | grid.
        while☐ user redbug do☐ [rblock corner← user mp | grid].
        rblock empty ⇒[]
```

```
        rblock color: tone mode: mode];
    @ ☞draw ⇒[pencil place: user mp.
        while☺ user redbug do☺ [pencil goto: user mp]];
    @ ☞drawgray ⇒[while☺ user redbug do☺
        [gpenrect brush: user mp mode: mode color: tone]];
    @ ☞setbrush ⇒[rblock origin← (user mp | grid); extent← 9☺9.
        while☺ user redbug do☺
            [rblock corner← (user mp | grid); comp; comp].
        rblock empty ⇒[]
        brush title: 'brush' in: rblock.
        brush moveto: brushpt - (0 ☺ brush extent y).
        brush show.
        action ← ☞paint];
    @ ☞line ⇒[pencil place: user mp | grid.
        while☺ user redbug do☺ [pencil stretchto: user mp | grid].
        pencil goto: user mp | grid];
    @ ☞blowup ⇒[self blowup]
    ]
setfrom: butvec | pt
    [butvec•1 has: (pt ← user mp) ⇒
        [action ← butvec•1 bug: pt.
        self maybegray.
        action @ ☞paint ⇒[brush show]]
    butvec•2 has: pt ⇒[tone ← butvec•2 bug: pt. self maybegray.
        tone = white ⇒[pencil white] pencil black]
    butvec•3 has: pt ⇒[mode ← butvec•3 bug: pt]
    butvec•4 has: pt ⇒[pencil width: (butvec•4 bug: pt). self maybegray]
    butvec•5 has: pt ⇒[grid ← butvec•5 bug: pt]
    butvec•7 has: pt ⇒[brush show]
    ]
showon: butvec
    [[action @ ☞drawgray ⇒[action ← ☞draw]].
    butvec•1 setvalue: action.
    butvec•2 setvalue: tone.
    butvec•3 setvalue: mode.
    butvec•4 setvalue: pencil width.
    butvec•5 setvalue: grid.
    self maybegray.
    action @ ☞paint ⇒[(butvec•7 ← brush) moveto: brushpt - (0☺ brush extent y). brush show]
    ]
!
```

"Cursor"

Class new title: 'Cursor';
fields: 'bitstr';
asFollows!

This class has not yet been commented

As yet unclassified

```
frompage1       "load this cursor from the hardware locations"
   [bitstr ← String new: 32.
   BitBlt new forCursor; sourcebase← 0431; copyToString: bitstr mode: storing]
fromString: bitstr
fromtext: str | i j n
   [bitstr ← String new: 32.
   for i to: 16 do
      [n ← 0.
      for j to: 16 do
         [n ← n*2 + (str•(i-1*17+j+1)-060)]
      bitstr word: i ← n]]
printon: strm | i
   [strm append: 'Cursor new fromtext: '".
   for i to: 16 do
      [strm cr.
      (bitstr word: i) printon: strm base: 2]
   strm append: '".']
showwhile expr | oldcursor value
   [oldcursor ← Cursor new frompage1.
   self topage1. value ← expr eval.
   oldcursor topage1. ↑value]
topage1       "copy this cursor into the hardware locations"
   [BitBlt new forCursor; destbase← 0431; copyFromString: bitstr mode: storing]
!
```

"Menu"

Class new title: 'Menu';
fields: 'str text thisline frame';
asFollows!

This class has not yet been commented

As yet unclassified

```
bug | pt bits
   [pt ← user mp - thisline center.     "center prev item on mouse"
   text frame moveby: pt. thisline moveby: pt.
   frame moveby: pt. bits ← frame bitsIntoString."save background"
   frame clear: black. text show.
   user nobug ⇒
         [frame bitsFromString: bits. ⇑0]"accidental bug returns 0"
   thisline comp.
   while☺ true do☺
      [text frame has: (pt ← user mp) ⇒
         [user anybug⇒
            [thisline has: pt⇒[]
            pt ← text ptofpt: pt.
            thisline comp.  "selection follows mouse"
            thisline moveto: text frame origin x ☺ pt y.
            thisline comp]
         frame bitsFromString: bits. "restore background, return index"
         ⇑1+ (thisline origin y-text frame origin y / text lineheight)]
      thisline comp.       "he left the menu"
      until☺ [text frame has: user mp] do☺
         [user nobug⇒[frame bitsFromString: bits. ⇑0]] "return 0 for abort"
      thisline comp]       "he came back"
   ]
rebug
   [user waitbug.  "wait for button down again"
   ⇑"bugcursor showwhile☺" self bug]
string: str | i pt tpara
   [text ← Textframe new para: (tpara ← str asParagraph)
            frame:  (Rectangle new origin: (pt ←  0 ☺ 0)
                                    corner: 1000 ☺ 1000).
   for☺ i to: str length+1 do☺
      [pt ← pt max: (text ptofchar: i)]
   text frame growto: pt + (4 ☺ text lineheight).
   tpara center.
   frame ← text frame inset: ¯2 ☺ ¯2.
   thisline ← Rectangle new origin: text frame origin
            corner: text frame corner x ☺ text lineheight]
!
```

"Point"

```
Class new title: 'Point';
   fields: 'x y';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
≤ pt
   [↑x≤pt x and: y≤pt y]
* scale
   ["Return a Point that is the product of me and scale (which is a Point or Number)"
   ↑(x * scale asPtX) @ (y * scale asPtY)
   ]
+ delta
   ["Return a Point that is the sum of me and delta (which is a Point or Number)"
   ↑(x + delta asPtX) @ (y + delta asPtY)
   ]
- delta
   ["Return a Point that is the difference of me and delta (which is a Point or Number)"
   ↑(x - delta asPtX) @ (y - delta asPtY)
   ]
/ scale
   ["Return a Point that is the quotient of me and scale (which is a Point or Number)"
   ↑(x / scale asPtX) @ (y / scale asPtY)
   ]
< pt
   [↑x<pt x and: y<pt y]
= pt
   [↑x=pt x and: y=pt y]
asPoint
   ["Return self."]
asPtX
   [↑x]
asPtY
   [↑y ]
asRectangle
   ["Return a Rectangle with me as both origin and corner."
   ↑self rect: self]
asRectCorner "pretend to be a Rectangle for Rectangle +-*/"
asRectOrigin "pretend to be a Rectangle for Rectangle +-*/"
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)]
printon: strm
   [strm print: x; append: '@'; print: y]
rect: p       "infix creation of rectangles"
   [↑Rectangle new origin: self corner: p]
x   [↑x]
x: x y: y
x ← x
y [↑y]
y ← y
| grid
   [↑Point new x: x|grid y: y|grid]
!
```

"RadioButtons"

```
Class new title: 'RadioButtons';
   fields: 'vec cur rect size';
   declare: 'rr ';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
· a
   [self comp: cur to: a.
   ↑vec·(cur ← a)]
bug: pt | r a
   [r ← (pt - rect origin - (1@1)) / size.
   a ← r x + r y + 1.
   self comp: cur to: a.
   ↑vec · (cur ← a)]
comp: a to: b | r
   [a=b ⇒[]
   r ← [size = rect extent y ⇒[1@0] 0@1] * size.
   [a=0 ⇒[rect color: 0202 mode: xoring]
      rr moveto: (r *(a-1) + rect origin);
         color: 0202 mode: xoring].
   b=0 ⇒[rect color: 0202 mode: xoring]
      rr moveto: (r *(b-1) + rect origin);
         color: 0202 mode: xoring]
has: pt [↑rect has: pt]
moveto: pt
   [rect moveto: pt.
   rr extent← size@size.
   cur ← 0.
   ↑rect corner x @ rect origin y]
setvalue: v | a
   [for: a to: vec length do: [v = (vec·a) ⇒[self·a. ↑a]].
   user notify: 'v not in vec']
value [↑vec·cur]
vec [↑vec]
vec: vec at: r height: size
   [rect ← r rect: r+ ((vec length @ 1)*size).
   cur ← 0. rr ← 0@0 rect: size@size]
vec: vec at: r width: size
   [rect ← r rect: r+ ((1 @ vec length)*size).
   cur ← 0. rr ← 0@0 rect: size@size]
!
```

"Rectangle"

Class new title: 'Rectangle'; fields: 'origin corner'; asFollows!

This class has not yet been commented

As yet unclassified

```
* scale
    ["Return a Rectangle which is the product of me and scale (which is a Rectangle, Point, or
Number)"
    ↑(origin * scale asRectOrigin) rect: (corner * scale asRectCorner)
    ]
+ delta
    ["Return a Rectangle which is the sum of me and delta (which is a Rectangle, Point, or
Number)"
    ↑(origin + delta asRectOrigin) rect: (corner + delta asRectCorner)
    ]
- delta
    ["Return a Rectangle which is the difference of me and delta (which is a Rectangle, Point, or
Number)"
    ↑(origin - delta asRectOrigin) rect: (corner - delta asRectCorner)
    ]
/ scale
    ["Return a Rectangle which is the quotient of me and scale (which is a Rectangle, Point, or
Number)"
    ↑(origin / scale asRectOrigin) rect: (corner / scale asRectCorner)
    ]
= r
    [↑origin = r origin and: corner = r corner]
asRectangle
    ["Return self."]
asRectCorner
    [↑corner ]
asRectOrigin
    [↑origin ]
bitsFromString: str [user croak] primitive: 52
bitsIntoString | extent str
    [extent ← corner-origin.
    str ← String new: 2 * extent y * (extent x+15/16).
    self  bitsIntoString: str. ↑str]
bitsIntoString: str [user croak] primitive: 51
bitStringLength
    [↑ 2*self extent y*(self extent x +15/16)]
blowup: at by: scale
    | z z1  inc sinc slice width height dest i j
    [width ← self extent x. height ← self extent y.
    dest ← Rectangle new origin: at extent: (width*scale)@(height*scale).
        [(dest has: origin) or: (dest has: corner) ⇒
            [z ← self bitsIntoString. dest outline.
            self moveto: dest origin. self bitsFromString: z]
        dest outline].
    z ← 1@0. z1 ← 0@1.
    for: i to: 2 do:                        "first do horiz, then vert"
        [inc ← z * ¯1. sinc ← z * scale.
        slice ← Rectangle new
            origin: [i = 1 ⇒[self origin] at] + (z * (width - 1))
            extent: z + (z1 * height).
        dest ← at + (z * (scale * (width - 1))).
        for: j to: width do:                "slice it up"
            [slice blt: dest mode: storing.
            dest ← dest - sinc.
            slice moveby: inc]
        slice ← Rectangle new origin: at - inc
                    extent: (z1*height)+(z*(scale-1)).
        for: j to: width/scale + 1 do:      "clear slice source"
            [slice clear: white. slice moveby: sinc]
        slice ← Rectangle new origin: at
```

```
                    extent: (z1 * height) + (z * ((scale*width)-1)).
        for: j to: scale - 2 do:                "spread it out"
            [slice blt: at - inc mode: oring]
        j ← z. z ← z1. z1 ← j.                  "flip to do vertical"
        j ← height. height ← scale*width. width ← j]
    ]
blt: dest mode: mode [user croak] primitive: 47
bltcomp: dest mode: mode [user croak] primitive: 48
border: thick color: color  "paints a border withoud disturbing interior"
    [(Rectangle new
        origin: origin-(thick@thick) corner: (corner x+thick)@origin y)
    clear: color;
    moveto: (origin x-thick)@corner y; clear: color;
    origin ← corner x@(origin y-thick); clear: color;
    moveto: origin-(thick@thick); clear: color]
boxcomp
    [(self +1) color: black mode: xoring.
    (self +¯1) color: black mode: xoring]
brush: dest mode: mode color: color [user croak] primitive: 49
center
    [⇑origin+corner/2]
clear      "default is backround"
    [self color: background mode: storing]
clear: color
    [self color: color mode: storing]
color: color mode: mode [user croak] primitive: 50
comp
    [self color: black mode: xoring]
comp: color
    [self color: color mode: xoring]
corner [⇑corner]
corner ← corner
dragto: dest | v i
    [self blt: dest mode: storing.
    v ← dest rect: dest+self extent.
    for: i from: (self minus: v) do: [i clear].
    origin ← dest. corner ← v corner]
empty
    [⇑(origin < corner)@false]
extent
    [⇑corner-origin]
extent ← extent
    [corner ← origin+extent. ⇑extent]
flash [self comp; comp]
fromuser
    [while: user anybug do: [].
    until: user anybug do: [origin ← corner ← user mp].
    while: user anybug do: [corner ← user mp. self comp. self comp].
    ]
growby: pt
    [corner ← corner + pt]
growto: corner
has: pt
    [origin≤pt⇒[⇑pt≤corner] ⇑false]
height
    [⇑corner y - origin y]
height ← h "change my bottom y to make my height h"
    [corner y ← origin y + h]
inset: p1
    [⇑origin+p1 rect: corner-p1]
inset: p1 and: p2
    [⇑origin+p1 rect: corner-p2]
intersect: r
    [⇑(origin max: r origin) rect: (corner min: r corner)]
minus: r | int s v
    [int ← self intersect: r.
    int empty ⇒[⇑self inVector]
    v ← Vector new: 5.
    [origin x ≠ int origin x ⇒
        [v•5 ← v•1 ← origin rect: int origin x @ corner y]].
    [origin y ≠ int origin y ⇒
```

```
        [v•2 ← origin rect: corner x ☺ int origin y]].
    [corner x ≠ int corner x ⇒
        [v•3← int corner x ☺ origin y rect: corner]].
    [corner y ≠ int corner y ⇒
        [v•4 ← origin x ☺ int corner y rect: corner]].
    s ← (Vector new: 4) asStream.
    for☺ int to: 4 do☺
        [v•int @ nil ⇒[]
        v•(1+int) @ nil ⇒[s next← v•int]
        s next← (v•int minus: v•(int+1))•1].
    ⇑s contents]
moveby: pt
    [origin ← origin+pt. corner ← corner+pt]
moveto: pt
    [corner ← corner+pt-origin. origin←pt]
nearest: pt
    [⇑((origin x max: pt x) min: corner x) ☺
    ((origin y max: pt y) min: corner y)]
origin [⇑origin]
origin: origin corner: corner
origin: origin extent: extent
    [corner ← origin+extent]
origin ← origin
outline  "default border is two thick"
    [self outline: 2]
outline: thick | t
    [t ← (¯1☺¯1)*thick.
    (self inset: t) clear: black.  self clear: white]
printon: strm
    [self fullprinton: strm]
width
    [⇑corner x - origin x]
width ← w "change my right x to make my width w"
    [corner x ← origin x + w]
!
```

"Turtle"

Class new title: 'Turtle';
fields: 'pen ink width dir x xf y yf frame]';
asFollows!

This class has not yet been commented

As yet unclassified

```
black
    [ink ← ~3]
erase
    [frame clear: white]
frame: frame
go: length [user croak] primitive: 53
goto: pt [user croak] primitive: 54
home
    [self place: frame center-frame origin. dir ← 270]
init
    [ink ← ~3. pen ← width ← 1. dir ← 270.
    frame ← user screenrect.
    self place: frame center]
ink: ink
pen: pen
pendn
    [pen ← 1]
penup
    [pen ← 0]
place [↑x@y]
place: pt
    [x ← pt x. y ← pt y. xf ← yf ← 0]
put: char font: font [user croak] primitive: 56
show: str font: font | a
    [for° a from: str do°
        [self put: a font: font]]
stretchto: pt | t old
    [t ← Turtle new init. old ← x@y.
    t xor; place: old; goto: pt; place: old; goto: pt]
turn: angle
    [dir ← dir+angle \ 360] primitive: 55
up
    [dir ← 270]
white
    [ink ← ~1]
width [↑width]
width: width
xor
    [ink ← ~2]
!
```

"sets and dictionaries"

"Version 3.5 6 Sept 77"

"Dictionary"

```
Class new title: 'Dictionary';
   subclassof: HashSet;
   fields: 'values';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
• name
   [⇑values•(self findorerror: name)]
• name ← val
   [⇑values•(self findorerror: name) ← val]
clean | name    "release unreferenced entries"
   [for: name from: self do:    "slick, huh"
      [(self•name) refct = 1 ⇒ [self delete: name]]]
copyfrom: dict
   [self objects ← dict objects copy.
   values ← dict values copy]
growto: size | name copy
   [copy ← self class new init: size."create a copy of the new size"
   for: name from: self do:
      [copy insert: name with: self•name]"hash each entry into it"
   self copyfrom: copy]
init: size
   [values ← Vector new: size. super init: size]
insert: name with: value
   [self insert: name.
   values•(self findorerror: name) ← value]
insertall: names       "default value is nil"
   [self insertall: names with: (Vector new: names length)]
insertall: names with: vals | i        "insert many entries"
   [for: i to: names length do:
      [self insert: names•i with: vals•i]]
invert: obj | i
   [for: i to: values length do:
      [values•i=obj⇒[⇑objects•i]]
   ⇑false]
lookup: name | x
   [x ← self find: name⇒ [⇑values•x] ⇑false]
values [⇑values]
with: names values: vals | i
   [for: i to: names length do:
      [self insert: names•i with: vals•i]]
!
```

"HashSet"

Class new title: 'HashSet';
fields: 'objects';
asFollows!

This class has not yet been commented

As yet unclassified

```
asStream
    [↑self contents asStream]
contents | obj strm
    [strm ← (Vector new: objects length) asStream.
    for° obj from: objects do°
        [obj@nil⇒[] ☞DeleteDenTry=obj⇒[]
        strm next← obj]
    ↑strm contents]
delete: obj | a        "delete, then rehash"
    [   [obj is: Vector⇒[for° a from: obj do°
            [objects·(self findorerror: a) ← ☞DeleteDenTry]]
        objects·(self findorerror: obj) ← ☞DeleteDenTry].
    ↑self growto: objects length]
find: obj | i "↑index if found, else false"
    [i ← self findornil: obj.
    objects·i=obj⇒[↑i] ↑false]
findorerror: name | i s
    [i ← self find: name⇒ [↑i]
    user show: name+
' cannot be found. Type a correction, or ctl-d to restart or top-blank to debug.'; cr.
    s ← user request: '*'.
    s@nil⇒[thisContext debug]
    s@false⇒[user restart]
    ↑self findorerror: nil'ss "eval correction"]
findorinsert: obj | i      "insert if not found, "
    [i ← self findornil: obj.
    objects·i=obj⇒[↑i]  "found it"
    self sparse⇒[objects·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.
    for° i to: objects length do°
        [loc ← [loc=objects length⇒[1] loc+1].
        objects·loc @ nil⇒ [↑loc]
        objects·loc = obj⇒ [↑loc]]
    ↑1 "table full - caller must check for hit"]
growto: size | copy i
    [copy ← self class new init: size."create a copy"
    for° i from: self do°
        [copy insert: i] "hash each entry into it"
    objects ← copy objects]
has: obj
    [self find: obj⇒ [↑true] ↑false]
init
    [self init: 4]
init: size
    [objects ← Vector new: (size max: 2)]
insert: obj | i
    [self findorinsert: obj. ↑obj]
objects [↑objects]
objects← objects
shrink | table oldtable
    [oldtable ← self.
    table ← oldtable growto: (2 max: oldtable size/2).
    until° table size=oldtable size do°
        [(oldtable size-table size) print.  user show: ' '.
        oldtable ← table.
        table ← oldtable growto: (2 max: oldtable size/2)]
    ↑table]
```

```
size [↑objects length]
sparse | i n
    ["↑true if (1 max: 1/4 of table) is nil"
    n ← objects length.
    for° i from: objects do°
        [i@nil⇒[(n←n-4)≤0⇒[↑true]]]
    ↑false]
!
```

"MessageDict"

Class new title: 'MessageDict';
subclassof: HashSet;
fields: 'methods "<Vector of Strings> which are the compiled methods for each message"
literals "<Vector of Vectors> which hold pointers to literals used in the methods"
code "<Vector of Strings> which are the source text for each message"
backpointers "<Vector of Vectors> which are the tables of text location vs pc for each message"';
asFollows!

MessageDicts hold the source code and compiled methods for each message to a class. The source code is a packed paragraph (see Paragraph packIntoString). The methods contain pointers to literals, and must be specially freed. If a method is being executed during its redefinition, its release must be delayed (its literals gets held in CodeKeeper). Finally, MessageDicts must be copied to be grown, so that current use is not disturbed.

Initialization

```
growto: size | name copy i
    [copy ← MessageDict new init: size."create a copy of the new size"
    for: name from: self do:
        [i ← self findorerror: name.        "hash each entry into it"
        copy ← copy insert: name method: methods·i
            literals: [literals@nil⇒[nil] literals·i]  code: code·i backpointers: nil]
    ↑copy]
init: size
    [methods ← Vector new: size.
    code ← Vector new: size.
    super init: size]
insert: name method: m literals: l
        code: c backpointers: b | i copy
    [i ← self find: name⇒       "if name is already there"
        [copy ← self literalsIn: methods·i.
        [literals@nil⇒[self freeLiterals: copy]
            literals·i@nil⇒[self freeLiterals: copy]
            literals·i ← nil].
        [(methods·i) refct>1⇒[CodeKeeper next← copy]].
        self holdLiterals: l.
        methods·i ← m. code·i ← c]      "then insert it, and return self"
    copy ← [self sparse⇒[self]
        self growto: methods length*2].     "Otherwise, copy if necessary"
    copy objects·(copy findornil: name) ← name."and insert"
    ↑copy insert: name method: m literals: l
            code: c backpointers: b]
```

Access to parts

```
code: name
    [↑code·(self findorerror: name)]
code: name ← str
    [↑code·(self findorerror: name) ← str]
literals: name
    [↑self literalsIn: methods·(self findorerror: name)]
method: name
    [↑methods·(self findorerror: name)]
```

Code aspect of Strings

```
freeLiterals: v | m i t        "lower refct of all literals"
    [v length=0⇒[]
    m ← v nail.
    for: i to: v length do:
        [t ← mem·(m+i-1). v·i ← nil. mem·(m+i-1) ← t]
```

```
    v unNail]
holdLiterals: v | m i t        "raise refct of all literals"
    [v@nil⇒[] v length=0⇒[]
    m ← v nail.
    for: i to: v length do:
        [t ← v'i. mem'(m+i-1) ← ¯1. v'i ← t]
    v unNail]
literalsIn: method | i v        "return the literal vector imbedded in this method"
    [method@nil⇒[⇑Vector new: 0]
    method length<8⇒[⇑Vector new: 0]
    method'2=41⇒[⇑Vector new: 0]
    v ← Vector new: method'6-6/2.
    for: i to: v length do:
        [v'i ← (method word: 3+i) asObject]
    ⇑v]
!
```

"ObjectReference"

```
Class new title: 'ObjectReference';
   fields: 'object';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
printon: strm
   [strm append: '->'; print: object]
value [↑object]
value ← object
!
```

"SymbolTable"

Class new title: 'SymbolTable';
subclassof: Dictionary;
fields: '';
asFollows!

This class has not yet been commented

As yet unclassified

```
• name
   [⇑(super•name) value]
• name ← x
   [⇑(super•name) value ← x]
declare: name | a
   [name is: Vector⇒[for° a from: name do°
         [self declare: a]]
   self has: name⇒ []
   Undeclared has: name⇒
      [super insert: name with: (Undeclared ref: name).
      Undeclared delete: name]
   self insert: name with: nil]
declare: name as: x
   [self declare: name.
   self•name ← x]
define: name as: x
   [self has: name⇒ [self•name ← x]
Undeclared has: name⇒
      [super insert: name with:(Undeclared ref: name).
      self•name ← x.
   Undeclared delete: name]
   self insert: name with: x]
growto: size | name copy
   [copy ← self class new init: size."create a copy of the new size"
   for° name from: self do°
      [copy insert: name withref: (self ref: name)]"hash each entry into it"
   self copyfrom: copy]
insert: name with: x
   [super insert: name with: (ObjectReference new value← x)]
insert: name withref: ref
   [super insert: name with: ref]
invert: obj | i
   [for° i to: values length do°
      [objects•i@nil⇒[]
      obj @ (values•i) value ⇒[⇑objects•i]]
   ⇑false]
ref: name
   [⇑super•name]
ref: name ← val
   [⇑super•name ← val]
!
```

"text objects"
"version 3.5 6 Sept 77"

"Dispframe"

```
Class new title: 'Dispframe';
   subclassof: Stream;
   fields: 'text';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
clear
   [self reset. self show]
eachtime | t
   [text window has: user mp⇒
      [user kbck⇒[t← self read.
         self print: [t@nil⇒[nil doit] nil'st].
         self cr; append: '☺'; show]]
   user anybug⇒[⇑false]]
ev | t
   [while☺ [t ← self request: '
☺'] do☺
      [t@nil⇒ [self print: nil doit; show] "redo"
      self print: nil 'st;.show]
   ⇑false]
firsttime
   [text window has: user mp⇒
      [text window outline. self cr; append: '☺'; show]
   ⇑false]
lasttime
   [self skip: [¯2 max: 0-position]; show]
moveto: pt
   [(text window inset: ¯2☹¯2) dragto: pt-(¯2☹¯2)]
outline
   [text window outline]
read | n t
   [self show. n ← 0.
   while☺ true do☺
      [t ← user kbd.
      t=4⇒ [self skip: 0-n; append: 'done.'; show. ⇑false]; "ctl-d for done"
      =8⇒ [n=0⇒[self show]. self skip: ¯1. n ← n-1.
            user kbck⇒[]. self show]; "backspace"
      =2⇒ [self skip: ¯1-n. ⇑nil]; "redo (top blank)"
      =30⇒ [t ← self last: n.
            self next ← 30; cr; show. ⇑t]; "do-it (LF)"
      =23⇒ [n=0⇒[self show] self skip: ¯1. n←n-1.
            while☺ (n>0 and: self last tokenish) do☺
               [self skip: ¯1. n←n-1]. self show]; "ctl-w"
      =24⇒[self reset; append: '☺'; show. n←0] "ctl-x"
      self next ← t. n ← n+1.
      user kbck⇒ [] self show]]
rect: r
   [text ← Textframe new para: nil frame: r.
   self of: (String new: 16). self clear]
request: s
   [self append: s.⇑self read]
show
   [text show: self contents.
   until☺ text lastshown≥self position do☺
      [self dequeue: (text scrolln: 2).
      text show: self contents]]
!
```

"FontWindow"

```
Class new title: 'FontWindow';
   fields: 'frame font fontht fontraster fontxtabl bitsetter char
charx
         charwid charstr altostyle fontnumber clearframe scale
boxer';
   declare: 'fontmenu ';
   asFollows
```

This class has not yet been commented

As yet unclassified

```
altostyle: altostyle fontnumber: fontnumber at: origin
   [      "set up an instance"
   [fontmenu@nil⇒[self init]].
   scale ← 9. charstr ← String new: 1. char ← 65. bitsetter ← BitBlt init.
   boxer ← Rectangle new
            origin: 0 @ 0 extent: (scale-1) @ (scale-1).
   frame ← Rectangle  new origin: origin  extent:  scale @ 0.
   clearframe ← Rectangle  new origin: origin  extent:  scale @ 0.
   self setfont: altostyle fonts•fontnumber.
   ]
appendxtable: thefont
   [                                "put font'sxtable on end of a grown/shrunk font"
   thefont ← thefont concat: font • ((fontxtabl * 2 - 1) to: font length).
   ↑thefont.
   ]
eachtime                          "while active"
   [clearframe has: user mp⇒
      [
      user redbug ⇒
         [self setbit: user mp color: black]          "make dot black"
      user yellowbug ⇒
         [self setbit: user mp color: white]          "make dot white"
      user bluebug ⇒
         [fontmenu bug
            =1⇒[];
            =2⇒[];
            =3⇒[self setwidth];                       "grow character"
            =4⇒[self frame]                           "move fontwindow"
         ]
      user kbck⇒
         [char ← user kbd.
            char = 04 ⇒ [self updateseglength: font raster: fontraster.
                         self updatemaxwidth.      "clean things up"
                         thisContext debug.        "then allow fishing"]

            char = 0177⇒ [clearframe clear.
                          self updateseglength: font raster: fontraster.
                          self updatemaxwidth.     "clean things up"
                          user unschedule: self. ↑false]
            self setchar: char]
       ]
      ↑false
   ]
firsttime                         "upon entry"
   [
   clearframe has: user mp⇒[self show]
   ↑false
   ]
frame
   [clearframe clear.
   frame moveto:
   (OriginCursor showwhile
      [user waitbug⇒[user mp]]).
   self setchar: char.
   ]
```

```
help
["
**to create a window for editing default font 0 at middle-click:
    user schedule: (yourfontwindow ← FontWindow new
       altostyle: DefaultTextStyle
       fontnumber: 1
       at: (OriginCursor showwhile:
          [user waitbug ⇒[user mp]])).

**to create a new font
   yourfont ← FontWindow new newfont: 16 maxcharwidth: 16 min: 0
         max: 177 ascent: 12 kern: 0.

**to edit newly created font
   yourtextstyle setfont: n name: 'yourfont.**insert it into a TextStyle
   **now create a window as above with yourtextstyle and appropriate fontnumber


**examples of manual manipulation of yourfontwindow:
   yourfontwindow setascent: 2.**Deltas -- for entire font**
   yourfontwindow setascent: ¯3.
   yourfontwindow setdescent: 2.
   yourfontwindow setdescent: ¯2.
   yourfontwindow setchar: 046.
   yourfontwindow setwidth: 5. **Absolute--for char in window.  Useful for characters of zero width.**
"]
init
   [fontmenu ← Menu new string:
'ascent
descent
width
frame']
lasttime                    "upon exit"
   []
newfont: fontht maxcharwidth: maxcharwidth min: min max: max ascent: ascent kern:
kern
   | raster i x
   [
   XeqCursor showwhile:
   [
   raster ← (2 + max - min * maxcharwidth + 15)/16.
   font ← String new:  (3 + max - min + (fontht * raster) + 9 * 2).
   font word: 1 ← 0100000.                                  "format: strike, simple, varwidth"
   font word: 2 ← min.                                      "min ascii code"
   font word: 3 ← max.                                      "max ascii code"
   font word: 4 ← maxcharwidth.                             "max char width"
   font word: 5 ← (2+max-min + 5 + (fontht*raster)).        "segment length"
   font word: 6 ← ascent.                                   "bits above baseline"
   font word: 7 ← fontht-ascent.                            "bits below baseline"
   font word: 8 ← kern.                                     "kerning offset"
   font word: 9 ← raster.                                   "#words per scan-line in bitmap"

   (font'((18 + 1) to: 2 * raster * fontht + 18)) all ← 0.  "chars all white"

   ascent ← ascent min: (fontht-1).                         "keep baseline within char"
   (font'(2 * raster * ascent + 18 + 1 to:
         ascent+1*raster*2 + 18)) all ← 0377.               "put in a black baseline"

   x ← 0.
   for: i  from: (raster * fontht + 9 + 1 to:
                  raster * fontht + 9 + 3 + max - min by: 1) do:
      [font word: i ← x. x ← x+maxcharwidth].               "table of left x"
   ]
   ↑font.
   ]
setascent: ascentdelta | updatedfont ascent
   [                              "ascent delta"
   ascent ← font word: 6.
   [ascent + ascentdelta < 0 ⇒[ascentdelta ← 0 - ascent]].
   [ascentdelta > 0 ⇒
      [
```

```
        updatedfont ← String new: (2 * fontraster * ascentdelta). "grow"
        updatedfont all ← 0.                                      "fill with white"
        updatedfont ←                                             "add oldfont header and new space
together"
                    (font'(1 to: 18) concat: updatedfont'(1 to: updatedfont length)).
        updatedfont ← (updatedfont concat: font'(19 to: font length))."now add on rest of old font"
        ]
    updatedfont ← (font'(1 to: 18) concat:                        "shrink"
                    font'((19 + (0 - (2 * fontraster * ascentdelta))) to: font length)).
    ].
    updatedfont word: 6 ← ascent + ascentdelta.                   "reset ascent word in font"
    self setfont: updatedfont.                                    "updatedfont now font of interest"
    ]
setbit: bitpoint color: color       "turn bits on, off"
                        | x y
    [
    bitpoint ← bitpoint - frame origin.
     x ← (0 max: (charwid-1)) min: (bitpoint x/scale).
     y ← (0 max: (fontht-1)) min: (bitpoint y/scale).
     boxer moveto: frame origin + ((scale*x) @ (scale*y)).
     boxer color: color mode: storing.                "turn bit on/off in blowup"

    bitsetter destraster ← fontraster.                "set up bitblt table."
    bitsetter destx ← charx + x.
    bitsetter desty ← y.
    bitsetter destbase ← (font lock) + 9.             "lock font and get core ptr"
    bitsetter fill: storing color: color.             "turn bit on/off in font"
    font unlock.                                      "release font"
    ]
setchar: char
    [
    charstr'1 ← char.
    [((font word: 2) ≤ char) and: (char ≤ (font word: 3))⇒
    [char ← char - (font word: 2)]
    char ← ((font word: 3) - (font word: 2)) + 1]."char out of range"
    charx ← (font word: (fontxtabl+ (char))).
    charwid ← (font word: (fontxtabl + char+1)) - charx.
    clearframe clear.
    frame extent ← charwid @ fontht.
    clearframe ←
        frame inset: ¯2 @ ¯2                  "for clearing everything including outline"
        and: (charwid - (charwid * scale + 2)) @ (fontht - (fontht * scale + 2)).
    self show.
    ]
setdescent: descentdelta | updatedfont descent space
    [                          "descent delta"
    descent ← font word: 7.
    [descent + descentdelta < 0 ⇒ [ descentdelta ← 0 - descent]].
    [descentdelta > 0 ⇒
          [space ← String new: 2 * fontraster * descentdelta.
          space all ← 0.
          updatedfont ← (font ' (1 to: fontxtabl - 1 * 2) concat: space).
          updatedfont ← (self appendxtable: updatedfont).
          ]
    updatedfont ←
        (font ' (1 to: ((fontxtabl - 1 * 2) + (fontraster * descentdelta * 2)))).
    updatedfont ← (self appendxtable: updatedfont).
    ].
    updatedfont word: 7 ← descent + descentdelta.          "reset descent word in font"
    self setfont: updatedfont.                             "updatedfont now font of interest"
    ]
setfont: font
    [
    altostyle fonts ' fontnumber ← font.
    fontraster ← font word: 9.
    fontht ← (font word: 6) + ( font word: 7).    "ascent + descent"
    fontxtabl ← fontraster * fontht + 9 "header" + 1 "for 0 addressing".
    bitsetter width ← 1. bitsetter height ← 1.
    self setchar: char.
    ]
setwidth | newextentx outlineframe
```

```
    [                    "get new size"
    outlineframe ← clearframe inset: 1 @ 1 and: 0 @ 1.
    OriginCursor showwhile:
        [user waitbug⇒
            [while: user anybug do:
                [outlineframe growto:
                ((clearframe origin x + 2) +
                        (newextentx ← (user mp x - clearframe origin x + 2) | scale))
                    @ (outlineframe corner y).
                outlineframe border: 2 color: black.
                outlineframe border: 2 color: backround
                ].
            ].
        ].
        outlineframe border: 2 color: black.
        self setwidth: newextentx / scale.
    ]
setwidth: delta | fontrightx newraster newxtabl newmaxwidth updatedfont i
    [                    "change in width"
    delta ← delta - charwid. delta = 0 ⇒ [self show. ↑false].
    fontrightx ← font word: (fontxtabl + ((font word: 3) - (font word: 2)) + 2).
    newraster ←
        [(fontrightx + 15 / 16) ≠ (i ← (fontrightx + delta + 15 / 16)) ⇒ [ i ] fontraster].
    newxtabl ← newraster * fontht + 9 "header" + 1 "for 0 addressing".

    XeqCursor showwhile:
    [
    updatedfont ← String new:
                (9 "header" + (newraster * fontht "bits")) * 2.  "grow/shrink the bits"
    for: i to: 8 do: [updatedfont word: i ← font word: i].       "fill in header of new font"
    updatedfont word: 9 ← newraster.                             "set raster in new font"
    "copy the xtable"
    updatedfont ← (self appendxtable: updatedfont).

    "set up to copy up to old bits of char"
    bitsetter destraster ← newraster.
    bitsetter destx ← 0. bitsetter desty ← 0.
    bitsetter sourcex ← 0. bitsetter sourcey ← 0.
    bitsetter width ← charx + charwid.
    bitsetter height ← fontht.
    bitsetter sourceraster ← fontraster.
    bitsetter destbase ← (updatedfont lock) + 9.
    bitsetter sourcebase ← (font lock) + 9.
    bitsetter copy: storing.
        [                                                        "if char grown, clean out right side
of char"
        delta< 0 ⇒[]
        bitsetter destx ← charx + charwid.
        bitsetter width ← delta.
        bitsetter fill: storing color: 0.
        ].
                                                                 "now copy remainder of font"
    bitsetter destx ← charx + charwid + delta.
    bitsetter width ← fontrightx - charx - charwid.
    bitsetter sourcex ← charx + charwid.
    bitsetter copy: storing.
    updatedfont unlock. font unlock.

    for: i from: ((char + 1)                                     "shift x-vals"
            to: (2 + (updatedfont word: 3 "max")) by: 1) do:
        [updatedfont word: (newxtabl+i) ← delta + (updatedfont word: (newxtabl+i))].
    clearframe clear.                                            "clear out old version of character"
    self setfont: updatedfont.                                   "set up the new copy of the font"
    ].
    ]
show |                              "refresh window"
                tempframe showrun showpara
    [
    showrun ← String new: 2.
    showrun word: 1 ←  16 * (fontnumber-1) + 0177400.
    showpara ← Paragraph new text: charstr runs: showrun alignment: 0.
```

```
    tempframe ← Textframe new para: showpara frame: frame.
    tempframe show.
    frame blowup: (frame origin) by: scale.
    ]
updatemaxwidth | newmaxwidth i
    [                                   "update max width"
    newmaxwidth ← 0.
    for§ i from: (fontxtabl to: ((font word: 3) - (font word: 2) + 1) by: 1) do§
            [newmaxwidth ← (newmaxwidth max: ((font word: i+1) - (font word: i)))].
    font word: 4 ← newmaxwidth.
    ]
updateseglength: newfont raster: newraster
    [                           "compute new segment length for a font"
    newfont word: 5 ← (5                                    "length, ascent, descent, kern, and raster"
                        + (newraster * fontht)              "bits"
                        + ((font word: 3 "max") -
                              (font word: 2"min") + 2)      "xtabl"
                      ).
    ]
!
```

"Paragraph"

```
Class new title: 'Paragraph';
   subclassof: Array;
   fields: 'text runs alignment';
   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 [↑Paragraph new text: text runs: runs alignment: alignment]
text: text
text: text alignment: alignment
text: text runs: runs alignment: alignment
```

Normal access

```
•x [↑text•x]
asParagraph [↑self]
asVector [↑text asVector]
changestyle: a to: b to: c      "Set font or toggle emphasis from char a to char b"
   [runs ← self runcat: (self runcat: (self run: 1 to: a-1)
      to: (self mergestyle: c into: (self run: a to: b)))
      to: (self run: b+1 to: text length)]
copy: a to: b      "Return a copied subrange of this paragraph"
   [↑Paragraph new
         text: (text copy: a to: b)
         runs: (self run: a to: b)
         alignment: alignment]
length [↑text length]
replace: a to: b by: c | t   "alters self - doesnt copy"
   [t ← text length.
   text ← text replace: a to: b by: [c is: String⇒[c] c text].
   runs@nil⇒[]
   runs ← self runcat: (self runcat: (self run: 1 to: a-1)
      to: [c is: String⇒[self makerun: c length
                        val: runs•((self runfind: b)•1+1)]
         c runs])
      to: (self run: b+1 to: t)]
subst: x for: y "runs are not supported yet here"
   [↑text subst: x for: y]
text [↑text]
```

Text alignment

```
alignment [↑alignment]
alignment ← alignment
center [alignment ← 2]
flushleft [alignment ← 0]
flushright [alignment ← 4]
justify [alignment ← 1]
```

Manipulation of format runs

```
makerun: len val: val      "Make up a solid run of value val"
      | str i
   [len=0⇒[↑nullString]
   str ← String new: len-1/255+1*2.
   for§ i from: (1 to: str length by: 2) do§
      [str•i ← [len>255⇒[255] len]. str•(i+1) ← val.
      len ← len-255]
   ↑str]
```

```
maskrun: i to: j under: m to: val "Alter my runs so that the bits selected by m become val."
        | r k        "Maybe merge this with mergestyle"
    [r ← self run: i to: j.
    for& k from: (2 to: r length by: 2) do&
        [r*k ← (r*k land: 0377-m) + val].
    runs ← self runcat: (self runcat: (self run: 1 to: i-1) to: r) to: (self run: j+1 to: text length)]
mergestyle: a into: b | c        "a=new style value, b=run String"
    [for& c from: (2 to: b length by: 2) do&
        [b*c ← [a=¯1⇒ [0]        "reset"
                0<a and: a≤017⇒ [a xor: b*c]    "toggle emphasis"
                a+(017 land: b*c)]]."new font"
    ↑b]
run: a to: b | c      "subrange of run"
    [a>b⇒[↑nullString]
    runs@nil⇒[↑self makerun: 1+b-a val: 0]
    a ← self runfind: a.
    b ← self runfind: b.
    c ← runs copy: a*1 to: b*1+1.    "copy the sub-run"
    (a*1)=(b*1)⇒
        [c*1 ← 1+ (b*2)-(a*2). ↑c]
    c*1 ← 1+(runs*(a*1))- (a*2).        "trim the end lengths"
    c*(c length-1) ← b*2.  ↑c]
runcat: x to: y     "concatenate two runs"
    [x length=0⇒[↑y] y length=0⇒[↑x]
    (x*x length)=(y*2)⇒
            [↑x*(1 to: x length-2) concat:
                ((self makerun: (x*(x length-1))+(y*1) val: y*2)
                concat: y*(3 to: y length))]
    ↑x + y]
runfind: index | run       "index into run"
    [run←1.
    while& runs*run<index do&
        [index ← index-(runs*run). run ← run+2]
    ↑run,index]
runs       "return runs or default if none"
    [runs@nil⇒[↑self makerun: text length val: 0]
    ↑runs]
```

Bravo conversions

```
applyBravo: trailer at: i to: j "Alter runs of characters i through j according to trailer"
        | s len ch t
    [s ← Stream new of: trailer.
        [(s upto: 0134) length>30⇒[user show: 'Suspicious Bravo trailer'; cr]].
    len ← 0.
    until& s end do&
        [(ch ← s next) isdigit ⇒ [len ← (len*10)+ch-060]
        i ← i+len. len ← 0.
        ch=0146 ⇒
            [t ← 0. while& (ch ← s next) isdigit do& [t ← (t*10)+ch-060].
            self maskrun: i to: j under: 0360 to: (t lshift: 4)
            ]
        (t ← 'bBiIuU' find: ch)=0 ⇒ []
        self maskrun: i to: j under: ☞(1 1 2 2 4 4)*t to: ☞(1 0 2 0 4 0)*t
        ]
    ]
fromBravo "Find Bravo trailers and return a copy of self with them applied"
        | newpara newtext loc i j
    [newpara ← self copy.
    loc ← 1.
    while& (i ← (newtext ← newpara text) find: 032)≠0 do&
        [j ← newtext*(i+1 to: newtext length) find: 015.
        newpara applyBravo: newtext*(i+1 to: i+j) at: loc to: i-1.
        newpara replace: i to: i+j-1 by: ''.
        loc ← i+1]        "Final CR not yet handled"
    ↑newpara]
toBravo "Encode the runs in a Bravo paragraph trailer"
        | i s old len dif new bit bits
    [bits ← ☞(1 2 4).
    s ← Stream default.
    s next← 032. s append: '\g'.
```

```
    len ← 0. old ← 0400.
    [runs@nil ⇒ []
    for⁸ i from: (1 to: runs length by: 2) do⁸
        [dif ← old xor: (new ← runs•(i+1)).
        (dif land: 0367)=0 ⇒ "No changes" [len ← len+(runs•i)]
        [i=1⇒[] len printon: s].
        for⁸ bit to: 3 do⁸
            [(dif land: bits•bit)=0 ⇒ []
            (new land: bits•bit)≠0 ⇒ [s next← 'biu'•bit]
            s next← 'BIU'•bit]
        [(dif land: 0360)≠0 ⇒ "Font change"
            [s append: 'f'; print: (new lshift: ¯4); space]].
        old ← new.
        len ← runs•i.
        ]
    ].
    s next← 015.
    ⇑(text concat: s contents) asParagraph]
```

Packing into String

```
packIntoString | s          "pack alignment, runs and text into a String"
    [runs @ nil⇒[⇑text]          "no runs - dont pack"
    s ← (String new: 4)+runs+text.
    s•1 ← 032. s•2 ← alignment.      "these 2 bytes signify packing"
    s word: 2 ← runs length. ⇑s]
unpackFromString: s | l          "undo packIntoString"
    [s is: Paragraph⇒[⇑s copy]
    s length<4⇒[alignment ← 0. text ← s]      "not packed format"
    s•1≠032 or: s•2>4⇒[alignment ← 0. text ← s]
    alignment ← s•2.
    l ← s word: 2.
    runs ← s copy: 5 to: 5+l-1.
    text ← s copy: 5+l to: s length]
```

As yet unclassified

```
makeBoldPattern | s
    [s ← text asStream upto: '['•1.
    s ← s asStream upto: ']'•1.
    s ← s asStream upto: '"'•1.
    self maskrun: 1 to: s length under: 1 to: 1.
    text•(text length-1)=015⇒
        [self replace: text length-1 to: text length-1 by: nullString]
    ]
!
```

"ParagraphEditor"

```
Class new title: 'ParagraphEditor';
   subclassof: Textframe;
   fields: 'loc1 loc2';
   declare: 'ctlchars Scrap p1 p2 bs paste runvals oldheight ctlw
typein esc oldpara cut Deletion ';
   asFollows!
```

ParagraphEditors handle selecting, typing and replacing of text. Some work remains, such as making p1, p2, oldpara and oldheight instance variables.

Scheduling

```
enter
   [oldpara ← oldheight ← false.
   typein ← false.
   self show. self select]
leave
   [self comp: p1 to: p2]
```

Public Messages

```
contents [↑para]
copy [Scrap ← self selection]
cut [self fintype; replace: nullString; select.
   Scrap ← Deletion]
fixframe: f | dy
   [dy ← [frame@nil⇒[0] frame origin y - window origin y].
   window ← f copy.
   frame ← Rectangle new
      origin: (window origin x+2) ☺ (window origin y + dy)
      extent: (window extent x-4) ☺ 9999.
   ↑window]
frame ← f
   ["Acquire a window of f or slightly smaller, and a frame inset slighly from that."
   self fixframe: f]
keyset | k
   [user keyset=1⇒[self cut. user waitnokey];
            =16⇒[self paste. user waitnokey];
            =0⇒[]
   self unselect.
   while☹ (k← user keyset)≠0 do☹
      [self scrollby:
         [k allmask: 2⇒[1] k allmask: 8⇒[¯1] 0] * [k allmask: 4⇒[4] 1]]
   self select]
kludge
   [  [loc1@nil⇒ [loc1←loc2←1]].
   p1 ← self ptofchar: loc1.
   p2 ← [loc2=loc1⇒[1☺0 + p1] self ptofchar: loc2].
   ]
oldpara
   [↑oldpara]
oldpara ← oldpara
outline
   [window outline]
paste [self replace: Scrap; select]
realign [para alignment ← ☞(1 2 4 0 0)·(1+para alignment).
   self show]
Scrap ← s [Scrap ← s]
scrollby: n | l w
   [n ← n max: (frame origin y-4-window origin y)/(l← self lineheight).
   n ← n*l.
   frame moveby: 0☺(0-n).
   w ← [n<0⇒[window inset: 0☺0 and: 0☺(0-n)]
         window inset: 0☺n and: 0☺0].
   w blt: w origin-(0☺n) mode: storing.
      [n<0⇒[w corner y ← w origin y - n]
         w origin y ← w corner y - n].
```

```
    self showin: w]
scrollUp: n | t
    [t← n/self lineheight. self scrollby: [n>0⇒[t+1] t-1]]
selecting        " Track and complement redbug selection "
    | a t drag2
    [t ← self charnearpt: user mp.
    self comp: p1 to: p2. self fintype.
    [t=loc1⇒[loc1=loc2⇒[loc1≠1⇒            " double-bug "
        [self selectword. ⇑self select] ]]].
    drag2 ← true. loc1 ← loc2 ← t.    " start new selection "
    p2 ← 1@0 + (p1 ← reply1).
    self comp: p1 to: p2.
    while§ user redbug do§            " draw out selection "
        [a ← reply1.
            [loc1=loc2⇒[drag2 ← t≥loc2]].
            [drag2⇒ [ [t<loc1⇒[t←loc1. a←self ptofchar: t]].
                    self comp: a to: p2. loc2 ← t. p2 ← a]
                [t>loc2⇒[t←loc2. a←self ptofchar: t]].
                self comp: p1 to: a. loc1 ← t. p1 ← a].
            [p1=p2⇒[self comp: p1 to: (p2 ← 1@0 + p1)]].
        t ← self charnearpt: user mp]]
selection
    [⇑para copy: loc1 to: loc2-1]
selectword        "Select bracketed or word range for double-click"
    | a b dir t level open close s
    [a← b← dir← ¯1.
    open ← '([{< ""
'.
    close ← ')]}> ""
'.
        [loc1>para length⇒[t←loc1-1]
        loc1≤1⇒[dir←1. t←loc1]
        t←open find: (a←para•(loc1-1)). t>0⇒ "delim on left"
            [dir←1. b←close•t. t←loc1-1]    "match to the right"
        t←close find: (a←para•loc1). t>0⇒    "delim on right"
            [dir←¯1. b←open•t. t←loc1]      "match to the left"
        a← ¯1. t←loc1].                  "no delims - select a token"
    level←1. s ← para text.
    until§ [dir=1⇒[t≥s length⇒[] level=0⇒[] false]
            t≤1⇒[] level=0⇒[] false] do§
        [s•(t← t+dir) = b⇒ [level← level-1];    "leaving nest"
        = a⇒ [level← level+1].                  "entering nest"
        a=¯1⇒[(s•t) tokenish⇒                   "token check goes left "
                [t=1⇒[loc1← dir← 1. t← loc2]]
            dir=¯1⇒[loc1 ← t+1. dir←1. t←loc2]"then right"
            level← 0]]
    [level≠0⇒[t← t+dir]].
    dir=1⇒[loc2← t] loc1← t+1]
typing | more char
    [[loc1<loc2⇒[self checklooks⇒[self show. ⇑self select]]].
    more ← Stream default.
    [typein⇒[] typein ← loc1].
    while§ user kbck do§
        [(char ← user kbd)
        =bs⇒ [more empty⇒[loc1 ← 1 max: loc1-1. typein ← typein min: loc1]
                more skip: ¯1];                  "backspace"
        =ctlw⇒ [more reset. loc1 ← 1 max: loc1-1."ctl-w for backspace word"
                while§ [loc1>1⇒[(para•(loc1-1)) tokenish] false] do§ [loc1 ← loc1-1]
                typein ← typein min: loc1];
        =cut⇒ [⇑self cut];
        =paste⇒ [⇑self paste];
        =esc⇒ [self replace: more contents; fintype. "select previous type-in"
                loc1 ← loc2-Scrap length. ⇑self select]
        more next← char].
    self replace: more contents. loc1 ← loc2.
    user kbck⇒[] self select]
undo
    [self fintype; replace: Deletion; select]
unselect [p2 ← p1]
```

Private Messages

```
checklooks | t
   [(t ← user rawkbck)<0253⇒[⇑false];
      =0334⇒[user rawkbd. self toBravo];
      =0337⇒[user rawkbd. self fromBravo].
   loc1≥loc2 ⇒ [⇑false]
   (t ← ctlchars find: t)=0⇒[⇑false]
   user rawkbd.
   para changestyle: loc1 to: loc2-1 to: runvals*t]
comp: a to: b [self comp: a to: b with: black]
comp: a to: b with: color | t        "complement from a to b"
   [  [a y<b y⇒[] (a y=b y) and: (a x<b x)⇒[] t←a. a←b. b←t]."force a < b"
      [a y < b y⇒
         [((a rect: (window corner x-4) ⊚ (a y+self lineheight))
               intersect: window) comp: color.     "top line"
          a ← (window origin x+2) ⊚ (a y+self lineheight).
          a y < b y⇒
            [((a rect: (window corner x-4) ⊚ b y)
               intersect: window) comp: color]    "middle (if any)"
         ]].
      a y > b y⇒[]
      ((a x ⊚ b y rect: b x ⊚ (b y + self lineheight))
               intersect: window) comp: color"bottom (or only)"]
fintype
   [typein⇒
      [  [typein<loc1⇒
            [Scrap ← para copy: typein to: loc1-1.
            loc1 ← typein]].
      typein ← false]
   ⇑false].

fromBravo
   [para ← para fromBravo. loc1 ← nil]
init | i
   [bs ← 8. ctlw ← 23. esc ← 27.
   cut ← 127. paste ← 2.
   ctlchars ← ☞(0342 0351 0255 0256 0362
      0260 0261 0262 0263 0264 0265 0266 0267 0270 0271
      0341 0342 0343 0344 0345 0346).
   runvals ← ☞(1 2 4 8 ¯1  "b i - ¯ r"
      0 1 2 3 4 5 6 7 8 9  "0 1 ... 9"
      10 11 12 13 14 15).  "a b ... f"
   for⊚ i to: 16 do⊚ [runvals*(5+i) ← runvals*(5+i)*16]]
replace: t
   [  [oldpara⇒[] oldpara ← para copy].
      [loc1<loc2⇒[Deletion ← para copy: loc1 to: loc2-1]].
   para ← para replace: loc1 to: loc2-1 by: t.
   loc2 ← loc1 + t length.
   self show]
select
   [  [loc1@nil⇒ [loc1←loc2←1]].
   p1 ← self ptofchar: loc1.
   p2 ← [loc2=loc1⇒[1⊚0 + p1] self ptofchar: loc2].
   self comp: p1 to: p2]
toBravo
   [para ← para toBravo]
!
```

ctr period

ctr b ctr i ctr -

"Textframe"

Class new title: 'Textframe';
fields: 'frame para style reply1 reply2 window';
asFollows!

This class has not yet been commented

As yet unclassified

```
aboutToFrame
    ["My frame is about to change.  I dont care."]
charnearpt: pt [user croak] primitive: 58
charofpt: pt [user croak] primitive: 58
comp
    [window comp]
frame [↑frame]
frame ← frame
    ["Change my frame and window."
    window ← frame.
    ]
lastshown
    [↑reply1]
lineheight
    [↑style lineheight]
para: para frame: frame
[window ← frame.
    reply1 ← reply1 ← 0.
    style ← DefaultTextStyle]
printon: strm
    [strm append: 'a Textframe']
ptofchar: char
    [self selectchar: char. ↑reply1]
ptofpt: pt
    [self charnearpt: pt. ↑reply1]
put: para at: pt
    [self put: para at: pt centered: false]
put: para at: pt centered: center
    [para ← para asParagraph.
    window ← frame ← pt rect: 1000@1000.
    self ptofchar: para length+1.
    window growto: reply2.
    [center→ [window moveby: pt-window center]].
    window ← window inset: ¯3@¯2.
    window clear: white. self show]
put: para centered: pt
    [self put: para at: pt centered: true]
rectofchar: char
    [self selectchar: char. ↑reply1 rect: reply2]
scrolln: n
    [↑self charofpt: frame corner x @ (frame origin y+(n*style lineheight))]
selectchar: char
    [self selectchar: char asInteger] primitive: 59
show [user croak] primitive: 57
show: para
    [para ← para asParagraph. self show]
showin: rect | old      "show clipped inside rect"
    [old ← window. window ← rect. self show. window ← old]
takeCursor
    ["Move the cursor to the center of my window."
    user cursorloc ← window center]
window [↑window]
!
```

"TextStyle"

Class new title: 'TextStyle';
fields: 'fonts "<Vector of Strings or Integers> which are the fonts.
An integer entry has a vertical offset in the high 8 bits, a 1 in
the 200-bit for descent, and another font number (zero-relative)
in the bottom 4 bits"
tabandspace "<Integer> =256*tabwidth + spacewidth"
maxascent "<Integer> max ascent for this fontset"
maxdescent "<Integer> max descent for this fontset"
mode "<Integer> =0 for normal, =4 for white-on-black"
fontnames "<Vector of Strings> corresponding to the fonts"';
asFollows!

This class has not yet been commented

As yet unclassified

```
default
    [self mode: 0; tab: 20; space: 5]
fonts [↑fonts]
lineheight
    [↑maxascent+maxdescent]
mode: mode
setfont: n name: name | f  "should update max-a/de-scent"
    [FontDict has: name⇒
        [fontnames•(n+1) ← name.
        fonts•(n+1) ← FontDict•name]
    (f ← File new old named: name + '.strike.')⇒
        [FontDict insert: name with: f contents.
        self setfont: n name: name]
    user notify: 'Font ' + name + '.strike. not on this disk']
setoffsetfont: n from: m by: d
    [fonts•n ← m + [d<0⇒ [0200] 0] + (d lshift: 8)]
space: t
    [tabandspace ← (tabandspace land: 0177400) + (t land: 0377)]
tab: t
    [tabandspace ← (tabandspace land: 0377) + (t lshift: 8)]
!
```

"browser and debugger"
"version 3.5 6 Sept 77"

"ClassPane"

```
Class new title: 'ClassPane';
   subclassof: ListPane;
   fields: 'systemPane organizationPane';
   declare: 'editmenu ';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
close
   [systemPane ← nil. super close]
compile: parag
   [systemPane compile: parag]
deselected
   ["I just lost my selection. Tell organizationPane to display nothing."
   organizationPane class: nil.]
dirty
   [↑organizationPane dirty]
from: systemPane to: organizationPane
   ["Acquire the specified relationships. Initialize editmenu."
   editmenu@nil⇒ [editmenu ← Menu new string: 'filout
forget']]
noCode
   [selection=0⇒ [↑systemPane noCode] ↑"]
selected
   ["My selection just changed. Tell organizationPane to display the categories of my newly
selected Class."
   organizationPane class: Smalltalk•(list•selection).]
yellowbug
   ["If there is a selection, let the user choose a command from the menu."
   selection=0⇒ [window flash]
   editmenu bug
      =1⇒ ["filout the selected class" (Smalltalk•(list•selection)) filout];
      =2⇒ ["forget the selected class" systemPane forget: list•selection]]
!
```

"CodeEditor"

```
Class new title: 'CodeEditor';
   subclassof: Window;
   fields: 'pared class selector';
   declare: 'editmenu ';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
aboutToFrame
   ["My frame is about to change.  Tell pared to unselect."
   pared unselect]
class: class selector: selector para: para
   [pared ← ParagraphEditor new para: para frame: nil.
   self newframe]
close [pared unselect]
enter [super show. pared enter]
fixframe: f [↑pared fixframe: f]
init
   [editmenu ← Menu new string:
'undo
copy
cut
paste
doit
compile
align']
kbd [pared typing]
keyset [↑pared keyset]
leave [pared leave]
outside
   [↑(ScrollBar new on: frame from: pared) startup]
redbug [pared selecting]
show [super show. pared show]
title [↑class title+' . '+selector]
yellowbug
   [editmenu bug
      =1⇒[pared undo];
      =2⇒[pared copy];
      =3⇒[pared cut];
      =4⇒[pared paste];
      =5⇒[pared Scrap ← XeqCursor showwhile
            (nil'spared selection) asString asParagraph];
      =6⇒[selector=☞ClassOrganization⇒
               [class organization fromParagraph: pared contents]
            class understands: pared contents];
      =7⇒[pared realign]]

!
```

"CodePane"

```
Class new title: 'CodePane';
   subclassof: CodeEditor;
   fields: 'selectorPane oldpara';
   declare: 'editmenu ';
   asFollows!
```

I am a CodeEditor without a titleframe or a class. My selectorPane compiles and doits for me. If oldpara@nil, I am clean, else pared contents was originally oldpara.

As yet unclassified

```
close
   [selectorPane ← nil. super close]
contents
   [⇑pared contents]
dirty
   [oldpara@false⇒ [⇑false] ⇑frame]
eachtime        "like window code, but leaves without bug"
   [frame has: user mp⇒
      [user kbck⇒[⇑self kbd]
      user anybug⇒
         [user redbug⇒[⇑self redbug]
         user yellowbug⇒[⇑self yellowbug]
         user bluebug⇒[⇑false]]
      user anykeys⇒[⇑self keyset]]
   self outside⇒[]
   ⇑false]
edit: expr "It would be better for pared oldpara to be a field"
   [pared oldpara ← oldpara. expr eval. oldpara ← pared oldpara]
enter
   [frame flash]
frame ← frame
   ["Change my frame and that of my pared (if any)."
   pared@nil⇒ [] pared frame ← frame]
from: selectorPane
init
   [editmenu@nil⇒ [editmenu ← Menu new string:
      'undo
copy
cut
paste
doit
compile
cancel'.]]
kbd
   [self edit: super kbd]
keyset
   [self edit: super keyset]
picked
   [⇑frame has: user mp]
redbug
   [pared kludge. super redbug]
show
   [frame outline. pared show]
showing: paragraph
   [pared ← ParagraphEditor new para: paragraph asParagraph frame: nil.
   oldpara ← false. pared fixframe: frame. self windowenter]
windowenter
   [pared outline; enter]
windowleave
yellowbug
   [editmenu bug
      =1⇒[self edit: pared undo];
      =2⇒[self edit: pared copy];
      =3⇒[self edit: pared cut];
      =4⇒[self edit: pared paste];
      =5⇒[pared Scrap ← XeqCursor showwhile:
            (selectorPane execute: pared selection) asString asParagraph];
```

```
=6⇒[oldpara⇒ [selectorPane compile: pared contents. oldpara ← false] frame flash];
=7⇒[oldpara⇒ [self showing: oldpara] frame flash]]!
```

"ListPane"

```
Class new title: 'ListPane';
   subclassof: Textframe;
   fields: 'list firstShown lastShown selection';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
close "Zero my selection so it wont be grayed when I close."
   [selection←0]
compselection "If I have a selection, complement its image."
   [selection≠0⇒ [self selectionRect comp]]
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]
dummy
   [⇑'.....']
eachtime
   [window has: user mp⇒
      [user kbck⇒[⇑self kbd]
      user anybug⇒
         [user redbug⇒[⇑self redbug]
         user yellowbug⇒[⇑self yellowbug]
         user bluebug⇒[⇑false]]
      user anykeys⇒[⇑self keyset]]
   self outside⇒[]
   ⇑false]
enter "Refresh my image.  Reaffirm selection."
   [window flash]
fill | dY i len s "Given firstShown, compute lastShown and show me."
   [
   dY ← self lineheight. len ← list length.
   lastShown ← firstShown-1 + (window extent y-4/dY) min: len+1.
   [self locked⇒
      [i ← (selection-lastShown max: 0) + (selection-firstShown min: 0).
      i≠0⇒ [para←nil. firstShown ← firstShown + i. lastShown ← lastShown + i]]].
   (frame ← window inset: 2) width ← 999.
   [para@nil⇒ "If para is not nil, refresh from it, else compute para."
      [s ← (String new: 200) asStream.
      for§ i from: (firstShown to: lastShown) do§
         [[0<i and: i≤len⇒ [(list˙i) printon: s] self dummy copyto: s].
         s cr].
      para ← s contents
      ]].
   self show]
firsttime
   [window has: user mp⇒[self enter]
   ⇑false]
frame ← window "(Re)initialize my window"
   [para ← nil]
grayselection
   [selection≠0⇒ [self selectionRect color: ltgray mode: oring]]
init
   [self para: nil frame: nil.]
kbd
   [window flash. user kbd.]
keyset | c
   ["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."
   c ← Cursor new frompage1.
   self scrollControl§ [user keyset=6⇒[2]; =12⇒[¯2]; =2⇒[1]; =8⇒[¯1] 0].
   c topage1]
lasttime
leave
   [self compselection; grayselection]
locked "My subclasses may want to prohibit a change of selection"
   [⇑[selection=0⇒ [false] self dirty]]
```

```
of: list "Acquire the specified list and show me scrolled to the top"
    [firstShown ← selection ← 0. para ← nil. self fill; deselected]
outline
    [window outline.]
outside [⇑(ScrollBar new on: window from: self) startup]
picked
    [⇑window has: user mp]
redbug | newSel f"Deselect selection and select cursor item, if any"
    [self compselection. f ← self locked⇒ [f flash. self compselection]
    newSel ← (user mp y - window origin y)/self lineheight + firstShown.
    self select: [newSel = selection⇒ [0] newSel]]
revise: list with: sel "Acquire list. Do not change firstShown. Select sel if in list."
    [firstShown ← firstShown min: list length.
    para ← nil. self fill. selection ← ¯1. self select: (list find: sel)]
scrollBy: expr copying: src into: dest showing: item in: frame direction: n
        | strm final stop pt delay chars locked t
    [strm ← Stream new. chars ← 2*frame width/self lineheight. para ← String new: chars.
    pt ← dest origin. final ← [n<0⇒ [0] list length+1].
    stop ← [locked←self locked⇒ [0 max: (list length+1 min: (lastShown - firstShown * n sign +
selection))] final].
    while: item≠stop do:
        [firstShown ← firstShown + n. lastShown ← lastShown + n. item ← item + n.
        strm of: para from: 1 to: chars.
        [item≠final⇒ [(list˙item) printon: strm] self dummy copyto: strm].
        strm cr. src blt: pt mode: storing. self show.
        (t← expr eval) abs ≤1⇒ [for: delay to: chars/4 do: [strm myend]. para ← nil. ⇑false]
            t*n<0⇒[⇑false]].
    para ← nil. locked and: stop≠final⇒ [locked flash]]
scrollControl: expr
        | dY onlyFirst butFirst onlyLast butLast x1 x2 y1 y2 y3 y4 k
    ["Selection is highlighted. Unhighlight it. Invalidate my saved para if I scroll. Then reselect
selection, or deselect if it is no longer displayed."
    self compselection. dY ← self lineheight.
    x1 ← window origin x. x2 ← window corner x.
    y1 ← window origin y+2. y4 ← window height-4 |dY + y1. y2←y1+dY. y3←y4-dY.
    onlyFirst ← x1+2@y1 rect: 2000@y2. butFirst ← x1@y2 rect: x2@y4.
    onlyLast ← x1+2@y3 rect: 2000@y4. butLast ← x1@y1 rect: x2@y3.
    while: (k←expr eval)≠0 do:
        [k>0⇒[UpCursor topage1.
            self scrollBy: expr eval copying: butFirst into: butLast showing: lastShown
                in: onlyLast direction: 1]
        DownCursor topage1.
        self scrollBy: expr eval copying: butLast into: butFirst showing: firstShown
                in: onlyFirst direction: ¯1].
    self select: selection]
scrollUp: n | l c p
    [l ← self lineheight. c ← window center y-8.
    self scrollControl:
        [user buttons=4⇒
            [c>(p← user mp y)⇒[p-c/l-1] p-c/l+1]
        0]]
select: lineNum | oldSel
    ["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."
    oldSel ← selection.
    (1 max: firstShown) ≤ lineNum and: lineNum ≤ (list length min: lastShown)⇒
        [selection ← lineNum. self compselection. oldSel≠selection⇒ [self selected]]
    selection ← 0. oldSel≠selection⇒ [self deselected]]
selected "A new selection is highlighted. I dont care, but my subclasses might"
selectionRect | h w
    ["I have a selection. Return its highlighting rectangle."
    (w ← window inset: 2) height ← h ← self lineheight.
    ⇑w + (0@(selection-firstShown *h))]
windowenter "Refresh my image. Reaffirm selection."
    [window outline. self fill; select: selection.]
windowleave
    [self compselection; grayselection]
yellowbug
    [window flash]
!
```

"NotifyWindow"

```
Class new title: 'NotifyWindow';
   subclassof: PanedWindow;
   fields: 'enoughpanes';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
aboutToFrame
   [enoughpanes ← panes length = 6. super aboutToFrame]
enter | stackPane codePane contextVarPane contextValuePane instanceVarPane instanceValuePane
   [enoughpanes⇒ [super enter]
   NotifyFlag ← false.
   "Create the remaining five panes."
   stackPane ← panes 1. codePane ← CodePane init.
   contextVarPane ← VariablePane init. contextValuePane ← CodePane init.
   instanceVarPane ← VariablePane init. instanceValuePane ← CodePane init.
   "Create the six-paned window."
   self title: title
      with: (stackPane, codePane, contextVarPane, contextValuePane, instanceVarPane, instanceValuePane)
      at: NotifyPaneFrames.
   self frame: frame; show.
   "Initialize the six panes."
   stackPane from: self context: contextVarPane instance: instanceVarPane code: codePane.
   codePane from: stackPane.
   contextVarPane to: contextValuePane. contextValuePane from: contextVarPane.
   instanceVarPane to: instanceValuePane. instanceValuePane from: instanceVarPane.
   stackPane select: 0; deselected; fill. enoughpanes ← NotifyFlag ← true]
!
```

"OrganizationPane"

```
Class new title: 'OrganizationPane';
   subclassof: ListPane;
   fields: 'classPane selectorPane class';
   asFollows!
```

Uncommented

As yet unclassified

```
class: class
   [self of: [class@nil⇒ [Vector new: 0]
      ☞(ClassDefinition ClassOrganization) concat: class organization categories]]
close
   [classPane ← nil. super close]
code: selector
   [⇑class code: selector]
compile: parag
      | sel cat
   [class@nil⇒ [classPane compile: parag]
   selection=1⇒ [nil'sparag]; "new definition"
      =2⇒ [class organization fromParagraph: parag. self class: class] "new organization"
   cat ← [selection=0⇒ ['As yet unclassified'] list'selection].
   sel ← class understands: parag classified: cat.
   [selection=0⇒ [self revise: list with: cat]].
   selection≠0⇒ [selectorPane revise: (class organization category: cat) with: sel]]
deselected
   ["I just lost my selection. Tell selectorPane to display nothing."
   selectorPane of: (Vector new: 0)]
dirty
   [⇑selectorPane dirty]
execute: parag
   [⇑class's parag]
forget: selector
   [class derstands: selector.
   selectorPane revise: (class organization category: list'selection) with: selector]
from: classPane to: selectorPane
noCode
   [class@nil⇒ [⇑classPane noCode]
   selection=0⇒ [⇑'']; =1⇒ [⇑class definition]; =2⇒ [⇑class organization]
   ⇑'Message name and Arguments | Temporary variables "short comment"
   ["long comment if necessary"
   Smalltalk
   Statements]']
selected
   [selectorPane of: [selection≤2⇒ [Vector new: 0] class organization category: list'selection]]
spawn: selector with: parag
   [user schedule: (CodeEditor new class: class selector: selector para: parag).
   selectorPane compselection; select: 0]
!
```

"PanedWindow"

```
Class new title: 'PanedWindow';
   subclassof: Window;
   fields: 'panes templates title';
   asFollows!!
```

This class has not yet been commented

As yet unclassified

```
close | pane
   [for: pane from: panes do: [pane close]]
eachtime | pane
   [frame has: user mp⇒
      [user bluebug⇒[⇑self bluebug]
      for: pane from: panes do: [pane startup]]
   self outside⇒[]
   user anybug⇒[frame has: user mp⇒[] ⇑false]
   user kbck⇒[user kbd. frame flash] "flush typing outside"]
enter | pane
   [((self titlerect clear: black) inset: 2@2) clear: dkgray. super show.
   for: pane from: panes do: [pane windowenter]]
erase
   [self titlerect clear. super erase]
fixframe: f
   [⇑Rectangle new origin: f origin extent: (f extent max: 160@80)]
frame: frame "(Re)initialize my frame, and tell my panes their locations."
      | templateStream template pane
   [templateStream ← templates asStream.
   for: pane from: panes do:
      ["It would be nice to have parallel fors as in MLISP."
      template ← templateStream next.
      pane frame ← (template*frame extent /36 +frame origin inset: 2)]]
kbd | pane
   [(pane ← self pickedpane)⇒ [⇑pane kbd]]
keyset | pane
   [(pane ← self pickedpane)⇒ [⇑pane keyset]]
leave | pane
   [for: pane from: panes do: [pane windowleave]]
pickedpane | pane
   [for: pane from: panes do: [pane picked⇒ [⇑pane]]
   frame flash. ⇑false]
redbug | pane
   [(pane ← self pickedpane)⇒ [⇑pane redbug]]
show | pane
   [((self titlerect clear: black) inset: 2@2) clear: dkgray. super show.
   for: pane from: panes do: [pane outline]]
takeCursor
   [(panes*1) takeCursor]
title
   [⇑title]
title: title with: panes at: templates
titlerect
   [⇑frame origin - (2 @ (DefaultTextStyle lineheight + 4)) rect: (frame corner x@frame origin y) +
(2@0)]
vanish
   [self close; erase. user unschedule: self.]
yellowbug | pane
   [(pane ← self pickedpane)⇒ [⇑pane yellowbug]]
!
```

simplify by using eachtime ?

"ScrollBar"

Class new title: 'ScrollBar'; fields: 'rect bitstr owner'; asFollows!

This class has not yet been commented

As yet unclassified

```
eachtime | p c
   [rect has: (p← user mp)⇒
      [c ← rect center y-8.
      p y<c⇒[DownCursor showwhile:
               [while: [(rect has: (p←user mp)) and: p y<c] do:
                  [user redbug⇒[owner scrollUp: p y-c]]]]
      UpCursor showwhile:
               [while: [(rect has: (p←user mp)) and: p y≥c] do:
                  [user redbug⇒[owner scrollUp: p y-c]]]]
   ⇑false]
firsttime      "If mouse within, then save background and turn gray"
   [rect has: user mp⇒
      [bitstr ← rect bitsIntoString.
      rect clear: 010504. (rect inset: 0@0 and: 16@0) clear: black]
   ⇑false]
lasttime    "restore background"
   [rect bitsFromString: bitstr]
on: frame from: owner
   [rect ← frame copy. rect moveby: ¯20@0; width ← 20]
!
```

"SelectorPane"

```
Class new title: 'SelectorPane';
   subclassof: ListPane;
   fields: 'organizationPane codePane';
   declare: 'editmenu ';
   asFollows!
```

I am a ListPane whose entries are the message selectors of a category within a class. Only organizationPane knows what the class and category are. I make codePane display the code of my selected selector, if any.

As yet unclassified

```
close
   [organizationPane ← nil. super close]
compile: parag
   [organizationPane compile: parag]
deselected
   [codePane showing: organizationPane noCode]
dirty
   [↑codePane dirty]
execute: parag
   [↑organizationPane execute: parag]
from: organizationPane to: codePane
   [editmenu@nil⇒ [editmenu ← Menu new string:
      'spawn
forget']]
selected
   [codePane showing: (organizationPane code: list•selection)]
yellowbug
   [selection=0⇒ [window flash]
   editmenu bug
      =1⇒ [organizationPane spawn: list•selection with: codePane contents];
      =2⇒ [organizationPane forget: list•selection]]
!
```

"StackPane"

```
Class new title: 'StackPane';
   subclassof: ListPane;
   fields: 'contextVarPane instanceVarPane codePane
notifyWindow';
   declare: 'stackmenu ';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
close
   [super close. notifyWindow ← nil. list⇒ [(list'1) release]]
code "the code of my selected context, if it has code, else false"
      | mclass selector
   [mclass ← (list'selection) mclass. selector ← self selector.
   ⇑[mclass canunderstand: selector⇒ [mclass code: selector] false]]
compile: parag
   [selection=0⇒ [nil'sparag] (list'selection) mclass understands: parag]
deselected
   [codePane @ false⇒ []
   codePane showing: ''.
   contextVarPane names: (Vector new: 0) values: ☞(nil) wrt: nil.
   instanceVarPane names: (Vector new: 0) values: ☞(nil) wrt: nil]
dirty
   [⇑codePane and§ codePane dirty]
execute: parag
   [⇑[selection=0⇒ [nil'sparag] list'selection execute: parag]]
from: notifyWindow context: contextVarPane instance: instanceVarPane code: codePane
   [stackmenu@nil⇒ [stackmenu ← Menu new string:
      'stack
spawn
proceed']]
proceed
   ["Close notifyWindow and return nil to my selected context, if any, else  to my first context."
   [selection=0⇒ [selection←1]].
   thisContext sender ← list'selection.
   ["release abandoned contexts"
      selection>1⇒ [(list'(selection-1)) sender ← nil. (list'1) release]].
   list ← false. "Inhibit me closing." notifyWindow vanish. ⇑nil]
selected
      | context instance code safeVec
   [codePane @ false⇒ []
   context ← list'selection. instance ← context receiver. code ← self code.
   codePane showing: [code⇒ [code] ''].
   [code⇒ [contextVarPane names: (☞(thisContext) concat: context variables) values: (context, context
tempframe) wrt: context]
   contextVarPane names: ☞(thisContext) values: context inVector wrt: context].
   safeVec ← Vector new: 2. safeVec all ← instance.
   instanceVarPane names: (☞(self) concat: instance fields) values: safeVec wrt: context.
   contextVarPane select: 1]
selector | context
   [context ← list'selection. ⇑[context sender@nil⇒ [false] context sender thisop]]
yellowbug | mclass selector parag
   [stackmenu bug
      =1⇒ ["show a full backtrace" self revise: (list'1) stack with: [selection=0⇒ [nil] list'selection]];
      =2⇒ ["spawn a code editor"
            mclass ← (list'selection) mclass. selector ← self selector.
            parag ← [codePane⇒ [codePane contents] mclass canunderstand: selector⇒ [mclass code:
selector] ''].
            user schedule: (CodeEditor new class: mclass selector: selector para: parag).
            self compselection; select: 0];
      =3⇒ ["return nil to selected context" self proceed]]
!
```

"SystemPane"

```
Class new title: 'SystemPane';
   subclassof: ListPane;
   fields: 'mySysOrgVersion classPane';
   declare: 'sysmenu ';
   asFollows!
```

Uncommented

As yet unclassified

```
classes "return a Vector of the classes in my selected category"
   [selection=1⇒ [⇑user classNames];
          ≤2⇒ [⇑Vector new: 0]
   ⇑SystemOrganization category: list˙selection]
compile: parag
      | class cat className
   [selection=2⇒ [SystemOrganization fromParagraph: parag. self update] "new organization"
   cat ← [selection≤1⇒ [false] list˙selection]. class←nil³sparag.
   [(class is: Class) or: (class is: VariableLengthClass)⇒
      [className ← class title unique. cat⇒ [SystemOrganization classify: className under: cat]]].
   selection=0⇒ []
   classPane revise: [cat⇒ [SystemOrganization category: cat] user classNames] with: className]
deselected
   [classPane of: (Vector new: 0)]
dirty
   [⇑classPane dirty]
enter "be sure I am up to date"
   [mySysOrgVersion@user classNames⇒ [super enter]
   window outline. self update]
forget: className
   [user notify: 'All '+className+'s will become obsolete if you proceed...'.
   (Smalltalk˙className) obsolete. Smalltalk delete: className.
   SystemOrganization delete: className.
   AllClassNames ← AllClassNames delete: className.
   classPane revise: self classes with: className]
leave "I am up to date"
   [mySysOrgVersion ← user classNames. super leave]
noCode
   [selection=0⇒ [⇑'']; =2⇒ [⇑SystemOrganization]
   ⇑'Class new title: "NameOfClass";
   subclassof: NameOfSuperclass;
   fields: "names of fields";
   declare: "names of class variables"' copy]]
selected
   [classPane of: self classes]
to: classPane
   [sysmenu@nil⇒[sysmenu ← Menu new string: 'filout']]
update
   [self of: (☞(AllClasses SystemOrganization) concat: SystemOrganization categories).
mySysOrgVersion←user classNames]
yellowbug | f a c s
   [selection<3⇒[window flash]
   sysmenu bug=1⇒[s ← list˙selection.
      f ← dp0 file: (s subst: '-' for: ' ')+'.st.'.
      for: a from: (SystemOrganization category: s) do:
         [c ← Smalltalk˙a.
         user displayoffwhile:
            [c fullprinton: f. f next← 014; cr]]
      f skip: ¯2; shorten; close]
   ]
!
```

"VariablePane"

```
Class new title: 'VariablePane';
   subclassof: ListPane;
   fields: 'valuePane values context';
   declare: 'varmenu ';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
compile: parag
   [window flash]
deselected
   [valuePane showing: '']
execute: parag
   [⇑[context⇒ [context execute: parag] (values*1)'sparag]]
names: vars values: values wrt: context
   [self of: vars]
selected
   [valuePane showing: self value asString]
to: valuePane
   [varmenu@nil⇒ [varmenu ← Menu new string: 'inspect']]
value
   [selection=1⇒ [⇑values*1] ⇑(values*2) inspectfield: selection-1]
yellowbug
   [selection=0⇒ [window flash] varmenu bug =1⇒ [self value inspect]]
!
```

"Window"

```
Class new title: 'Window';
   fields: 'frame collapsed';
   declare: 'exitflag titleloc border titleframe windowmenu growing
';
   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.

Scheduling

```
eachtime
   [frame has: user mp⇒
      [user kbck⇒[⇑self kbd]
      user anybug⇒
         [user redbug⇒[⇑self redbug]
         user yellowbug⇒[⇑self yellowbug]
         user bluebug⇒[⇑self bluebug]]
      user anykeys⇒[⇑self keyset]]
   self outside⇒[]
   user anybug⇒[frame has: user mp⇒[] ⇑false]
   user kbck⇒[user kbd. frame flash] "flush typing outside"]
firsttime
   [frame has: user mp⇒
      [exitflag←true. growing←false. self enter]
   ⇑false]
lasttime
   [self leave. ⇑exitflag]
```

Framing

```
erase
   [(frame inset: ¯2@¯2) clear.
   titleframe window clear]
frame: f
   [frame ← (self fixframe: (f inset: border)) inset: 0@0-border]
newframe | a oldframe
   [user waitnobug.
   [frame@nil⇒[] self aboutToFrame. self erase].
   a ← OriginCursor showwhile: user waitbug.
   growing ← true.
   self frame: (frame ← self fixframe: (a rect: a+16)). self show.
   CornerCursor showwhile:
      [while: user anybug do:
         [oldframe ← frame copy.
         self frame: (frame ← self fixframe: (frame growto: user mp+16)).
         (oldframe inset: ¯2) clear. self show]].
   growing ← false.
   user cursorloc ← frame center]
outline
   ["Clear and outline me."
   frame outline]
show
   [frame outline. growing⇒[]
   titleframe put: self title at: frame origin+titleloc; comp]
takeCursor
   ["Move the cursor to my center."
   user cursorloc ← frame center]
title [⇑'unoccupied']
```

Default Event responses

```
aboutToFrame
   ["My frame is about to change.  I dont care."]
bluebug
   [windowmenu bug
      =1⇒[⇑exitflag ← false];
      =2⇒[self newframe. self enter];
```

```
        =3⇒[self close. self erase.
            user unschedule: self. ⇑exitflag ← false];
        =4⇒[frame flash]]
close []
enter [self show]
kbd [user kbd. frame flash]
keyset [frame flash]
leave []
outside [⇑false]
redbug
    [frame flash]
yellowbug
    [frame flash]
```

As yet unclassified

```
fixframe: f [⇑f]
init
    [border ← 2@2.
    titleframe ← Textframe new para: nil frame: nil.
    titleloc ← 1@(¯2-titleframe lineheight).
    windowmenu ← Menu new string:
'under
frame
close
print']
!
```

"files and compiler"
"version 3.5 6 Sept 77"

"Compiler"

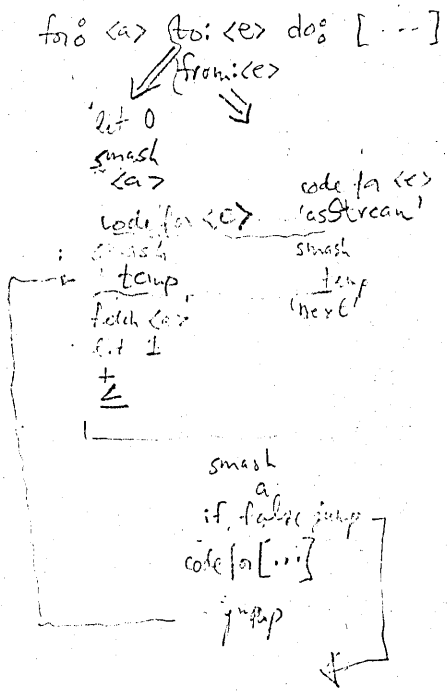
Class new title: 'Compiler';
fields: 'source code cascading stackused fixups';
declare: 'environment tempnames cdict selector literals maxstack instcode tempcode liticode areccode selfcode shortjmp shortbfp initcdict initopdict maxtemp precode litcode nilcode popcode endcode curcode jmpcode bfpcode ntemps opcode opdict constcode nargs stack trace smashpcode smashcode returncode supercode minopcode canoptpop ';
asFollows

This class has not yet been commented

As yet unclassified

```
addlit: a | b
   [b ← literals length.
   b≥48⇒[user notify: 'too many literals']
   literals ← literals , a. ↑b]
cascade | t
   [cascading⇒[]  "cascading is either false"
   (t ← code˙code loc)<minopcode⇒
      [user notify: 'improper cascading [...;].']
   code˙(code loc-1)<areccode⇒
      [cascading ← code˙(code loc-1)]    "or = temp code for receiver"
   code skip: ¯1. code next← smashcode.  "alloc temp if not simple"
   code next← cascading ← self newtemp. code next← t]
code [↑code contents]
compile: b in: class | a c i t
   [self init: class.  a ← b asVector asStream.
   self compilepattern: a.              "pattern and decls"
   [class canunderstand: selector⇒[]  "install code early first time"
      class install: selector method: nil literals: nil
         code: b backpointers: nil].
   self compileblock: a. self push.      "Smalltalk code"
   [self nofixups⇒[]  "no redundant return"
      [stack=maxstack⇒[self popopt]].
      code next← selfcode. code next← returncode].
   [
   c ← [a primitive: ⇒[a next] 0].   "primitive: <integer> clause"
   a← self qcodeck: c ⇒ []        "ckeck for rd/wrt accessors"
   a ← Stream default.                        "now make up code object"
   a next← 0; next← c; next← maxtemp+maxstack.  "header"
   a next← nargs; next← maxtemp; next← 6+(2*literals length).
   for: i from: literals do:                              "literals"
      [a next← i PTR lshift: ¯8; next← i PTR land: 0377]
   a append: precode contents; append: code contents.     "code body"
   ].
   literals ← literals concat: tempnames contents.
   class install: selector
      method: (code←a contents) literals: [literals length>0⇒[literals] nil]
      code: b backpointers: nil.
   ↑selector]
compilearg: a | b
   [tempnames next ← a. b ← self newtemp.
   cdict has: a⇒
      [precode next← b.      "compile assignments"
      precode next← smashpcode.  "for non-temp args"
      precode next← cdict˙a]
   cdict insert: a with: b]
compileblock: source | a b c inblock
   [cascading ← false.
   code ← Stream default. fixups ← (Vector new: 0) asStream.
   source end⇒[↑nullString] a ← stack.
   [source ⇒[]
      user notify: '[ missing before: '+source peek asString].
   while: inblock do:
```

```
        [source end⇒[user notify: 'unbalanced brackets']
        source ☞] ⇒[code next← nilcode. self push. inblock ← false]
        source ☞⇑ ⇒[code append: (self compileexpr: source). self push.
            code next← returncode. source ☞. . source ☞] ⇒[inblock ← false]
            "user show: 'junk following return stmt'."
            inblock ← false]
        self controlstmt ⇒[self uncascade. source ☞.]
        canoptpop ← true.
        code append: [cascading⇒[self compilecascade: source]
            self compileexpr: source]. self push.
        source ☞] ⇒[inblock ← false]
        source ☞⇒ ⇒[self pop. c ← Compiler new.
            b ← c compileblock: source.
                [source ☞; ⇒[self cascade]
                self uncascade. source ☞.].
            c nofixups⇒
                [code append: (self encodejmp: bfpcode by: b length); append: b]
            fixups next← code contents; next← b. code reset]
        [source ☞; ⇒[self cascade]
        self uncascade.  source ☞.⇒[]
        "user show: 'per missing before: '+source peek asString"].
        self pop. self popopt]      "period pops stack"
    self pop. stack≠a⇒[user notify: 'unbalanced stack']
    fixups empty⇒ [⇑code contents] canoptpop←false.
    self fixup: code contents and: fixups contents.
    ⇑code contents]
compilecascade: s [⇑self compilephrase: 4 from: s]
compileexpr: s [⇑self compilephrase: 3 from: s]
compilefactor: s [⇑self compilephrase: 1 from: s]
compilefor | a b lim
    [a ← source next.
    [(cdict has: a) and° (cdict*a<litcode)⇒[]
            user show: a asString+' is not local.'].
    a ← self encoderef: a.
    [source ☞to: ⇒
        [code next← self encodelit: 0; next← smashpcode; next← a.
        code append: (self compileterm: source). self push.
        code next← smashcode. b← code loc. self exstack: 2.
        code next← self newtemp; next← a; next← self encodelit: 1.
        code next← self encodeop: ☞+; next← self encodeop: ☞≤]
    source ☞from: ⇒
        [code append: (self compileterm: source). self push.
        code next← self encodeop: ☞asStream.
        code next← smashcode.
        b← code loc. code next← self newtemp. "reentry point"
        code next← self encodeop: ☞next]
    user notify: 'to: or from: expected before '+source peek asString].
    code next← smashcode; next← a.
    self compileloop: b.
    ntemps ← ntemps-1]
compileloop: b|a c
    [self pop.
    [source ☞do° ⇒[]
        user notify: 'do° expected before '+source peek asString].
    c ← Compiler new. c compileblock: source.
    c popopt. a ← c code.       "end with pop [opt]"
    code append: (self encodejmp: bfpcode by: 2+a length)."2 for jmp "n"
    code append: a; append: (self encodejmp: jmpcode by: b-code loc-2)]
compilepattern: a | c
    [ntemps ← 0. selector ← a peek.
    [selector iskeyword⇒      "selector and args"
        [c ← nullString.
        until° [a end⇒[] a peek iskeyword @ false] do°          "keywords"
            [c ← c+ a next.
            self compilearg: a next],
        selector ← c unique],
    a next.
    selector isinfix⇒    "infix"
        [self compilearg: a next],
    ].
    [a ☞← ⇒             "possible ← phrase"
```



```
        [selector ← (selector + '←') unique.
        self compilearg: a next].
    ].
    nargs ← ntemps.
    a ↞☞| ⇒          "further temp declarations"
        [until§ ☞[=a peek do§
            [self compiletemp: (c ← a next). ..  > why doesn't compiletemp do this ?
            tempnames next ← c]]]
compilephrase: level from: input | a b c i t stk more
    [a ← Stream default.
        [level=4⇒ [a next← cascading. level←3]
        self track§ (self compileprimary: input into: a level: level)⇒["no assign"]
        ⇑(self compileexpr: input) + a contents].
    stk ← stack. b ← (Vector new: 4) asStream.  "arg stack"
    b next← a; next← stackused.  a ← Stream default.        "operators"
    for§ i to: level do§
        [more ← true.  while§ more do§
            [input end⇒[more ← false] c←input peek.
            (c is: UniqueString) @ false⇒
                    [user notify: 'unexpected: '+ c asString]
                [i=1⇒[c isinfix⇒[more ← false]
                    c iskeyword⇒[more ← false]  c ← input next];
                =2⇒[c iskeyword⇒[more ← false]
                    ☞( . ; ⇒ [ ] ← ⇑) has: c⇒[more ← false]
                    c ← input next.
                    b next← self track§ (self compilefactor: input).
                    b next← stackused];
                =3⇒[☞(for§ until§ while§) has: c⇒[more ← false]
                    while§ [input end⇒[false] (c←input peek) iskeyword] do§
                        [t ← [t @ nil⇒[input next] t + input next].
                        a append: [c isuneval⇒
                                [self remote§ (self compileterm: input)]
                            self compileterm: input].
                        self push]
                    t@nil⇒[more ← false] c ← t unique]].
            more@false⇒[]
                [level=3⇒[input ↞☞← ⇒          "check here for  ← clause"
                    [c←(c+'←')unique.
                    [i=2⇒[self exstack: b pop. a append: b pop. self push]].
                    a append: (self compileexpr: input). self push]]].
            (b*1) next← self encodeop: c. i=3⇒[more ← false]]]
    b*1 ← (b*1) contents.  until§ b empty do§
        [self exstack: b pop. a append: b pop. self push]
    stack ← stk. ⇑a contents]
compileprimary: input into: a level: n | c
    [☞[ = input peek ⇒[c ← Compiler new. a append: (c compileblock: input)]
    c ← input next.
    c is: Vector⇒[a append: (self compileexpr: (c← Stream new of: c)).
        c end⇒[] user notify: 'extra stuff in parens: '+ a asString]
    c ← [☞☞=c⇒[self encodelit: input next] self encoderef: c].
    [n=3⇒[input ↞☞← ⇒          "expr-level call checks for ← "
        [c<(areccode+8)⇒[a next← smashcode; next← c. ⇑false]"assignment"
        user notify: 'assigning into read-only field']]].
    self exstack: 1.
    c=supercode⇒[a next← selfcode; next← supercode] a next← c]
compiletemp: a | b
    [b ← self newtemp.
    cdict has: a⇒
        [user notify: 'temp name used elsewhere: '+a asString]
    cdict insert: a with: b]
compileterm: s [⇑self compilephrase: 2 from: s]
compileuntil: a | b
    [b ← code loc.
    code append: (self compileterm: source). self push.
    [a⇒[code next← self encoderef: ☞false. self push.
        code next← self encodeop: ☞@. self pop]].
    self compileloop: b]
controlstmt
    [source ↞☞for§ ⇒[self compilefor]
    source ↞☞until§ ⇒[self compileuntil: true]
    source ↞☞while§ ⇒[self compileuntil: false]
```

```
    ⇑false]
encodejmp: a by: n | b
    [1≤n and: n≤8⇒[b←String new: 1.
        b˙1 ← n+[a=bfpcode⇒[shortbfp] shortjmp]-1. ⇑b]
    [n<0⇒[n←n+1024. n<0⇒[user notify: 'block too long.']]
        a ← a+4. n>1023⇒[user notify: 'block too long.']].
    b ← String new: 2.
    b˙1 ← n/256+ a.  b˙2 ← n\256.  ⇑b]
encodelit: a | b i
    [  [  [(b←a class)@Integer⇒[0≠(i← ☞(¯1 0 1 2 10) find: a)⇒
                [⇑constcode+i-1]]].
        for§ i to: (32 min: literals length) do§
            [literals˙i is: b⇒[literals˙i=a⇒[⇑litcode+i-1]]]].
    literals length<32⇒[⇑litcode +(self addlit: a)]
    b ← ObjectReference new. b value← a.
    ⇑liticode + (self addlit: b)]
encodeop: a
    [[trace⇒[user show: a asString]].
    opdict has: a⇒[⇑opdict˙a]
    opdict insert: a with: (opcode + (self addlit: a)).
    ⇑opdict˙a]
encoderef: a | t
    [[trace⇒[user show: a asString]].
    a is: UniqueString⇒
        [t ← cdict lookup: a ⇒ [⇑t]
        a isinfix or: a iskeyword⇒
            [user notify: 'missing receiver before: ' + a asString]
        cdict insert: a with: (t← self newref: a).  ⇑t]
    ⇑self encodelit: a]
exstack: x
    [maxstack ← maxstack max: stack + x]
fixup: a and: b | c i t
        "b˙odd=main code, c˙odd ← bfp around consequents,
        b˙even=consequents, c˙even ← jmp to end"
    [c ← Vector new: b length. t ← a length.
    for§ i from: (b length-1 to: 1 by: ¯2) do§      "compute the jumps"
        [c˙(i+1) ← self encodejmp: jmpcode by: t.
        c˙i ← self encodejmp: bfpcode by: ((b˙(i+1)) length+(c˙(i+1)) length).
        t ← t+(b˙i) length+(b˙(i+1)) length+(c˙i) length+(c˙(i+1)) length]
    code reset. for§ i to: b length do§       "collate all back into code"
        [code append: b˙i; append: c˙i]
    code append: a]
init: class | a i
    [environment ← class environment concat: Smalltalk , Undeclared.
    [initcdict@nil⇒      "initial symbol defs"
        [initcdict ← Dictionary new init: 16.
        initcdict insertall: ☞(self thisContext super nil false true)
            with: ☞(113 133 134 125 126 127).
        initopdict ← Dictionary new init: 64.
        initopdict insertall: ☞(+ - < > ≤ ≥ = ≠
                * / \ | min: max: land: lor:
                ˙ "˙←" next "next←" length @
                class and: or: new new: to: oneToMeAsStream asStream)
            with: ☞(176 177 178 179 180 181 182 183
                184 185 186 187 188 189 190 191
                192 "193" 194 "195" 196 197
                200 201 202 203 204 205 206 207).
        initopdict insert: '˙←' unique with: 193.
        initopdict insert: 'next←' unique with: 195] ].
    cdict ← Dictionary new copyfrom: initcdict.
    opdict ← Dictionary new copyfrom: initopdict.
    a ← class instvars. for§ i to: a length do§
        [cdict insert: a˙i with: instcode+i-1].
    stack← ntemps← maxtemp← 0. maxstack← 1.
    precode ← Stream default. tempnames ← (Vector new: 4) asStream.
    literals ← Vector new: 0]
newref: a | i
    [for§ i from: environment do§
        [i has: a⇒
            [⇑liticode + (self addlit: (i ref: a))]]
    user show: a asString + ' is undeclared.'.
```

```
    Undeclared define: a  as: nil.
    ↑liticode + (self addlit: (Undeclared ref: a))]
newtemp
    [maxtemp ← maxtemp max: (ntemps ← ntemps+1).
    maxtemp>16⇒[user notify: 'Too many temps required.']
    ↑tempcode + ntemps-1]
nofixups    "↑true if no jmpout needed"
    [fixups empty⇒          "...meaning no internal branches"
        [code loc=0⇒[↑false]  ↑code'code loc=returncode]"AND ends with ↑"
    ↑false]
pop
    [0>(stack ← stack-1)⇒[user notify: 'negative stack']]
popopt | t        "append pop and optimize"
    [2>(t←code loc)⇒[t=1⇒[code skip: ¯1]]
    canoptpop@false⇒ [code next← popcode]
    code't=nilcode⇒          "jmp1-nil-pop -> pop"
        [[code'(t-1)=shortjmp⇒[code skip: ¯2]]. code next← popcode]
    code'(t-1)=smashcode⇒
        [code'(t-1) ← smashpcode]"sto-var-pop -> stopop var"
    code next← popcode]
push
    [maxstack ← maxstack max: (stack ← stack+1).
    trace⇒[stack print]]
qcodeck: t | a b i        "fast read/write accessors"
    [code loc>2⇒[↑false]
    t≠0⇒[↑false]    "cant opt primitives"
    code'2≠returncode⇒[↑false]
    nargs>0⇒[↑false]   "cant opt if fetch args"
    a ← Stream default. a next← 0.
    code'1≠selfcode⇒
        [code'1<tempcode⇒
            [a next← 40; next← 0; next← 0; next← code'1.
            ↑a]         "quick ↑inst var"
        ↑false]
    a next← 1. ↑a]       "quick ↑self"
remote: b | a t
    [a←Stream default.
    a next← curcode; next← self encodeop: ☞remoteCopy.
    t ← b length+3. a next← t/256+jmpcode+4; next← t\256.
    a append: b; next← endcode; append: (self encodejmp: jmpcode by: 0-t).
    ↑a contents]
track: expr | a b val
    [a←maxstack.  maxstack← b← stack.
    val ← expr eval.
    b≠stack⇒[user notify: 'unbalanced stack']
    stackused ← maxstack-stack.  maxstack ← a.
    ↑val]
uncascade
    [cascading⇒[cascading<areccode⇒[cascading ← false]
        ntemps ← ntemps-1. cascading ← false]]
!
```

"Directory"

Class new title: 'Directory';
fields: 'dirinst bitinst filinst junkfile closed';
declare: 'CWW boffset disksize dirname CRR dfmask ';
asFollows!

This class has not yet been commented

As yet unclassified

```
alloc: dadr      "allocate a new disk page from bitTable"
      | index start stop ch i m
   [start ← (bitinst altoToVirtual: dadr) land: 0177770.  "start near dadr"
   stop ← disksize.
   for§ i to: 2 do§
      [bitinst settopage: 1 char: start/8 + boffset.
      for§ index from: (start to: stop by: 8) do§
         [ch ← bitinst next.
         ch = 0377⇒ []
         m ← 0200.
         while§ m > 0 do§ [
            (ch nomask: m) and§
               [ch ← ch lor: m.
               junkfile zaplabel.  "check if page is really free"
               junkfile dskio: (junkfile virtualToAlto: index)
                  command: CRR page: 0 pages: 1
                  onerror§ [false]] ⇒[m←¯1]
            index ← index + 1.
            m ← m lshift: ¯1]
         bitinst last ← ch.          "update bittable"
         m = ¯1⇒ [↑junkfile curadr]]
      stop ← start.     "wrap around to where we started"
      start ← 0]
   bitinst error: 'nofreepages']
alloc: dadr with: l      "alloc a page ready to append"
   [↑junkfile link: (self alloc: dadr) to: dadr with: l]
asStream [self reset]
close [closed⇒[]
   filinst close.
   bitinst close.
   closed ← true]
dealloc: dadr | index ch m [
   until§ dadr = 0 do§ [
      index ← bitinst altoToVirtual: dadr.
      bitinst settopage: 1 char: index/8 + boffset.
      "mark page as free in bittable"
      ch ← bitinst next.
      [ch allmask: (m ← 0200 lshift: 0-(index land: 7))⇒ [
         bitinst last ← ch - m]
      user show: 'page already free (Dir.dealloc:)'].
      junkfile dskio: dadr
         command: CRR page: 0 pages: 1
         onerror§ [self error: 'dealloc'].
      dadr ← junkfile nextp.
      junkfile zaplabel.
      junkfile docommand: CWW]
   bitinst flush]
delentry: name | entry  "**later merge with adjacent**"
   [entry ← self find: name⇒
      [filinst skip: 0-entry length. "mark entry deleted"
      filinst nextword ← (entry word: 1) land: dfmask-1.
      filinst flush. ↑entry]
   ↑false]
delete: name | entry
   [entry ← self delentry: (junkfile namecheck: name)⇒
      [self dealloc: (junkfile virtualToAlto: (entry word: 6)).
      ↑name+' deleted. ']
   ↑false]
file: fname
```

```
    [⇑[File new on: self; named: fname]]
find: name | entry entrylen w
    [name = dirname⇒
        [entry ← String new: 12.
        entry word: 2 ← 0100000. entry word: 3 ← 0.
        entry word: 4 ← 1. entry word: 6 ← 1. ⇑entry]
    self reset.
    while⁸ (w ← filinst nextword) do⁸
        [entrylen ← 2*(w land: dfmask-1).
        w allmask: dfmask⇒
            [filinst skip: 10.          "normal entry - check name"
            name length=filinst next⇒
                [name-(filinst into: (String new: name length))=0⇒
                    [filinst skip: 0-13-name length.
                    ⇑filinst into: (String new: entrylen)]
                filinst skip: entrylen-13-name length]
            filinst skip: entrylen-13]
        name˙1=03⇒                  "deleted entry"
            [entrylen≥(13+name length)⇒  "name.1=03 => want a hole"
                [entry ← String new: 13 + (name length lor: 1).
                entrylen>entry length⇒
                    [filinst skip: entry length-2. "excess marked deleted"
                    filinst nextword← (entrylen-entry length/2) land: dfmask-1.
                    filinst skip: ¯2-entry length. ⇑entry]
                filinst skip: ¯2. ⇑entry]
            filinst skip: entrylen-2] "skip hole too small"
        filinst skip: entrylen-2] "skip hole not wanted"
    name˙1=03⇒[⇑String new: 13 + (name length lor: 1)]
    ⇑false]
free | npages tpages ch i [
    self reset.
    npages ← 0.
    bitinst settopage: 1 char: boffset.
    for⁸ i from: (1 to: disksize by: 8) do⁸ [
        ch ← bitinst next.
        ch=0⇒ []; =0377⇒ [npages ← npages+8]
        until⁸ ch = 0 do⁸ [
            [ch allmask: 1⇒ [npages ← npages+1]].
            ch ← ch lshift: ¯1]]
    ⇑disksize - npages]
getNewSn | sn [
    bitinst settopage: 1 char: 010.
    sn ← bitinst into: (String new: 4).
    sn word: 2 ← (sn word: 2) + 1.
    [(sn word: 2) = 0⇒ [sn word: 1 ← (sn word: 1) + 1]].
    bitinst appendlast: sn.
    bitinst flush.
    ⇑sn]
init
    [dfmask ← 02000.  "bit meaning active directory entry"
    boffset ← 040.  "byte offset of bit table in DiskDescriptor"
    dirname ← 'SysDir.'.
    disksize ← 4872]
insert: name | entry t
    [t← name copy. t˙1← 03.
    entry ← self find: t. "fill a hole"
    entry word: 1 ← entry length/2 lor: dfmask.
    entry˙(3 to: 6) ← self getNewSn.
    entry word: 4 ← 1. "version"
    entry word: 5 ← 0. "?"
    entry˙13 ← name length.
    entry˙(14 to: 13+name length) ← name.
    ⇑entry]
list | entry
    [for⁸ entry from: self do⁸
        [user cr; show: entry˙(14 to: entry˙13 + 13)]]
list: pattern | entry
    [  [pattern last≠056⇒[pattern←pattern+'.']].
    for⁸ entry from: self do⁸
        [pattern match: entry˙(14 to: entry˙13 + 13)⇒
            [user cr; show: entry˙(14 to: entry˙13 + 13)]]]
```

```
next | w entrylen
    [whileº (w ← filinst nextword) doº
        [entrylen ← 2*(w land: dfmask-1).
        w allmask: dfmask⇒[filinst skip: ¯2.
            ⇑filinst into: (String new: entrylen)]
        filinst skip: entrylen-2]
    ⇑false]
next ← entry [
    filinst append: entry.
    filinst flush]
on: dirinst []
open [self reset]
rename: name to: newname | entry nentry
    ["assumes newname already checked to be not there"
    entry ← self delentry: name⇒
        [nentry ← self insert: newname.
        nentry'(3 to: 12) ← entry'(3 to: 12).
        self append: nentry]
    ⇑false]
reset [
    "only system directories implemented now"
    closed⇒ [
        closed ← false.
        filinst @ nil⇒ [
            filinst ← [File new readwrite old on: self; named: dirname].
            bitinst ← [File new readwrite old on: self; named: 'DiskDescriptor.'].
            junkfile ← [File new readwrite old on: self; named: 'Com.cm.']]
        filinst reopen.
        bitinst reopen.
        junkfile reopen]
    filinst reset.
    bitinst @ nil ⇒[] bitinst flush]
restore
    [filinst← nil. closed← true. self open]
!
```

"File"

Class new title: 'File';
subclassof: Stream;
fields: 'dirinst filename rvec label rwmode leader curadr status';
declare: 'oldornew CCW CWW pagelength version shorten created lnused sn1 sn2 old new read nextp backp numch pagen write CRR CRW CCR ';
asFollows!

This class has not yet been commented

As yet unclassified

```
altoToKeys: dadr | i k [
    k ← Stream default.
    for° i to: 14 do° [
        dadr allmask: (0100000 lshift: 1-i)⇒ [
            k next←'546E7DUV0K-P/\'•i]]
    ⇑k contents]
altoToVirtual: dadr [
    ⇑(dadr lshift: ¯12) + (3 * (dadr land: 07774))]
appendlast: x [
    self skip: 0-x length.
    self append: x]
close [
    rwmode allmask: shorten⇒ [self shortento: label•pagen char: position]
    self flush]
contents | s [
    self settoend.
    s ← String new: (label•pagen-1)*pagelength + (label•numch).
    self reset.
    ⇑self into: s]
curadr [⇑curadr]
delete
    [rwmode allmask: write⇒
        [⇑dirinst delete: filename]
    ⇑false]
docommand: com [
    self dskio: curadr
        command: com
        page: label•pagen
        pages: 1
        onerror° [⇑self error: 'docommand']]
dskio: curadr
    command: command
    page: startpage
    pages: npages
    onerror° exp
    [self dskprim: curadr
        command: command
        page: startpage
        pages: npages⇒[]
    ⇑exp eval]
dskprim: curadr
    command: command
    page: startpage
    pages: npages
    [⇑false] primitive: 80
edit [user schedule:
        (CodeEditor new class: self selector: 'editing' para: self contents asParagraph)]
end [
    position < (label•numch)⇒ [⇑false]
    ⇑label•nextp = 0]
error: s [
    user notify: 'File error: '+s+' in '+filename asString.
    ⇑false]
extendby: n
    [self settoend; extendto: label•pagen+n]
```

```
extendto: dest
    [rwmode nomask: write⇒ [↑false]
        [rvec @ nil⇒[] rvec ← rvec copyto: (Vector new: dest)].
    until° label•pagen = dest do°
        [label•nextp ← dirinst alloc: curadr with: label.
        label•numch ← pagelength.
        self docommand: CWW.
        label•backp ← curadr.
        curadr ← label•nextp.
        label•pagen ← label•pagen + 1.
        rvec @ nil⇒[]
        rvec•(label•pagen) ← curadr]
    label•nextp ← 0.
    label•numch ← 0.
    position ← 0]
fileid | v [
    v ← Vector new: 5.
    v•1 ← label•sn1.
    v•2 ← label•sn2.
    v•3 ← label•version.
    v•4 ← 0.
    v•5 ← curadr.
    ↑v]
filin | s
    [until° self end do°
        [FilinSource ← self.
        s ← self upto: 036.
        user show: (nil'ss) asString; space]]
filout [user displayoffwhile° [self filout: Changes contents sort]]
filout: source | s selector class old heading h
    [for° s from: source do°
        [user show: s; cr.
        s ← s asStream.
        class ← Smalltalk•(s upto: 040) unique.
        selector ← (s upto: 040) unique.
            [class=old⇒[]
                [old@nil⇒[] self append: '!'; ctlz: '\g'; cr].
            self append: class title+' asFollows!'; ctlz: '\f5bg'.
            old ← class. heading ← 'As yet unclassified'].
        h ← class organization invert: selector.
            [h≠heading⇒[self append: (heading← h); ctlz: '\f5bg']].
        self append: (class code: selector) makeBoldPattern toBravo text]
    self append: '!'; ctlz: '\g'; shorten; close]
filoutclass: class | c
    [class is: Vector⇒
        [for° c from: class do°
            [self filoutclass: c; next← 014 "FF"]
        self skip: ¯1]
    [class is: UniqueString⇒
        [class ← Smalltalk•class]].
    user displayoffwhile°
        [class fullprinton: self].
    self shorten; close]
find | entry [
    "not debugged yet... uses new insert"
    [status @ nil⇒ [self oldornew]].
    [rwmode @ nil⇒ [self readwrite]].
    [dirinst @ nil⇒ [dirinst← dp0]].
    [array @ nil⇒ [self of: (String new: pagelength)]].
    label ← (Vector new: 8) all← 0.
    label•numch ← pagelength.
    entry ← dirinst find: filename⇒
        [status = new⇒ [↑false]
        label•sn1 ← entry word: 2.
        label•sn2 ← entry word: 3.
        label•version ← entry word: 4.
        self dskio: (leader ← self virtualToAlto: (entry word: 6))
            command: CCR
            page: 0
            pages: 1
            onerror° [↑self error: 'leader page'].
```

```
        self init]

    status = old⇒ [⇑false]
    rwmode nomask: write⇒ [⇑false]
    entry ← dirinst insert: filename.
    status ← created. "for leaderstamp"
    label'sn1 ← entry word: 2.
    label'sn2 ← entry word: 3.
    label'version ← entry word: 4.
    (curadr ← leader ← dirinst alloc: 0) @ false⇒ [⇑false]
    entry word: 6 ← self altoToVirtual: leader.
    dirinst next ← entry.
    self init]
flush [
    rwmode nomask: write⇒ [⇑false]
    label'numch < position⇒ [
        position ≠ pagelength⇒ [label'numch ← position.
                self docommand: CWW]
        label'nextp = 0⇒ [self extendto: label'pagen + 1]
        self docommand: CWW]
    self docommand: CCW]
getname: name | t [
    user show: 'type a file name'.
    user show: [name length > 0⇒ [', just ⏎ for ' + name] ' ']
    t ← user read.
    t @ false⇒ [⇑false]
    ⇑self named: [t @ nil⇒ [name] t]]
init | oldmode s [
    s ← String new: 4.
    s word: 1 ← mem'0572.
    s word: 2 ← mem'0573.
    oldmode ← rwmode.
    self readwrite.

    "assumes positioned at page 0"
    position ← 0.
    [status ≠ created⇒ [self skip: 4]
    self append: s]. "creation date"

    [oldmode allmask: write⇒ [self append: s]"write date"
    self skip: 4].

    "read date"
    self append: s.

    [status = created⇒ [
        self next ← filename length. "file name (Bcpl style)"
        self append: filename]].

    [rvec @ nil⇒ []
    "random access"
    self settoend.
    [label'pagen > rvec length⇒ [
        rvec ← rvec copyto: (Vector new: label'pagen)]].
    for° s to: rvec length do° [
        rvec's @ nil⇒ [
            self settopage: s char: 0.
            rvec's ← curadr]]].

    self reset.
    rwmode ← oldmode]
last ← x [
    self skip: ¯1.
    self next ← x]
link: curadr to: dadr with: l       "valid eof, ready to link"
    [l copyto: label.
    label'nextp ← 0.
    label'backp ← dadr.
    label'pagen ← label'pagen+1.
    label'numch ← 0.
    self docommand: CWW.
```

```
    ⇑curadr]
namecheck: name | i x [
    name length = 0⇒ [⇑self error: 'empty name']
    name length > 255⇒ [⇑self error: 'name too long']
    for§ i to: name length do§ [
        "check characters"
        x ← name·i.
        x isletter⇒ []
        x isdigit⇒ []
        0 = ('+-()⇑⇒.' find: x)⇒ [
            ⇑self error: 'illegal character ' + (name·(i to: i))]]
    x ≠ ('.'·1)⇒ [⇑name + '.']
    ⇑name]
named: filename [
    filename ← self namecheck: filename⇒ [⇑self find]
    ⇑false]
new [status ← new]
nextp [⇑label·nextp]
nextword | hi lo
    [hi ← self next⇒
        [lo ← self next⇒
            [⇑256*hi+lo]
        ⇑false]
    ⇑false]
nextword ← w | s [
    s ← String new: 2.
    s word: 1 ← w.
    self positioneven.
    self append: s]
old [status ← old]
oldornew [status ← oldornew]
on: dirinst []
overlay [user overlay: self fileid]
pastend [
    self end⇒ [⇑false]
    self settopage: label·pagen+1 char: 0.
    ⇑self next]
pastend ← x [
    [limit < pagelength⇒ [limit ← pagelength]
    self settopage: label·pagen+1 char: 0].
    ⇑self next ← x]
positioneven [
    position allmask: 1⇒ [self skip: 1]]
printon: strm
    [strm append: 'File ' + filename asString]
random [rvec ← Vector new: 0]
readonly [rwmode ← read]
readwrite [rwmode ← read + write]
rename: newname [
    rwmode nomask: write⇒ [⇑false]
    (newname ← self namecheck: newname) @ false⇒
        [⇑self error: 'badnew name']
    dirinst find: newname⇒ [⇑self error: newname + ' already exists']
    (dirinst rename: filename to: newname) @ false⇒
        [⇑self error: 'file not there']
    filename ← newname.
    self settopage: 0 char: 12.
    self next ← filename length.
    self append: filename]
reopen [
    self dskio: leader
        command: CCR
        page: 0
        pages: 1
        onerror§ [
            "leader page failed"
            self oldornew.
            ⇑self find].
    filename ≠ (array·(14 to: array·13 + 13))⇒ [
        "name failed - how about ser no?"
        self oldornew.
```

```
        ↑self find]

    [rvec @ nil⇒ []
    self random].
    self old.
    self init]
reset [self settopage: 1 char: 0]
settoend | oldmode [
    self flush.
    oldmode ← rwmode.
    self readonly.
    self settopage: 5000 char: 0.
    rwmode ← oldmode]
settopage: page char: char | pch pn [
    pch ← [char<0⇒ [0-char] char].
    page ← char/pagelength + page.
    pch ← pch \ pagelength.
    [char < 0 and: pch ≠ 0⇒[
        pch ← pagelength - pch. page ← page-1]].
    page < 0⇒ [↑self error: 'negative page']
    "possibly write current page"
    [label·pagen < page and° label·nextp = 0⇒ [self extendto: page]
    label·pagen=page⇒[] self flush].
    "try random access"
    [rvec @ nil⇒ []
    label·pagen = page⇒ ["already there"]
    "chain read from the nearest page we can"
    pn ← page min: rvec length.
    while° [pn > 0 and° rvec·pn @ nil] do° [pn ← pn - 1]
    self dskio: [pn = 0⇒ [leader] rvec·pn]
        command: CCR page: pn pages: page+1-pn
        onerror° [↑self error: 'random access']].
    "do sequential reads to get there, possibly extending file"
    [label·pagen < page⇒ [
        "chained read forward"
        [label·nextp ≠ 0⇒ [
            self dskio: label·nextp
                command: CCR page: label·pagen+1
                pages: page-(label·pagen)
                onerror° [↑self error: 'forward chaining']]].
        label·pagen = page⇒ ["finished"]
        rwmode allmask: write⇒ [self extendto: page]
        "change the request"
        page ← label·pagen. pch ← label·numch]
    label·pagen > page⇒
        [label·pagen-1 = page⇒
            ["one page backwards"
            self dskio: label·backp
                command: CCR page: page pages: 1
                onerror° [↑self error: 'reading backwards']]
        "forward from leader"
        self dskio: leader
            command: CCR page: 0 pages: page+1
            onerror° [↑self error: 'leader reading']]].
    label·pagen ≠ page⇒ [↑self error: 'didnt get to page']
    position ← [rwmode allmask: write⇒ [pch] label·numch min: pch].
    limit ← label·numch]
shorten [self shortento: label·pagen char: position]
shortento: page char: char [
    rwmode nomask: write⇒ [↑false]
    self settopage: page char: char.
    dirinst dealloc: label·nextp.
    label·nextp ← 0.
    label·numch ← label·numch min: char.
    self docommand: CWW.
    rvec @ nil⇒ []
    rvec ← rvec copyto: (Vector new: label·pagen)]
skip: n | t
    [t ← position + n.
        [t≥0⇒[t≤limit⇒[position ← t. ↑self]]].
    self settopage: label·pagen char: t]
```

```
title [↑filename]
understands: s      "rewrites file with contents of edit window"
    [self reset; append: s; shorten; flush]
virtualToAlto: vadr [
    ↑(vadr \ 12 lshift: 12) + (vadr/12 * 4)]
writeonly [rwmode ← write]
zaplabel [
    label'nextp ← 0.
    label'pagen ← 0.
    label'sn1 ← ¯1.
    label'sn2 ← ¯1.
    label'version ← ¯1]
!
```

"Reader"

```
Class new title: 'Reader';
  fields: 'source sink token nextchar typetbl scantbl';
  declare: 'typetable scantable separ digit letter notcr notcomnt
notapos dot ';
  asFollows!
```

This class has not yet been commented

As yet unclassified

```
mknum: str base: base | val c
   [val ← [str length>4⇒ [0.0] 0].
   for° c from: str do°
      [val ← val*base + (c-060)]
   ⇑val]
of: source
   [typetbl ← typetable. scantbl ← scantable.
   sink ← (Vector new: 10) asStream. token ← Stream default.
   self step]
rdcom
   [self step. self scan: notcomnt.
   nextchar@false⇒ [user notify: 'Unmatched comment quote']
   self step]
rdnum | sign val d
   [sign ← [nextchar=025⇒ [self step. -1] 1]."sign"
   d ← [nextchar=060⇒ [8] 10].          "base"
   val ← self mknum: (self scan: digit) base: d. "integer part"
   dot ← false.
   nextchar=056⇒      "check for decimal point"
      [self step.
      nextchar isdigit@false⇒
         [dot ← true. ⇑sign*val]    "was <Integer> . "
      d ← self scan: digit.
      val ← (self mknum: d base: 10)asFloat / (10.0 ipow: d length) + val.
      nextchar=0145⇒     "check for e<exponent> "
         [self step. ⇑val*(10.0 ipow: self rdnum)*sign]
      ⇑val*sign]
   ⇑sign*val]
rdstr | a
   [self step. a ← self scan: notapos.
   nextchar @ false⇒ [user notify: 'Unmatched String quote']
   self step.
   nextchar=047⇒ "imbedded String-quote"
      [token next ← 047. a← token contents
      ⇑a + self rdstr]
   ⇑a]
read | x
   [while° nextchar do°
      [x ← typetbl'(nextchar+1).
      x=1⇒ [self scan: separ];        "separators"
      =2⇒ [sink next ← (self scan: letter+digit) unique];"identifiers"
      =3⇒ [sink next ← (self scan: 0) unique]; "1-char tokens"
      =4⇒ [sink next ← self rdnum. dot⇒[sink next← ☞.]];"Numbers"
      =5⇒ [sink next ← self rdstr];          "Strings"
      =6⇒ [self step. sink next ← self subread]; "sub-Vectors"
      =7⇒ [self step. ⇑sink contents]; "close-paren"
      =8⇒ [self rdcom];           "comments"
      =9⇒ [self scan: notcr];         "↑Z format runs"
      =10⇒ [⇑sink contents];        "null and DOIT"
      ]
   ⇑sink contents]
scan: mask | x
   [mask=0 ⇒[x ← String new: 1. x'1 ← nextchar. self step. ⇑x]
   token reset.
   while° nextchar do°
      [(mask land: scantbl'(nextchar+1))=0⇒ [⇑token contents]
      token next ← nextchar. nextchar ← source next]
   ⇑token contents]
```

print out typetbl, scantbl!
[add into message?]

```
step
    [nextchar ← source next]
subread | a b
    [a ← sink.  sink ← (Vector new: 10) asStream.
    b ← self read.  sink ← a.  ⇑b]
!
```

"primitive access"

"version 3.5 6 Sept 77"

"BitBlt"

```
Class new title: 'BitBlt';
  fields: 'function color destbase destraster destx desty
      width height sourcebase sourceraster sourcex sourcey';
  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

```
color [↑color]
color ← color
destbase [↑destbase]
destbase ← destbase
destraster [↑destraster]
destraster ← destraster
destx [↑destx]
destx ← destx
desty [↑desty]
desty ← desty
function [↑function]
function ← function
height [↑height]
height ← height
sourcebase [↑sourcebase]
sourcebase ← sourcebase
sourceraster [↑sourceraster]
sourceraster ← sourceraster
sourcex [↑sourcex]
sourcex ← sourcex
sourcey [↑sourcey]
sourcey ← sourcey
width [↑width]
width ← width
```

Setup

```
forCursor [function← color← 0.
    destbase← sourcebase← 0431.
    width← height← 16. destraster← sourceraster← 1.
    destx← desty← sourcex← sourcey← 0]
init
    [function ← color ← destbase ← destraster ← destx ← desty ← width ←
    height ← sourcebase ← sourceraster ← sourcex ← sourcey ← 0]
```

Operations

```
callBLT
    [] primitive: 33
copy: mode
    [function ← mode land: 3. self callBLT]
copycomp: mode
    [function ← 4 + (mode land: 3). self callBLT]
copyFromString: s mode: m
    [sourcebase ← s lock. self copy: m. s unlock]
copyToString: s mode: m
    [destbase ← s lock. self copy: m. s unlock]
fill: mode color: color
    [function ← 12 + (mode land: 3). self callBLT]
paint: mode
    [function ← 8 + (mode land: 3). self callBLT]
!
```

"CoreLocs"

```
Class new title: 'CoreLocs';
   subclassof: Array;
   fields: 'base length';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
· x [user croak] primitive: 42
· x ← val [user croak] primitive: 43
base [⇑base]
base: base length: length
length [⇑length]
!
```

"FieldReference"

```
Class new title: 'FieldReference';
   fields: 'object offset';
   asFollows!
```

This class has not yet been commented

As yet unclassified

```
object: object offset: offset
value [⇑object instfield: offset]
value ← value
   [⇑object instfield: offset ← value]
!
```

"PriorityInterrupt"

```
Class new title: 'PriorityInterrupt';
   fields: 'scheduler priority';
   asFollows!
```

PriorityInterrupts fill the need for (sched, level) pairs. All messages are simply passed on to the scheduler with the priority as an argument

As yet unclassified

```
+ arg
    [↑priority + arg]
deepsleep
    [scheduler deepsleep: priority]
disable
    [scheduler disable: priority]
enable
    [scheduler enable: priority]
from: scheduler at: priority
reset
    [scheduler reset: priority]
restart
    [scheduler restart: priority]
run: newContext
    [scheduler run: newContext at: priority] primitive: 87
sleep
    [scheduler sleep: priority]
swap: newContext
     with: fieldReference
    [scheduler
        swap: newContext
        at: priority
        with: fieldReference] primitive: 88
terminate
    [scheduler terminate: priority]
wakeup
    [scheduler wakeup: priority]
!
```

"PriorityScheduler"

```
Class new title: 'PriorityScheduler';
  fields: ' sourceIndirect
      "an indirect reference to the source of power,
      ie the source from which this scheduler was spawned,
      and who therefore holds the suspension if it is suspended."
    suspendedContexts
      "<Vector of Contexts> the suspended processes"
    initialContexts
      "<Vector of Contexts> root processes for restarting"
    enabledPriorities
      "<Integer> priorities which can receive control"
    awakePriorities
      "<Integer> priorities which are requesting control"
    interruptedPriorities
      "<Integer> new priorities which are requesting control"
    currentPriority
      "<Integer> priority which currently has control"
    usedPriorities
      "<Integer> priorities which have processes installed"';
  declare: 'CtlShftEscInt CtlCDisp UserInt CtlCInt ';
  sharing: BitMasks;
  asFollows!
```

The underlying machine has a pointer to the top-level scheduler (Top), so that physical interrupts can also cause interrupts in Smalltalk. This they do by calling their own copy of wakeup and reselect. This copy is also invoked (primitive 65) when reselect is sent to the top-level scheduler, since its source is the virtual machine itself.

As yet unclassified

```
critical: expr| t v
    ["Execute <expr> without allowing it to be interrupted"
     t ← self disable.
     v ← expr eval.
     enabledPriorities ← t.
     self reselect. ↑v]
deepsleep: priority
    ["Request the process at <priority> to cease running and ignore any new wakeups. Turn off the
corresponding bit in awakePriorities and interruptedPriorities and check if that changes who should run"
     awakePriorities ← awakePriorities land: bitoff•priority.
     interruptedPriorities ← interruptedPriorities land: bitoff•priority.
     self reselect]
disable | t
    ["This message should deffinitely be protected. Zero all the enabled flags and return the previous
value of them"
     t ← enabledPriorities. enabledPriorities ← 0. ↑t] primitive: 66
disable: priority
    ["Prevent the process at <priority> from being activated by a wakeup. Turn off the corresponding
bit in enabledPriorities and check if that changes who should run"
     enabledPriorities ← enabledPriorities land: bitoff•priority.
    self reselect]
enable: priority
    ["Allow the process at <priority> to be activated by a wakeup. Turn on the corresponding bit in
enabledPriorities and check if that changes who should run"
     enabledPriorities ← enabledPriorities lor: biton•priority.
     self reselect]
fromSource: sourceIndirect
    ["Initialize a scheduler having 16 spaces for processes"
     suspendedContexts ← Vector new: 16.
     initialContexts ← Vector new: 16.
```

```
    enabledPriorities ← 0.
    awakePriorities ← 0.
    interruptedPriorities ← 0.
    usedPriorities ← 0.
    currentPriority ← 0.]
init1
   [UserInt ← Top
      install: [user show: 'Level 1 restart'. user restart]
      at: 1]
init11 | ThisSaveFile GrowFlasher
   [ThisSaveFile ← dp0 file: 'small.sv'.
    GrowFlasher ← (540@0) rect: (556@16).
    GRODSK ← Top
      install:
         [GrowFlasher comp.
         ThisSaveFile reopen.
          dp0 free<100⇒
            [ThisSaveFile extendby: dp0 free.
             GRODSK deepsleep].
          ThisSaveFile extendby: 100.
          GrowFlasher comp.
          GRODSK deepsleep]
      at: 11.
   GRODSK enable]
init8 | t
   [CtlCDisp ← Dispframe new rect: ((10@0) rect: (250@112)).
    user show.
    CtlCInt ← Top
      install:
         [CtlCDisp clear; append:
            '///   Control c Window   ///
     Control d to exit.
If the compiler is running, type control d'.
         while: [t ← CtlCDisp request: '
↑c :'] do:
            [t@nil⇒[CtlCDisp print: nil doit; show]
             CtlCDisp print: nil↑st; show]
         user show. CtlCInt sleep]
      at: 8.
    CtlCInt enable]
init9
   [CtlShftEscInt ← Top
      install: [UserInt restart. CtlShftEscInt sleep]
      at: 9.
    CtlShftEscInt enable]
init: priority
      | newPriority oldPriority newContext tempenabled tempinterrupts
   ["This message should be protected. It is used by reset: and restart: to actually switch suspended
processes"
    tempenabled ← self disable.
    tempinterrupts ← interruptedPriorities land: awakePriorities.
    awakePriorities ← interruptedPriorities lor: awakePriorities.
    interruptedPriorities ← tempinterrupts.
    newPriority ← (awakePriorities land: tempenabled) hibit.
    (newPriority = currentPriority)
      and: (currentPriority = priority)⇒
         [enabledPriorities ← tempenabled.
          sourceIndirect run: (initialContexts↑priority) cleancopy]
    suspendedContexts↑priority ← (initialContexts↑priority) cleancopy.
    newPriority = 0⇒[enabledPriorities ← tempenabled.]
    newPriority = currentPriority⇒[enabledPriorities ← tempenabled.]
    newContext ← suspendedContexts↑newPriority.
    suspendedContexts↑newPriority ← nil.
    oldPriority ← currentPriority.
    currentPriority ← newPriority.
    enabledPriorities ← tempenabled.
    sourceIndirect
      swap: newContext
      with: (suspendedContexts ref: oldPriority)]
initsched
   [Top ← self fromSource: (PriorityInterrupt new).
```

```
    self init1.
    self init11.
    self init8.
    self init9.
    Top top]
install: newContext above: priority | i
   ["Install a process in the next empty level above <priority> which is initialized from <newContext>
(a remote Context). If there is no empty level above that priority, tell the user and return false"
    newContext sender ← nil.
    for: i from: (priority+1 to: 16) do:
      [(usedPriorities land: biton•i) = 0⇒
        [↑self
          INSTALL: [while: true do: [newContext cleancopy eval]]
          AT: i]]
    user show:
      'PriorityScheduler unable to install above level '
        +priority asString
        +'. false returned'. ↑false]
INSTALL: newContext AT: priority
   ["This message should be protected, it is used by install:at: and install:above: to do the actual
initialization of the process"
    newContext sender ← nil.
    usedPriorities ← usedPriorities lor: biton•priority.
    initialContexts•priority ← newContext.
    suspendedContexts•priority ← newContext cleancopy.
    ↑PriorityInterrupt new from: self at: priority]
install: newContext at: priority
   ["Install a process at level <priority> which is initialized from <newContext> (a remote Context). If
there is already a process at that priority, tell the user and return false"
    newContext sender ← nil.
    (usedPriorities land: biton•priority) = 0⇒
      [↑self
        INSTALL: [while: true do: [newContext cleancopy eval]]
        AT: priority]
    user show:
      'PriorityScheduler unable to install at level '
        +priority asString
        +'. false returned'. ↑false]
reselect
      | newPriority oldPriority newContext tempenabled tempinterrupts
   ["Switch to the highest priority enabled process"
    tempenabled ← self disable.
    tempinterrupts ← interruptedPriorities land: awakePriorities.
    awakePriorities ← interruptedPriorities lor: awakePriorities.
    interruptedPriorities ← tempinterrupts.
    newPriority ← (awakePriorities land: tempenabled) hibit.
    newPriority = 0⇒[enabledPriorities ← tempenabled. ↑false]
    newPriority = currentPriority⇒[enabledPriorities ← tempenabled]
    newContext ← suspendedContexts•newPriority.
    suspendedContexts•newPriority ← nil.
    oldPriority ← currentPriority.
    currentPriority ← newPriority.
    enabledPriorities ← tempenabled.
    sourceIndirect
      swap: newContext
      with: (suspendedContexts ref: oldPriority)]
  primitive: 65
reset: priority
   ["Initialize the state of the process at <priority> and request it to cease running"
    awakePriorities ← awakePriorities land: bitoff•priority.
    self init: priority]
restart: priority
   ["Initialize the state of a suspended process and request it to run"
    interruptedPriorities ← interruptedPriorities lor: biton•priority.
    self init: priority]
run: newContext
      at: priority
   ["replace the process at <priority> with <newContext>. If that is the currently running priority,
abandon what is running and start from <newContext>"
    priority = currentPriority⇒
      [sourceIndirect run: newContext]
```

```
    suspendedContexts*priority ← newContext]
sleep: priority
    ["Request the process at <priority> to cease running, if a new wakeup has arrived the process will
be reawakened. Turn off the corresponding bit in awakePriorities and check if that changes who should
run"
    awakePriorities ← awakePriorities land: bitoff*priority.
    self reselect]
swap: newContext
        at: priority
        with: fieldReference
    ["replace the process at <priority> with <newContext> and place the old contents in the field
referred to by <fieldReference>"
    priority = currentPriority⇒
      [sourceIndirect
          swap: newContext
          with: fieldReference]
    fieldReference value← suspendedContexts*priority.
    suspendedContexts*priority ← newContext]
terminate: priority
    ["Remove a process from the scheduler, allowing that level to be reused"
    enabledPriorities ← enabledPriorities land: bitoff*priority.
    suspendedContexts*priority ← initialContexts*priority ← nil.
    awakePriorities ← awakePriorities land: bitoff*priority.
    interruptedPriorities ← interruptedPriorities land: bitoff*priority.
    usedPriorities ← usedPriorities land: bitoff*priority.
    self reselect]
top
    ["Make this scheduler the top level one. It will receive all physical (non-Smalltalk) interrupts"
    enabledPriorities ← enabledPriorities lor: biton*1.
    awakePriorities ← awakePriorities lor: biton*1.
    currentPriority ← 1] primitive: 61
wakeup: priority
    ["Request the process at <priority> to run. Turn on the corresponding bit in interruptedPriorities
and check if that changes who should run"
    interruptedPriorities ← interruptedPriorities lor: biton*priority.
    self reselect]
!
```

Changes
from 3.5 to 3.7

BitRect asFollows!

How to Use

```
AComment
    ["BitRect is a Rectangle that remembers the bits within it.
    To create and edit one, say:
        (BitRect new fromuser) edit.
    (This installs a BitRectEditor in the scheduler)
    "]
```

Printing and Access

```
scaffold
 ["(cc ← BitRectEditor new) init.
(aa ← BitRect new) fromuser; filin: (☞actionpic asString); show.
cc actionpic: aa.
(aa ← BitRect new) fromuser; filin: (☞toolpic asString); show.
cc toolpic: aa.
(aa ← BitRect new) fromuser; filin: (☞sample asString); edit.
"]
!
```

Class asFollows!

Editing

```
edit: selector | para s v
    [para ←
        [selector=☞ClassOrganization⇒
            [self organization asParagraph]
        messagedict has: selector⇒[self code: selector]
        nullString asParagraph].
    self edit: selector para: para formerly: false]
edit: selector para: para formerly: oldpara | codePane panedWindow
    [codePane ← CodePane new class: self selector: selector para: nil.
    panedWindow ← PanedWindow new title: title+' . ' + selector
        with: codePane inVector at: SingleFrameFrames.
    panedWindow newframe.
    codePane showing: para; formerly: oldpara; from: codePane.
    user schedule: panedWindow]
```

Initialization

```
fields: myinstvars        "list of instance variables"
    [  [self≠self realself⇒
            [self instvars≠self realself instvars⇒
                [user notify: 'All '+title+'s become obsolete if you proceed...'.
                self realself obsolete.        "if he chooses to proceed"
                Smalltalk·title unique ← self]
            self realself environment← nil; myinstvars← myinstvars; subclassof: superclass.
            ↑self]].
    fieldtype ← 16.
    instsize ← self instvars length.
    instsize>16⇒
        [user notify: 'too many instance variables']
    messagedict ← MessageDict init.
    self organization]
myinstvars ← myinstvars
!nil
```

File asFollows!

```
compile: parag        "rewrites file with contents of edit window"
    [self reset; append: parag; shorten; flush]
edit | codePane panedWindow
    [codePane ← CodePane new.
    panedWindow ← PanedWindow new title: filename asString
        with: codePane inVector at: SinglePaneFrames.
    panedWindow newframe.
    codePane showing: self contents asParagraph; from: self.
    user schedule: panedWindow]
execute: parag    "doit in file edit"
    [↑nil·sparag]
!
```

Integer asFollows!

Subscripts

```
subscripts: a
    [self≥1 and: self≤a length⇒[↑a•self]
    user notify: 'Subscript out of bounds: ' + self asString]
subscripts: a ← val
    [self≥1 and: self≤a length⇒
        [(a is: String) and: (val isnt: Integer)⇒
            [user notify: 'Improper store (non-char into String?)']
        ↑a•self ← val]
    user notify: 'Subscript out of bounds: ' + self asString]
!
```

Object asFollows!

Comparison

```
@ x [↑self@x "In case this is reached by perform:"] primitive: 4
```

System Primitives

```
perform: selector      "Send the unary message, selector, to self"
    [selector mustTake: 0. ↑self performDangerously: selector]
perform: selector with: arg1  "Send the 1-argument message, selector, to self"
    [selector mustTake: 1. ↑self performDangerously: selector with: arg1]
perform: selector with: arg1 with: arg2   "Send the 2-argument message, selector, to self"
    [selector mustTake: 2. ↑self performDangerously: selector with: arg1 with: arg2]
perform: selector with: arg1 with: arg2 with: arg3    "Send the 3-argument message, selector,
to self"
    [selector mustTake: 3. ↑self performDangerously: selector with: arg1 with: arg2 with: arg3]
performDangerously: selector "Send self the message, selector; it had better be unary"
    [user notify: 'can''t perform: nil'] primitive: 102
performDangerously: selector with: arg1  "selector had better take 1 arg"
    [user notify: 'can''t perform: nil with:'] primitive: 102
performDangerously: selector with: arg1 with: arg2  "selector had better take 2 args"
    [user notify: 'can''t perform: nil with:with:'] primitive: 102
performDangerously: selector with: arg1 with: arg2 with: arg3  "selector had better take 3
args"
    [user notify: 'can''t perform: nil with:with:with:'] primitive: 102
startup      "loopless scheduling"
    [self firsttime⇒
        [while§ self eachtime do§ [].
        ↑self lasttime]
    ↑false]
!
```

ParagraphEditor asFollows!

Private Messages

```
checklooks | t
    [(t ← user rawkbck)<0253⇒[↑false];
        =0334⇒[user rawkbd. self toBravo];
        =0337⇒[user rawkbd. self fromBravo].
    loc1≥loc2 ⇒ [↑false]
    (t ← ctlchars find: t)=0⇒[↑false]
    [oldpara⇒[] oldpara ← para copy].
    user rawkbd.
    para changestyle: loc1 to: loc2-1 to: runvals•t]
```

Public Messages

```
realign
    [  [oldpara⇒[] oldpara ← para copy].
    para alignment ← ☞(1 2 4 0 0)•(1+para alignment).
    self show; select]
!
```

Stream asFollows!

Static reading and writing

```
• x
    [↑array•x]
• x ← val
```

```
    [↑array*x ← val]
```

Sequential reading and writing

```
≪ x | y
    [y← self next⇒      "peek for matching element"
        [x=y⇒ [↑y]      "gobble it if found"
        position ← position-1. ↑false]
    ↑false]
append: x | i    "Array arg"
    [for: i from: x do:
        [self next ← i].
    ↑x]
```

Coercions

```
asStream
asVector "Convert a string to a vector of tokens"
    [↑(Reader new of: self) read]
```

Static reading and writing

```
contents | a i "↑(array*(1 to: position)) copy -- written out for speed"
    [a← array species new: position.
    for: i to: position do:
        [a*i← array*i].
    ↑a]
```

Character printing

```
cr
    [self next ← 015]
ctlz: chars
    [self next← 032; append: chars; next← 015]
```

Initialization

```
default
    [self of: (String new: 8)]
```

Sequential reading and writing

```
dequeue      "use it as a FIFO"
    [↑self dequeue: 1]
dequeue: n | t
    [position<n⇒ [↑false]
    t ← (array*(1 to: n)) copy.
    array*(1 to: position-n) ← array*(n+1 to: position).
    position ← position-n. ↑t]
```

Test and alter position

```
empty        "for"
    [↑position=0]
end
    [↑position≥limit]
```

Sequential reading and writing

```
into: x | i        "Array result"
    [for: i to: x length do:
        [x*i ← self next].
    ↑x]
```

Static reading and writing

```
last
    [↑array*position]
last: n
    [↑(array*(position-n+1 to: position)) copy]
```

Test and alter position

```
loc    "synonym for compiler"
    [↑position]
myend
    [↑position≥limit]
```

Sequential reading and writing

```
next  "simple result"
    [self myend⇒ [↑self pastend]
    ↑array*(position ← position+1)] primitive: 17
next ← x      "simple arg"
    [self myend⇒ [↑self pastend ← x]
    ↑array*(position ← position+1) ← x] primitive: 18
```

Initialization

```
of: array
```

```
    [position ← 0. limit ← array length]
of: array from: position to: limit
    [position ← position-1]
```

Test and alter position

```
pastend
    [⇑false]
pastend ← x
    [array ← array grow. limit ← array length.
    ⇑self next ← x]
```

Sequential reading and writing

```
peek | x
    [x← self next⇒ [position ← position-1.  ⇑x]   "peek at next element"
    ⇑false]
pop       "use it as a LIFO"
    [position<1⇒ [⇑false]
    position ← position-1. ⇑array·(position+1)]
pop: n | t
    [position<n⇒ [⇑false]
    t ← self last: n.
    position ← position-n. ⇑t]
```

Test and alter position

```
position
    [⇑position]
```

Character printing

```
print: obj
    [obj printon: self]
```

Test and alter position

```
reset
    [position ← 0]
```

Character printing

```
semicrtab
    [self append: ';
    ']
```

Test and alter position

```
skip: x
    [position ← position+x]
```

Character printing

```
space
    [self next ← 040]
tab
    [self next ← 011]
```

Sequential reading and writing

```
upto: x | y s
    [s ← Stream default.
    for□ y from: self do□
        [y=x⇒[⇑s contents]
        s next ← y]
    ⇑s contents]
□
```

String asFollows!

```
length [⇑self length "In case this is reached by perform:"]
!
```

SymbolTable asFollows!

```
allRefsTo: symbol from: classNames | className s r w
    [s ← Stream default. r ← self ref: symbol.
    user displayoffwhile□ [for□ className from: classNames do□
        [w ← self·className whosends: r. w length=0⇒ []
        s append: className;space; append: w;cr]].
    ⇑s contents]
!
```

UniqueString asFollows!

```
mustTake: nargs "fatal error if I am not a selector that takes nargs arguments"
```

```
    [self numArgs≠nargs⇒
        [user notify: self + ' does not take ' + nargs asString + ' arguments']]
numArgs  | len n i "the number of arguments I take when I am a selector"
    [len ← self length.
    len=1⇒ [⇑[(self˙1) isletter⇒ [0] 1]]
    n ← 0. "count colons"
    for§ i to: len do§ [self˙i=072⇒ [n←n+1]; =03⇒ [n←n+1]].
    ⇑n]
!
```

UserView asFollows!

Misc System Stuff

```
Version [⇑'3.7 September 16']
    "
    You are invited to comment your favorite class.
        Then say (ClassName filoutOrganization).  Give the resulting '.org' file to Ted.
    classes Date and Time defined (in 'Numbers' category)
    fixed some of the highlighting bugs in code panes
    scroll bar shown whenever a pane is awake and is not in the midst of a task:
        so star-cursor does not appear during a doit; but a temporary, harmless 'bug':
            proceed from an error will sometimes leave scrollbars displayed wrong
    class CodePane (with cancel feature) supersedes now-defunct class CodeEditor
    formatting changes now are considered edits requiring compile or cancel
    streams now refuse to initialize limit > array length
    stream contents is faster
    best way to init class variables is a msg you call 'classInit':
        filout of ClassWhatever puts out 'ClassWhatever classInit' for filin time
    computed selectors (perform:, perform:with:, etc.) are implemented (see manual)
    - - -
    freelist time bomb for Vector 8 patched (cause still unknown ...)
    growing enabled again.
    mouse scrolling inverted
        goes up from bottom, down from top
        in text, adjacent line will jump to middle
    proceed now works in debugger (when NotifyFlag is true)
    # now works in debugger (when NotifyFlag is false)
    - - -
    Keyset cursor bug fixed
    Class.filoutOrganization fixed
    Undeclared cleaned out
        (Array.replace, Dispframe.eachtime and UserView.unschedule
        were all sharing the undeclared variable t!)
    - - -
    Scroll bars controlled by mouse (keyset still works)
        redbug does it, dist from center sets speed
    Several debugger bugs fixed.
    Faster cursor show
    Filin prints selector names again
    Group filout puts headings at page tops as originally intended
    "
!
```

Vector asFollows!

Misc

```
length "This is actually done in microcode"
    [⇑self length "perform: needs this"]
!
```