"Class"

```
Class new title: 'Class'
   subclassof: Object
   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'
   declare: '';
   veryspecial: 1;
   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]]
classInit       "gets propagated to a dummy instance"
   [self new classInit]
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 | r a b          "list of instance variables"
   [messagedict ← MessageDict init.
   r ← self realself.
      [self≠r⇒
         [(a← self instvars)≠(b← r instvars)⇒
            [[r howMany>0⇒
```

```
                [user notify: 'All '+title+'s become obsolete if you proceed...']].
            [a length>b length and: a∘(1 to: b length)=b⇒          "just adding new
inst fields"
                [classvars ← r classvars.          "so copy classvars"
                messagedict ← r md copy.          "and messages"
                r md init]          "keep obsoleting from recycling"
            user notify: title+ ' methods and comments disppear if you
proceed...'].
        r obsolete.
        Smalltalk∘title unique ← self]          "make this the real one"
      r environment← nil; myinstvars← myinstvars; subclassof: superclass.
      ↑self]].
  fieldtype ← 16.
  instsize ← self instvars length.
  instsize>256⇒
    [user notify: 'too many instance variables']
  self organization]
myinstvars ← myinstvars
obsolete          "invalidate further communication"
  [title ← 'AnObsolete'+title.
  classvars ← nil.          "recycle class variables"
  messagedict close.          "invalidate and recycle local messages"
  environment ← self.          "keep me around for old instances".
  superclass ← Object.          "invalidate superclass messages"]
realself [↑Smalltalk∘title unique]          "as opposed to possible filin ghost"
rename: newtitle
    | name newname oldclass category
  [name ← title unique. newname ← newtitle unique.
    [Smalltalk has: newname⇒
      [oldclass ← Smalltalk∘newname.
      user notify: 'All ' + newtitle + 's will become obsolete if you proceed'.
      oldclass obsolete]
    category ← SystemOrganization invert: name.
    AllClassNames ← AllClassNames insertSorted: newname.
    SystemOrganization classify: newname under: category].
  Smalltalk delete: name.
  AllClassNames ← AllClassNames delete: name.
  SystemOrganization delete: name.
  title ← newtitle.
  Smalltalk declare: newname as: self]
sharing: table
  [self≠self realself⇒[self realself sharing: table]
  environment ← environment asVector , table]
subclassof: supercl
  [[(supercl is: Class) or: (supercl is: VariableLengthClass)⇒[]
    user notify: 'Superclass is not yet defined or not a Class'].
  messagedict≡nil⇒ [superclass ← supercl] "Still defining"
  supercl instvars≠superclass.instvars⇒
    [user notify: 'New superclass is not compatible with old']
  superclass ← supercl]
title: title
  [self title: title unique insystem: Smalltalk]
```

```
title: name insystem: system | cl
   [superclass ← Object.
   [system has: name⇒
      [cl ← (system∘name) class.
      cl≡self class⇒ [↑self]
      user notify: name + ' will change from a ' + cl title + ' to a ' + self class
title + ' if you proceed...']].
   system declare: name as: self.
   AllClassNames ← AllClassNames insertSorted: name.
   SystemOrganization classify: name under: 'As yet unclassified']
title: t subclassof: s fields: f declare: d
   [t∘1≠((t∘1) asUppercase)⇒
      [user notify: 'Please capitalize each word in class title: ' + t. ↑false]
   self title: t; subclassof: s; fields: f; declare: d]
veryspecial: n          "inaccessible fields"
   [instsize ← self instvars length + n]
```

Access to parts

```
∘x [↑classvars∘x]
∘x ← val [↑classvars∘x ← val]
fieldNamesInto: collector
   [[superclass≡nil⇒ [] superclass fieldNamesInto: collector].
   ↑(Reader new of: myinstvars) readInto: collector]
instsize
["Return the number of user accessable instance fields (self instvars length)."
↑[fieldtype≥32⇒ [0]
   self≡Class⇒ [instsize-1]
   self≡VariableLengthClass ⇒[instsize-20]
   instsize]]
instvars
   [↑self fieldNamesInto: FieldNameCollector default]
invertRef: refs "Refs may be a vector (to allow batching)"
      | cl env source ref inv sym t
   [refs isnt: Vector⇒ [↑(self invert: refs inVector)∘1]
   env ← (self environment concat: (Undeclared, Smalltalk)) asStream.
   source ← Dictionary init.
   ↑refs transform: ref to:
      [cl ← self. env reset.
         until:
            [(sym ← env next)≡false⇒ [inv ← 'unknown ' concat: ref asOop
base:8]
            [cl≠nil and: sym≡cl classvars⇒ [t ← cl title. cl ← cl superclass] t ←
false].
            (inv ← sym invertRef: ref)≡false⇒ [false]
            [t⇒ []
             t ← source lookup: sym⇒ []
             source insert: sym with: (t ← Smalltalk invert: sym)].
            inv ← (t concat: ' ') concat: inv]
         do: [].
      inv]
   ]
isa: x  "is x on my superclass chain?"
```

```
   [superclass ≡ x ⇒[↑true]; ≡ nil ⇒[↑false]
   ↑superclass isa: x]
md [↑messagedict]
superclass [↑superclass]
title [↑title]
```

Organization

```
classvars [↑classvars]
clean | name      "release unreferenced classvars"
   [for: name from: classvars do:
      [name≠↗ClassOrganization and: (classvars ref: name) refct=1⇒
         [classvars delete: name]]]
environment
   [↑(classvars asVector concat: environment asVector) concat:
      [superclass≡nil⇒ [↗()] superclass environment]]
organization | o
   [   [classvars ≡ nil⇒[self declare: ↗ClassOrganization]].
   o ← classvars lookup: ↗ClassOrganization.
   o is: ClassOrganizer⇒[↑o]
   o ← ClassOrganizer new init: messagedict contents sort.
   classvars insert: ↗ClassOrganization with: o.  ↑o]
```

Editing

```
ed: selector | c s
   [c← self code: selector. user clearshow: c.
   while: (s← user request: 'substitute: ') do:
      [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].
   self edit: selector para: para formerly: false]
edit: selector para: para formerly: oldpara
   [user topWindow leave.
   user restartup: (CodeWindow new class: self selector: selector para: para
formerly: oldpara)]
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]].
   for: selector from: messagedict do:
      [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⇒ [↑self]
   superclass≡nil⇒ [↑false]
   ↑superclass canUnderstand: selector]
canunderstand: selector
   [↑messagedict has: selector]
code: selector | c
   [selector=↷ClassOrganization⇒
         [↑self organization asParagraph]
   ↑Paragraph new unpackFromString: (messagedict code: selector)]
compileall | s
   [for: s from: messagedict do: [self understands: (self code: s)]]
"to recompile the whole system (check out big changes) execute:
   | n [for: n from: AllClassNames do:
      [user show: n; cr. (Smalltalk∘n) compileall.
      Changes init. MessageDict new freeMethods]] "
compressAll | s
   [user displayoffwhile:
      [self organization compress.
      for: s from: messagedict do:
         [(messagedict code: s)∘1<16⇒
            [messagedict code: s ← (self code: s) compressIntoString]]]]
copy: sel from: class
   [self copy: sel from: class classified: nil]
copy: sel from: class classified: cat "Useful when modifying an existing class"
      | s code
   [sel is: Vector⇒ [for: s from: sel do: [self copy: s from: class classified: cat]]
   sel is: String⇒ [self copy: (class organization category: sel) from: class
classified: cat]
   code ← class code: sel.  code≡nil⇒ []
   [cat≡nil⇒ [cat ← class organization invert: sel]].
   [messagedict has: sel⇒
      [code text=(self code: sel) text⇒ []
      user notify: title+' '+sel+' will be redefined if you proceed.']].
   self understands: code classified: cat]
derstands: selector | c          "overstands? undersits? - forget it"
   [messagedict ← messagedict delete: selector.
   self organization delete: selector.
   Changes has: (c←title+' '+selector)⇒ [Changes delete: c]]
describe: method on: strm | sel cls  "append mclass and selector"
   [cls ← self.
   until: [cls≡nil⇒ [cls←self. sel←↷?] sel ← cls md invert: method] do:
      [cls ← cls superclass].
   strm append: cls title; space; append: sel]
install: name method: method literals: literals
      code: code backpointers: backpointers
   [messagedict ← messagedict insert: name method: method
      literals: literals code: code makeBoldPattern packIntoString
```

```
      backpointers: backpointers.
   Changes insert: title+' '+name]
messages [↑messagedict contents , ↗ClassOrganization]
method: sel
   [↑messagedict methodorfalse: sel]
notify: errorString at: position in: stream
   [↑self notify: errorString at: position in: stream for: self]
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"
   [↑self understands: code classified: 'As yet unclassified']
understands: code classified: heading          "compile and install method"
   [↑Generator new compile: code asParagraph asStream
      in: self under: heading notifying: self]
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 | indx vec PCLs i "returns a vector containing all instances of this
class mixed with nils"
   ["Works for all classes.  Some additional instances may be created after the
   vector is filled but before you get to use it."
   PCLs ← Vmem pclassesOf: self.  "vector of PCLs"
   vec ← Vector new: 128*PCLs length.
   for: i to: PCLs length do:
      [(vec∘[i-1*128+1 to: i*128]) all← PCLs∘i].
   thisContext destroyAndReturn: (self fromFreelist: Class instsize fill: vec)]
copy: inst | t i
   [t ← self new.
   for: i to: self instsize do:
      [t instfield: i ← inst instfield: i]
   ↑t]
default
   [↑self new default]
fromFreelist: i fill: vec "i = zero order index of freelist in class instance.
   vec = vector in pclasses of all possible instances."
   [user croak] primitive: 60
howMany | v "how many instances of this class are in use now?"
   [v ← self allInstances.
   thisContext destroyAndReturn: v length-(v count: nil)]
init       "init and default get propagated to instances"
   [↑self new init]
init: n       "init and default get propagated to instances"
```

```
   [↑self new init: n]
instfield:    i    "prevent user from getting freelist"
   [i > Class instsize ⊃[user notify: 'arg too big']
   ↑super instfield: i]
new [user croak] primitive: 28
new: length "To allow fixed-length classes to simulate variable-length ones"
   [↑self new init: length] "By convention"
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: self instsize do:
      [t instfield: i ← (inst instfield: i) recopy]
   ↑t]
```

Filin and Filout

```
asFollows | s heading trailer selector
   [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]
      trailer='\f5bg'⊃[heading ← s]
      s ← s asParagraph applyBravo: trailer at: 1 to: s length.
      self canunderstand: (selector ← [heading ≡ nil⊃[self understands: s]
            self understands: s classified: heading])⊃
               [user show: selector; space. messagedict purge: selector]
      user show: '(an uncompiled method) ']
   FilinSource upto: 015]
changelist: cat | a
   [↑(self organization category: cat) transform: a to: title+' '+a]
definition | strm        "return a string that defines me (Class new title etc.)"
   [strm ← (String new: 50) asStream.
   self printdefon: strm.
   ↑strm contents]
endCategoryOn: pstrm
endChangesOn: pstrm
   [pstrm print: '⌟' asParagraph]
filout
   [user displayoffwhile:
      [(dpo file: title+'.st.') filoutclass: self.
      self noChanges]]
filoutCategory: cat
```

```
    [user displayoffwhile:
        [(dpo file: (title+'-'+cat+'.st') asFileName) filout: (self changelist: cat)]]
filoutOrganization | f          "So we can merge separate work on organization"
    [user show: title; cr.
    user displayoffwhile:
        [f ← dpo file: title+'.org.'.
        f append: title+' organization fromParagraph:'; cr.
        f append: self organization asParagraph text asString.
        f append: 'asParagraph_!'; shorten; close]]
noChanges | s t
    [t← title+' *'.
    for: s from: Changes contents do:
        [t match: s⇒
            [Changes delete: s]]]
paraprinton: strm          "Strm is actually a ParagraphPrinter"
        | para frame s heading org
    [user show: title; cr.
    para ← ('"'+title+'"') asParagraph.
    para maskrunsunder: 0361 to: 0121.          "Font ← 5, Bold"
    frame ← strm defaultframe.
    strm frame ← 15000⊙frame origin y rect: 20000⊙frame corner y.
    strm print: para.
    strm frame ← frame.
    strm print: ((self definition+';
    asFollows_!') asParagraph maskrunsunder: 0361 to: 0121).
    org ← self organization.
    strm print: (('
'+org globalComment) asParagraph maskrunsunder: 2 to: 2).          "Italic"
    for: heading from: org categories do:
        [self printCategory: heading on: strm]
    self endChangesOn: strm.
    strm print: ('SystemOrganization classify: ↗'+title+' under: '''+
        (SystemOrganization invert: title unique)+'''._!') asParagraph.
    [self ≡ Class or: self ≡ VariableLengthClass⇒ [] self canunderstand:
↗classInit⇒
        [strm print: (title+' classInit_!') asParagraph]].
    ]
printCategory: s on: pstrm | sel
    [self startCategory: s on: pstrm.
    for: sel from: (self organization category: s) do:
        [self printMethod: sel on: pstrm].
    self endCategoryOn: pstrm]
printdefon: strm | s          "print my definition on strm"
    [strm append: self class title; append: ' new title: '; append: title asString.
    strm cr; tab; append: 'subclassof: ' +
            [superclass≡nil⇒['nil'] superclass title].
    strm cr; tab; append: 'fields: ' + myinstvars asString.
    strm cr; tab; append: 'declare: '''.
    for: s from: classvars contents do:
            [s=↗ClassOrganization⇒[]
            strm append: s; space]
    strm append: ''''.
```

```
    [fieldtype=16⇒[]
        strm semicrtab; append: 'bytesize: '; print: fieldtype-32].
    [instsize = (s← self instvars) length⇒[]
        strm semicrtab; append: 'veryspecial: '; print: instsize-s length].
    [environment≡nil⇒[]
        for: s from: environment do:
            [strm semicrtab; append: 'sharing: '+ (Smalltalk invert: s)]]]
printMethod: sel on: pstrm
    [pstrm print: (self code: sel).
    messagedict purge: sel]
printout
    [user displayoffwhile: [
        PressPrinter init of: (dpO file: title+'.press.'); stamp;
        printclass: self; close]]
printoutCategory: cat
    [user displayoffwhile:
        [(dpO file: (title+'-'+cat+'.press') asFileName) printout: (self changelist:
cat)]]
readfrom: strm [↑self readfrom: strm format: nil]
readfrom: strm format: f [
    ↑self new readfrom: strm format: f]
startCategory: s on: pstrm
    [pstrm print: (('
'+s) asParagraph maskrunsunder: 0361 to: 0121).          "Font 5, Bold"
    ]
startChangesOn: pstrm
    [pstrm print: (('
'+title+' asFollows_↓') asParagraph maskrunsunder: 0361 to: 0121).          "Font 5,
Bold"
    ]
_↓
SystemOrganization classify: ↗Class under: 'Kernel Classes'._↓
```

```
                                                        "Context"
Class new title: 'Context'
   subclassof: Object
   fields: 'sender "<Context> from which this message was sent"
      receiver "<Object> to which this message was sent"
      keep "<true, nil> nil means reclaimable"
      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"'
   declare: '';
   asFollows_!
```

A context keeps track of the progress of a method

Initialization

```
cleancopy | i
   [↑Context new
      sender: sender
      receiver: receiver
      mclass: mclass
      method: method
      tempframe: tempframe copy
      pc: pc
      stackptr: stackptr]
sender: sender receiver: receiver mclass: mclass
      method: method tempframe: tempframe pc: pc stackptr: stackptr
```

Access to parts

```
mclass | sel cls "Find the class in which the code for this Context was found"
   [sender≡nil⊃[↑receiver class]
    sel ← sender thisop.
    cls ← receiver class.
    until: cls≡nil do:
      [(cls method: sel)≡method⊃[↑cls]
       cls ← cls superclass]
    ↑receiver class]
method
   [↑method]
receiver
   [↑receiver]
sender [↑sender]
sender← sender []
stackIndex "Return the subscript in tempframe of my top of stack."
   [↑stackptr+1]
swapSender: coroutine | oldSender
   [oldSender ← sender. sender ← coroutine. ↑oldSender]
tempframe
   [↑tempframe]
```

Control structures

```
for: var1 from: expr1 with: var2 from: expr2 do: stmt | s1 s2
    [s1 ← expr1 asStream. s2 ← expr2 asStream.
    while: [(var1 value ← s1 next) and: (var2 value ← s2 next)] do: stmt eval]
```

Debugging

```
debug | t v
    [self print.
    while: [user cr. t ← user request: '*'] do: "until ctrl-d"
        [v ← Generator new evaluate: t asStream in: false to: self notifying: nil.
        ↱debugret=v⇒[self print] v print]
    ∩↱debugret]
printon: strm | mc
    ["Print the selector which invoked this Context
      and the class in which code was found for that selector"
     mc ← self mclass.
     strm append: mc title. sender≡nil⇒ []
     [receiver is: mc⇒[] strm append: '('+receiver class title+')'].
     strm append: '⇒'; print: sender thisop]
restartWith: method
    [tempframe ← tempframe copy: 1 to: method∘3.
    self restart]
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.]
thisop | a
    [a ← method∘pc.
    a≥0320⇒ [∩self litof: a-0320]
    a≥0260⇒ [∩SpecialOops∘(10+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]
variableNamesInto: dest with: block | class selector parser
    ["For each method variable name, call
          dest declaration: block name: string asArg: <true or false>
      If cant find source code, call dest notify: "
    class ← self mclass.
    selector ← class md invert: method⇒
        [parser ← Parser new from: (class code: selector) asStream to: dest.
        parser pattern: block; temporaries: block; terminate]
    dest notify: 'thisContext is not running a currently defined method']
verifyFrames | c "be sure frames on stack aren't nil"
    [c ← self.
     until: c≡nil do:
```

```
    [c tempframe≡nil⇒
      [user notify: 'Sorry, that stack has been released -- proceeding is
impossible'; restart]
     c ← c sender]]
```

Simulation

```
docode: code toclass: class | i v
  [[(i←code∘2)≠ 0 ⇒
    [i = 1  ⇒ [↑self];
      = 30 ⇒ [v ← self pop. v push: self. ↑v];
      = 40 ⇒ [v ← self pop instfield: code∘5 + 1. ↑self push: v];
      = 41 ⇒ [user notify: 'Field← primitive unimplemented'];
      = 101 ⇒ [user notify: 'Doprimitive unimplemented'];
      = 102 ⇒ [↑self performing: code∘4 toclass: class]
     v ← self doprimitive: code⇒
      [stackptr ← stackptr-(code∘4+1). ↑self push: v]]].
   ↑self newToRun: code]
dojump: displacement
  [pc ← pc+displacement]
dopop
  [stackptr ← stackptr-1]
doprimitive: code
  [↑false] primitive: 101
doremotereturn | t f
  [t ← self pop. f ← self pop. ↑f push: t]
doreturn | t
  [t ← tempframe∘(stackptr+1). tempframe ← nil. ↑sender push: t]
dostore | byte
  [byte ← method∘(pc← pc+1).
   byte<020⇒ [self smashField: byte];
        <040⇒ [self smashTemp: byte-020];
        <0100⇒ [user notify: 'Store into literal'];
        <0160⇒ [self smashLitInd: byte-0100];
        <0170⇒ [self instfield: (byte-0157) ← tempframe∘(stackptr+1)];
        =0210⇒ [self smashField: self nextByte];
        =0211⇒ [self smashTemp: self nextByte];
        =0213⇒ [self smashLitInd: self nextByte]
   user notify: 'Illegal store'
  ]
dosuper | byte
  [byte ← self nextByte.
  [byte=0214⇒ [byte ← self nextByte+0320]].
  byte<0260⇒ [user notify: 'non-selector after super']
  ↑self sendmess: nil byte: byte toclass: (self litof: method∘6-8/2) value
superclass]
litof: a
  [↑(method word: a+4) asObject]
newToRun: code | r v i
  [r ← self pop. v ← Vector new: code∘3.
   forg i from: (code∘4 to: 1 by: ¯1) do "Move args to new tframe"
     [v∘i ← tempframe∘(2+(stackptr ← stackptr-1)).
     tempframe∘(stackptr+2) ← nil       "Nil args on caller's stack"
```

```
      ].
    ↑self class new sender: self receiver: r mclass: nil method: code
      tempframe: v pc: code∘6 stackptr: code∘5 - 1. "Allocate a new Context"
    ]
nextByte
    [↑method∘(pc ← pc+1)]
nonEmptyStack
    [↑(stackptr≠¯1)]
performing: nargs toclass: class | i sel
    [i ← stackptr-nargs.
    sel ← tempframe∘(i+1).      "Selector is first arg"
    while: i<stackptr do:
      [tempframe∘(i+1) ← tempframe∘(i+2).
      i ← i+1].
    stackptr ← stackptr-1.
    ↑self sendmess: sel byte: 0320 toclass: class]
pop
    [↑tempframe∘(2+ (stackptr← stackptr-1))]
push: n "Push n on top of stack"
    [tempframe∘(1+(stackptr←stackptr+1)) ← n]
pushField: i
    [tempframe∘(1+(stackptr←stackptr+1)) ← receiver instfield: i+1]
pushLit: i
    [tempframe∘(1+(stackptr←stackptr+1)) ← (method word: i+4) asObject]
pushLitInd: i
    [tempframe∘(1+(stackptr←stackptr+1)) ← (method word: i+4) asObject
value]
pushTemp: i
    [tempframe∘(1+(stackptr←stackptr+1)) ← tempframe∘(i+1)]
restart
    [pc ← method∘6. stackptr ← method∘5-1]
runsimulated: cntxt | ctx
    [cntxt push: self. ctx ← cntxt.
    until: ctx≡self do: [ctx ← ctx step].
    ↑self pop]
sendmess: sel byte: byte toclass: class | cl code
    [cl ← class. until: cl≡nil do:
      [[byte<0320⇒ "Is it a special message?"
        [self specialmess: byte toclass: cl⇒ [↑self] "Can we perform it?"
          sel ← SpecialOops∘(byte-0246)] "Send the special message selector"
        sel≡nil⇒ [sel ← self litof: byte-0320]]. "Send the selector from the literals"
       code ← cl md methodorfalse: sel⇒ [↑self docode: code toclass: cl]
        cl ← cl superclass]. "Try up the superclass chain"
     user notify: ('Simulated message not understood: ' concat: sel asString)]
simulate: cntxt
    [cntxt push: thisContext. ↑cntxt inspect]
smashField: i
    [receiver instfield: i+1 ← tempframe∘(stackptr+1)]
smashLitInd: i
    [(self litof: i) value ← tempframe∘(stackptr+1)]
smashTemp: i
    [tempframe∘(i+1) ← tempframe∘(stackptr+1)]
```

```
specialmess: byte toclass: class | r n d i s l
   [byte<0270⇒
      [class ≡ Integer⇒
         [r ← self pop. n ← self pop.
          r class ≡ Integer⇒
            [↑self push: [byte
               =0260⇒ [r+n];  =0261⇒ [r-n];
               =0262⇒ [r<n];  =0263⇒ [r>n]; =0264⇒ [r≤n];  =0265⇒ [r≥n];
               =0266⇒ [r=n];  =0267⇒ [r≠n]]]
          stackptr ← stackptr+2. ↑false]
       ↑false];
      <0300⇒[↑false];
      <0304⇒
         [d ← stackptr.  r ← self pop. [(byte land: 1)=1⇒[n ← self pop]].
          [byte<0302⇒
            [[class≡Vector⇒[] class≡String⇒[] stackptr ← d. ↑false].
             i ← self pop. l ← r length]
           class≡Stream⇒
            [s ← r. i ← s position+1. l ← s limit. r ← s asArray.
             [r class≡Vector⇒[]; ≡String⇒[]; ≡Interval⇒[] stackptr ← d. ↑false].
            ]
            stackptr ← d. ↑false].
          [(i class ≡ Integer) and: (l class ≡ Integer)⇒
            [1≤i and: i≤l⇒
               [ [(byte land: 1)=1⇒[r0i ← n]]. self push: r0i.
                 [byte>0301⇒[s position ← i]]. ↑self]]].
          stackptr ← d. ↑false];
      =0304⇒ [ [class≡String⇒[] class≡Vector⇒[] ↑false]. self push: (self pop
length)];
      =0305⇒ [self push: self pop≡self pop]
   ↑false]
step | byte t "Execute the next byte code and return the current Context"
   [byte ← method∘(pc ← pc+1).
   ↑[byte<0200⇒ "load"
         [byte<020⇒ [self pushField: byte];
            <040⇒ [self pushTemp: byte-020];
            <0100⇒ [self pushLit: byte-040];
            <0160⇒ [self pushLitInd: byte-0100];
            <0170⇒ [self push: (self instfield: byte-0160+1)]
          self push: SpecialOops∘(byte-0170+2)];
      <0220⇒
         [byte=0200⇒ [self dostore; dopop];
            =0201⇒ [self dostore];
            =0202⇒ [self dopop];
            =0203⇒ [self doreturn];
            =0204⇒ [self doremotereturn];
            =0205⇒ [self push: self];
            =0206⇒ [self dosuper];
            =0210⇒ [self pushField: self nextByte];
            =0211⇒ [self pushTemp: self nextByte];
            =0212⇒ [self pushLit: self nextByte];
            =0213⇒ [self pushLitInd: self nextByte];
```

```
            =0214⇒ [self sendmess: nil byte: self nextByte+0320
                        toclass: (tempframe∘(stackptr+1)) class]
          user notify: 'Unimplemented instruction code'];
        <0260⇒ "jump"
          [t ← [byte<0240⇒[(byte land: 7)+1]
                "short, just mask off bfp bit for displacement"
           (byte land: 7)-4*256+self nextByte].      "long, calculate displacement"
          [(byte land: 010)≠0⇒[self pop≡false⇒[self dojump: t]]
                "bfp, do branch if top is false"
           self dojump: t].     "unconditional, do the branch"
          self]
      self sendmess: nil byte: byte toclass: (tempframe∘(stackptr+1)) class]
    ]
```

Remote evaluation

```
eval [user croak] primitive: 30
remoteCopy
    [↑Context new sender: sender receiver: receiver mclass: nil
      method: method tempframe: tempframe pc: pc+2 stackptr: stackptr]
```

Interpretation

```
have: receiver interpret: method
    ["Warning -- someone must be holding method's literals."
    "Warning -- do not call as self have:interpret: or cycle will result"
    tempframe ← (Vector new: method∘3).
    self restart. ↑self interpretFor: thisContext sender]
interpret: objectCode with: tframe
    ["interpret in new context with tempframe tframe"
    "Warning -- someone must be holding objectCode's literals."
    "Warning -- do not call as self interpret:with: or cycle will result"
    ↑(Context new
      sender: nil receiver: receiver mclass: nil
      method: objectCode tempframe: tframe
      pc: objectCode∘6 stackptr: objectCode∘5-1)
     interpretFor: thisContext sender]
interpretFor: sender  "resume execution"
    [PriorityInterrupt new run: self]
```

Deallocation

```
destroyAndReturn: value
    [thisContext sender ← sender.
    tempframe all ← nil.
    ↑value]
eraseFully
    [tempframe≡nil⇒ []
    tempframe all ← nil. tempframe ← nil]
release       "release frames to break cycles"
    [tempframe ← nil.
    sender≡nil⇒[] sender release]
releaseFully | c      "nullify frames to break cycles"
    [c ← self.
```

```
    until: c≡nil do: [c eraseFully. c ← c sender]]
SystemOrganization classify: ↱Context under: 'Kernel Classes'.
```

"Object"

```
Class new title: 'Object'
   subclassof: nil
   fields: ''
   declare: '';
   asFollows_!
```

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

Comparison

```
≤ x [↑self>x≡false]
≡ x [↑self≡x "In case this is reached by perform:"] primitive: 4
≠ x [↑self=x≡false]
≥ x [↑self<x≡false]
= x [↑self≡x]
and: x [self⊃[↑x eval] ↑false]
and: x [self⊃[↑x] ↑false]
equ: x [x⊃[↑self] ↑self≡false]
or: x [self⊃[↑true] ↑x eval]
or: x [self⊃[↑true] ↑x]
sameAs: object
   [↑self≡object]
xor: x [x⊃[↑self≡false] ↑self]
```

Classification

```
class [user croak] primitive: 27
is: x [↑self class≡x]
Is: x  "Is the class x a superclass or class of self"
   [self class ≡ x ⊃[↑true]
   ↑self class Isa: x]
isArray
   [↑false]
isnt: x [↑(self class≡x) ≡ false]
Isnt: x
   [↑(self Is: x)≡false]
isNumber
   [↑false]
```

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 is: Object⇒[↑self]
    ↑self class recopy: self]
```

Aspects

```
asOop [user croak] primitive: 46
asText [↑self asString]
asTextEntity [↑TextEntity new text: self asText style: DefaultStyle]
canunderstand: selector
    [↑self class canunderstand: selector]
fields
    ["Return an Array of all my field names or many of my subscripts."
    self class is: VariableLengthClass⇒
        [self length ≤ 50⇒ [↑1 to: self length]
        ↑ (1 to: 20) concat: (self length-20 to: self length)]
    ↑self class instvars]
hash [user croak] primitive: 46
inspect
    [user topWindow leave.
    user restartup: (InspectWindow new of: self)]
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]
itself
ref: index
    [↑FieldReference new object: self offset: index]
title
    [↑self class title + '.' + self asOop base8]
```

Printing

```
asFullString | strm
    [strm ← (String new: 20) asStream.
    self fullprinton: strm. ↑strm contents]
asString | strm
    [strm ← (String new: 16) asStream.
```

```
    self printon: strm. ↑strm contents]
filout | file
    [↑user displayoffwhile:
        [file ← dp0 file: self title asFileName.
        self fullprinton: file.
        file shorten; close]]
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]
```

Compiler Defaults

```
's code
    [↑Generator new evaluate: code asStream in: false to: self notifying: self]
argsOff: stack
    [self⇒ [stack pop: 1]]
asRemoteCode: generator
    [↑ParsedRemote new expr: self]
emitForEffect: code on: stack
emitForTruth: trueSkip falsity: falseSkip into: code on: stack
    [self emitForValue: code on: stack.
    (trueSkip jmpSize + falseSkip) emitBfp: code on: stack.
    trueSkip emitJmp: code on: stack]
emitForValue: code on: stack
emitsLoad
    [↑false]
emittedReceiver
    [↑false]
emittedVariable
    [↑false]
firstPush
    [↑¯1]
interactive
    [↑false]
isField
    [↑false]
notify: errorString at: position in: stream
    [↑self notify: errorString at: position in: stream for: self class]
notify: errorString at: position in: stream for: class | syntaxWindow
    [NotifyFlag⇒
        [syntaxWindow ← SyntaxWindow new of: errorString at: position in:
```

```
stream for: class from: thisContext sender.
      thisContext sender ← nil.
      user restartup: syntaxWindow]
    user notify: errorString. ↑false]
remote: generator
returns
  [↑false]
sizeForEffect: nextPush
  [↑0]
sizeForTruth: trueSkip falsity: falseSkip
      | jump
  [jump ← trueSkip jmpSize.
  ↑self sizeForValue + (jump+falseSkip) bfpSize + jump]
sizeForValue
  [↑0]
```

System Primitives

```
error | sender op n args i          "after compiling execute: nil installError. "
  [sender ← thisContext sender.
  op ← sender thisop.
  n ← op numArgs.
  args ← Vector new: n.
  for: i from: (n to: 1 by: -1) do:
    [args↓i ← sender pop].
  ↑self messageNotUnderstood: op withArgs: args from: sender]
installError | code old
  [code ← Object md method: &error.
  old ← SpecialOops↓1.
  old asOop≠(mem↓3)⇒ [user notify: 'Object installError failed']
  Top criticals:
    [mem↓3 ← code asOop.
    SpecialOops↓1 ← code]]
messageNotUnderstood: op withArgs: args from: sender
  [thisContext sender ← sender.
  user notify: 'Message not understood: '+op]
nail [user croak] primitive: 31          "Nail me in core and return my core
address"
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]
perform: selector withArgs: vec
  [selector mustTake: vec length.
  ↑self performDangerously: selector withArgs: vec]
```

```
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
performDangerously: selector withArgs: vec
   [vec length=0⇒ [↑self performDangerously: selector];
      =1⇒ [↑self performDangerously: selector with: vec◦1];
      =2⇒ [↑self performDangerously: selector with: vec◦1 with: vec◦2];
      =3⇒ [↑self performDangerously: selector with: vec◦1 with: vec◦2 with: vec◦3]
   user notify: 'More than 3 args for perform:']
PTR [] primitive: 46
refct [user croak] primitive: 45
startup        "loopless scheduling"
   [self firsttime⇒
      [while: self eachtime do: [].
      ↑self lasttime]
   ↑false]
swap: variable | x  "assign me to variable and return its old value"
   [x ← variable value. variable value ← self. ↑x]
unNail [user croak] primitive: 32          "Release me from being nailed"
↵
SystemOrganization classify: ↶Object under: 'Kernel Classes'.↵
```

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

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

Terminal

```
anybug [↑self buttons >0 or: (mem∘0177100) < 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
   ["stuff char into the event queue"
   [char is: String ⇒[char ← char∘1]].
   Events next ←
      UserEvent new
      x:          self x                          "event x"
      y:          self y                          "event y"
      type:        1                              "2=up, 1=down"
      stroke:     (kbMap find: char)                 "1-336"
      elapsed:     Events   elapsedtime              "1-32767 sixtieths of a sec"
      time:        Events time + Events elapsedtime.]
kbdnext | event [
   "returns next character (mapped) if any; otherwise false"
   while: [
      (event ← Events dequeue) or:
      (event ← Events primitiveDequeue)] do: [
      event isKbdDown⇒ [↑kbMap∘event stroke]].
   "self rawkbck⇒ [↑kbMap∘Events next stroke]"
   ↑false]
keyset
   ["Fix a bug in sysdefs."
   ↑037 - ((mem∘0177030 lshift: ¯3) land: 037)]
mp [
```

```
    ↑Point new
      x: mem○0424 + mxoffset
      y: mem○0425 + myoffset]
mpnext [
    "return next mouse point if any button is down; otherwise false"
    self anybug⊃ [↑self mp]
    ↑false]
nobug
    [↑self anybug ≡ false]
rawkbck | event              "flush event queue until down keyboard event or
queue empty."
        [while: (event ← Events peek) do:
          [event isKbdDown ⊃[↑event stroke] Events next].
        ↑false                    "if queue empty, return false"
        ]
rawkbd | stroke
        [until: (stroke ← self rawkbck) do: [].          "wait for activity"
        Events next.  ↑stroke                    "if key down, pop queue and
return stroke"
        ]
redbug [↑self buttons=4 or: (mem○0177100) < 0]
tablet [↑(mem○0177100) ≠ 0]
tabletabsolute [mem○0126 ← 1]
tabletbug [↑(mem○0177100) < 0]
tabletrelative [mem○0126 ← ¯1]
waitbug
    [until: self anybug do: [] ↑self mp]
waitclickbug
    [self waitnobug. ↑self waitbug]
waitnobug
    [while: self anybug do: [] ↑self mp]
waitnokey [until: self keyset=0 do: [self rawkbck]]
x [↑mem○0426 - htab        "cursorx, horiz tab adjusted"]
y [↑mem○0427 - vtab        "cursory, vert tab adjusted"]
yellowbug [↑self buttons=1]
```

Dialog Window

```
clear       "clear disp of debris and characters"
    [disp clear]
clearshow: str
    [disp clear; append: str; show]
cr [disp cr]
croak
    [self notify: 'A primitive has failed.']
ev
    [disp ev]
frame       "return rectangle of dialogue window"
    [↑disp text frame]
newdisploc: origin and: corner "for moving disp"
    ["user newdisploc: 4⊙4 and: 600⊙74.
          lets disp exactly fill top of screen and allows for
          five lines of cream.strike"
```

```
   (disp text frame inset: ¯2⊙¯2) clear.
   disp text frame ← origin rect: corner.
   disp outline. disp show]
print: x [disp print: x; show]
read [↑disp read]
request: s[↑disp request: s]
show [disp show]
show: str
   [disp append: str; show]
space
   [disp space]
tab [disp tab]
```

Display

```
currentCursor [↑currentCursor]
currentCursor: c | coff [
   currentCursor ← c.
   coff ← c offset.
   mxoffset ← coff x - htab.
   myoffset ← coff y - vtab]
cursorloc ← pt [
   mem◦0424 ← pt x - mxoffset.
   mem◦0425 ← pt y - myoffset]
displayoffwhile: expr | t v
   [t ← mem◦067. mem◦067 ← 58.
   v ← expr eval. mem◦067 ← t. ↑v]
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 ← 1 max: tab y/2.
   htab ← tab x|16.
   vtab ← mem◦063*2.
   screenrect ← 0⊙0 rect: (extent x|32)⊙(extent y|2).
   self currentCursor: currentCursor;
      reconfigure]
screenrect [↑screenrect]
```

Window Scheduling

```
promote: window
   [sched promote: window]
restart | i
      [[Events ≡ nil ⇒
         [Events ← EventQueue init. "Top init3. initialize Event queue and
Time interrupt"]].
      NormalCursor topage1.
      self restart: [user run]]
restart: code | u
   [u ← code cleancopy. u sender ← nil.
```

```
    thisContext sender releaseFully.
    thisContext sender ← nil.      code ← nil.      "release caller chain"
    MessageDict new freeMethods.          "release held code"
    disp frame flash.
    while: true do: [u eval]]
restartup: window
    [ "Equivalent to schedule new window, restart, and redbug in window,
except firsttime is already done."
    thisContext sender releaseFully. thisContext sender ← nil.
    NormalCursor topage1.
    self schedule: window.
    thisContext tempframe all ← nil.
    self run: true]
restore | w
    [screenrect clear. disp outline.
    for: w from: (sched length to: 1 by: ¯1) do:
        [(sched∘w) show]]
run
    [self run: false]
run: topFlag | i w forward "topFlag means sched∘1 already is awake"
    [forward ← [topFlag⇒ [w←sched∘1. while: w eachtime do: []. w lasttime]
true].
    while: true do:
        [i←0.
        until: [(i←i+1)>sched length⇒[]
            w← [forward⇒[sched∘i] sched∘(sched length+1-i)].
            w firsttime] do: []
        i>sched length⇒[]
        sched promote: w.
        while: w eachtime do: []
        forward← w lasttime]]
sched [∩sched]
schedule: window
    [sched≡nil⇒[sched ← window asVector]
    sched ← window asVector concat: sched]
scheduleOnBottom: window
    [sched≡nil⇒[sched ← window asVector]
    sched ← sched concat: window asVector]
topWindow [∩sched∘1]
unschedule: window | t
    [0<(t← sched find: window)⇒
        [sched ← sched∘(1 to: t-1) concat: sched∘(t+1 to: sched length)]]
```

Special windows

```
browse      "Let the user draw a five-paned window to browse through
classes."
    [user topWindow leave.
    self restartup: BrowseWindow default]
notifier: titleString level: lev interrupt: flag
    ["Restore the full display.  Schedule a one-paned window to notify the user
that errorString happened."
    self restoredisplay.
```

```
    NotifyFlag≡false⇒
        [disp cr;
            append: 'NotifyFlag is false...'; cr;
            append: ' top-blank shows stack, user restart_! aborts,'; cr;
            append: ' tempframe_! shows args, ctrl-d proceeds'; cr;
            append: titleString; cr; show.
        (Top•lev) debug. ↑false]
    ↑NotifyWindow new of: titleString level: lev interrupt: flag]
notifier: titleString stack: stack interrupt: flag
    ["Restore the full display.  Schedule a one-paned window to notify the user
that errorString happened."
    self restoredisplay.
    NotifyFlag≡false⇒
        [disp cr;
            append: 'NotifyFlag is false...'; cr;
            append: ' sender debug_! shows stack, user restart_! aborts,'; cr;
            append: ' tempframe_! shows args, ctrl-d proceeds'; cr;
            append: titleString; cr; show.
        stack debug. ↑false]
    ↑NotifyWindow new of: titleString stack: stack interrupt: flag]
notify: errorString | notifyWindow
    ["Create a notify window looking at the Context stack"
    notifyWindow ← self notifier: errorString stack: thisContext sender
interrupt: false.
    notifyWindow⇒
        [thisContext sender ← nil.
         Top currentPriority=1⇒
            [self restartup: notifyWindow]
         self scheduleOnBottom: notifyWindow.
         Top errorReset]
    ↑nil]
```

Time

```
convertTime: s returnSecs: format | d dd t dfirst dlast m570 m571 [
    "s is total seconds from midnight Jan 1 1901 GMT (Greenwich mean time).
    see maxc <AltoDocs>AltoTime.Press for details"

    "time zone specific parameters"
    m570 ← mem•0570. m571 ← mem•0571.

    "adjust for time zone"
    s ← s + (([m570 ≥ 0⇒ ["west" ¯1] "east" 1]) * (
        (3600 * ("hours" m570 bits: (1 to: 4))) +
        (60 * ("additonal minutes" m571 bits: (1 to: 6))))).

    "current day (in local standard time)"
    d ← Date new fromSeconds: s.
    [format⇒ [] t ← Time new fromSeconds: s].

    "check for DST. correct DST parameters for nonleap years and
    round to previous Sunday if necessary"
```

```
   "day of the year on or before which DST takes effect"
   dfirst ← m570 land: 0777 "bits: (7 to: 15)".

   [[dfirst = 366⇒ ["DST not in effect" false]
   (dd ← d day) ≥ (dfirst ← dfirst + d leap - 1)⇒ [

      "day of the year on or before which DST ends"
      dlast ← (m571 land: 0777 "bits: (7 to: 15)") + d leap - 1.
      dd < dlast "if false, definitely after" and:
      dd < ((Date new day: dlast year: d year) previous: 6) day]
   "possibly earlier than or at beginning of range"
   dd ≥ ((Date new day: dfirst year: d year) previous: 6) day]⇒ [

      "daylight savings time in effect. add an hour"
      format⇒ [s ← s + 3600]
      t hours = 23⇒ [
         d ← d+1.
         t hours: 0]
      t hours: t hours+1 ]].

   "return either total seconds or Date and Time"
   ↑[format⇒ [s] d,t]]
now [
   "return a Vector of current Date and Time"
   ↑self convertTime: self rawtotalsecs returnSecs: false]
rawtotalsecs [
   "seconds (in GMT) since Jan 1 1901: as a largeInteger.
   rewrite with next release"
   ↑user timewords viewer nextNumber: 2]
ticks "Return the 38.08-millisecond interval timer"
   [↑mem∘0430]
time [↑self now∘2]
times: expr | t
   [t ← self ticks. expr eval. ↑self ticks-t]
timewords | s [
   "seconds (in GMT) since Jan 1 1901: as a String"
   Stream new of: (s ← String new: 4);
      nextword ← mem∘0572;
      nextword ← mem∘0573.
   ↑s]
today [↑self now∘1]
totalsecs [
   "convert from GMT to local and correct for Daylight Savings"
   ↑self convertTime: self rawtotalsecs returnSecs: true]
```

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/resume

```
backup  "back up smalltalk on ifs and resume"
   [self quitThen: 'ftp ifs delete/c small.sv store/c small.sv; resume small.sv
']
InLd: fileid
   [user notify: 'file problem'] primitive: 86
OutLd [] primitive: 85
overlay: fileid [
   [E≡nil⇒ [] E etherKill].
   dp0 close. dp1 close.
   self OutLd⇒[self InLd: fileid]
   [mem◦0147≠0⇒[self restore]].          "If this is an XM"
   while: user keyset>0 do: [user show: 'The keyset is stuck'; cr]]
quit
   [self overlay: 0,0,0,0,0]
quitThen: str | rem rest   "quit, then have OS execute str"
   [rem ← dp0 file: 'rem.cm'.
   rest ← rem contents.
   rem reset; append: str; cr; append: rest; shorten; close.
   self quit]
quitThen: s continue: r [
   [s⇒ [
      "something for O.S. to do"
      (dp0 oldFile: 'rem.cm.') settoend;
         append: [s is: File⇒ ['@' + s sclose name + '@'] s]; append: '; ';
         append: [r⇒ ['Resume.~ small.boot'] 'Quit.~; Resume.~ small.boot'];
         cr flush]].
   user overlay: ↱(0 0 0 0 0) "quit"]
Swat [] primitive: 90
```

Misc System Stuff

```
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
    ]
growSmalltalk: numberofdiskpages
    "for preemptive growth of Small.boot on disk"
    [(dp0 file: 'small.boot') extendby: numberofdiskpages]
initCompiler "Initialize shared variables of parser and generators"
        | code sel c t
    [
    Smalltalk declare: ↗(TokenCodes ByteCodes).
    [TokenCodes≡nil⇒ [TokenCodes←SymbolTable new init: 32]].
    [ByteCodes≡nil⇒ [ByteCodes←SymbolTable new init: 32. Integer sharing:
ByteCodes]].
    TokenCodes
        declare:
            ↗(aRightBrack aPeriod "First 2 in this order"
            aLeftPar aSemicolon aCondArrow aHand aReturnArrow
            aLeftBrack aRightPar aLeftArrow
            aBinary "All above must be less, all below must be greater"
            aNumber aString "All below must be in that order"
            aKeyword aGibberish aColon aDigit aWord)
        as:
            ↗(1 2
            3 4 5 6 7
            8 9 10
            20
            30 31
            41 42 43 44 45
            ).
    ByteCodes
        declare:
            ↗(toLoadField toLoadTemp toLoadLit toLoadLitInd
            toLoadCtxt toLoadTempframe
            toLoadConst toLoad0 toLoad1 toLoadSelf toLoadNil toLoadFalse toLoadTrue
            toSmashPop toSmash toPop toReturn toEnd toLoadThisCtxt toSuper
            toShortJmp toShortBfp toLongJmp toLongBfp
            toPlus toMinus toGtr toGeq toNext toEq toNew toAsStream
            toSendLit)
        as:
            ↗(0 16 32 64
                112 116
                120 121 122 113 125 126 127
                128 129 130 131 132 133 134
                144 152 160 168
                176 177 179 181 194 197 203 207
                208).
    c ← Dictionary new init: 16.
    c insertall: ↗('self' 'thisContext' 'super' 'nil' 'false' 'true')
        with: ↗(113 133 134 125 126 127).
    ByteCodes declare: ↗stdPrimaries as: c.
    c ← Dictionary new init: 64.
    code ← 175.
    for: sel from: ↗(+ - < > ≤ ≥ = ≠ * / \ | min: max: land: lor: ∘ '∘←' next
```

```
'next←' length ≡ UNUSED UNUSED class and: or: new new: to: oneToMeAsStream
asStream) do:
      [ "Atoms not wanted here -- only strings and characters"
      code ← code+1.
      sel≡↗UNUSED⇒ []
      sel ← [sel length=1⇒ [sel↓1] sel is: String⇒ [sel] sel asString].
      c insert: sel with: code.
      ].
   ByteCodes declare: ↗stdSelectors as: c.
   c ← Dictionary new init: 8.
   c insertall: ↗('while:do:' 'until:do:' 'for:to:do:' 'for:from:do:'
'for:from:to:by:do:' 'for:from:to:do:' 'if:then:else:' 'if:then:')
      with: ↗(whiledo:args: untildo:args: fortodo:args: forfromdo:args:
forfromtobydo:args: forfromtodo:args: ifthenelse:args: ifthen:args:).
   ByteCodes declare: ↗inLineMsgs as: c.
   for: t from: ↗((toLoadFieldLong 0210) (toLoadTempLong 0211) (toLoadLitLong
0212)
         (toLoadLitIndLong 0213) (toSendLitLong 0214)
         (codeLoadField 0200) (codeLoadTemp 0400) (codeLoadLit 0600)
         (codeLoadLitInd 01000) (codeSendLit 01200)) do:
      [ByteCodes declare: t↓1 as: t↓2]]
oopsToFile | a c i t s cs f ts
   [f ← dp0 file: 'oops'.
   cs ← user classNames transform: a to: Smalltalk↓a.
   t ← cs transform: a to: a asOop.
   ts ← t permutationToSort.
   for: i from: ts do:
      [f append: (t↓i) base8; tab; append: (cs↓i) title; cr].
   for: c from: cs↓ts do:
      [f cr; append: c title; cr.
      s ← c md contents.      "selectors"
      t ← s transform: a to: (c md method: a) asOop.  "method oops"
      for: i from: t permutationToSort do:
         [f tab; append: (t↓i) base8; tab; append: s↓i; cr]
      user show: c title; cr]
   f close]
printCrossreference | dict m md frame l f each s class
      " user displayoffwhile: user printCrossreference "
   [dict ← Dictionary init.
   for: m to: 32 do:
      [dict insert: SpecialOops↓(9+m) with: ↗((Primitives) ()) copy].
   AllClassNames transform: each to:
      [user show: each; space.
      md← (Smalltalk↓each) md.
      for: m from: md do:          "Tally all the UniqueString literals"
         [   [s← dict lookup: m⇒[] dict insert: m with: (s← ↗(()()) copy)].
         s↓1 has: each⇒[] s↓1← s↓1, each.
         for: l from: (md literals: m) do:
            [l is: UniqueString⇒
               [   [s← dict lookup: l⇒[] dict insert: l with: (s← ↗(()()) copy)].
               s↓2 has: each⇒[] s↓2← s↓2, (each,m)]]]].
   f← PressPrinter init of: (dp0 file: 'CrossReference.press').
```

```
    f stamp.
    frame← f defaultframe.          "Print the messages out sorted"
    for: m from: dict contents sort do:
      [f frame← frame.
      md← dict.m.
      s← Stream default.
      s append: m; append: [(md.1) length=0⇒[' ( - undefined - ']' ('].
      for: l from: (md.1) sort do: [s append: l; append: ', '].
      s skip: ¯2; append: ')'.
      f print: (s contents asParagraph maskrun: 1 to: m length under: 1 to: 1).
      f frame← (frame minX+500)@frame minY rect: frame corner.
      s reset.
        [md.1 has: ↗Primitives⇒[s append: 'untallied.'. md.2← ↗()]
        (md.2) length=0⇒[s append: '- unreferenced -']].
      class← ↗-.
      for: l from: (md.2) sort do:
        [  [l.1=class⇒[s append: ', ']
          [class≠↗-⇒[s cr]]. s append: '('; append: l.1; append: ') '.
          class← l.1].
        s append: l.2].
      f print: s contents asParagraph].
    f close]
pstats | zadd pmbase ["user pstats"
    user displayoffwhile: [
    zadd ← Vmem specialLocs.12.
    pmbase ← Vmem specialLocs.2.
    user cr print: dp0 free; show: ' disk pages left'; cr;
      print: mem.(zadd-2); show: ' Ooze pages used, ';
      print: mem.(zadd-1) - (mem.(zadd-2)); show: ' before grow'; cr;
      print: (mem.(pmbase-2) - (mem.(pmbase-1))) unsigned;
      show: ' objects left, '; print: Class howMany; show: ' classes used'; cr]]
purgealittle [] primitive: 89
release | t m ms class
      "prepare to release this version (after editing version)"
    [(t← Undeclared contents) length>0⇒
      [user notify: 'Undeclared contains '+ t asString]
    ChangedMessages← Stream default. class← ''.
    for: m from: Changes contents sort do:
      [ms← m asStream.
      (ms upto: 040)=class⇒
        [ChangedMessages append: ', '; append: (ms upto: 040)]
      ChangedMessages cr; append: m.
      class← m asStream upto: 040].
    ChangedMessages← ChangedMessages contents.
    t← dp0 file: 'ChangedMessages'.
    t settoend; cr; cr.
    t asParagraphPrinter stamp.
    t append: ChangedMessages; close.
    Changes init.
    user clearshow: 'Welcome to '+user version]
systemStartup "To do after system flush and installation of new core image"
    [Top top.
```

```
    Window classInit.

    "The following screen extent seems to really fill the screen in x,
    the Alto Hardware Manual to the contrary notwithstanding."
    self screenextent: 640@580 tab: 0@50.

    dp0 restore.
    dp1 Close.
    (VirtualMemory new) thisvmem.
    Vmem afterBirth.
    user restore]
version [^'Smalltalk 5.3i']      " user release
5.3i- (July 5, The LRG Summer of 78 Release)
        The system has been compacted.
        All subclasses of Number respond uniformly to negated.
        Some small fixes to random files, PressPrinter, EtherSocket
          and PanedWindow hardcopy
        The Ethernet kernel and EFTP transmitter are here at last!
          See Steve Weyer for info on printing Press files directly.
          See John Shoch for basic info on Ethernet.
        News about pools (variables shared among classes):
          Ethernet shared variables (except E) are now in EtherPool.
          To learn how to maintain pools safely see examples in:
            UserView initCompiler, DefineVariables classInit.
          Variables may be safely moved from Smalltalk to a SymbolTable pool by:
            pool declare: @(the variable names) from: Smalltalk
        The reclaim facility and class Citation have been excised.
        Steve Chernicoff fixed a compiler bug, so now thisContext can be
          the implied receiver of a keyword message, as promised in the manual.
          For example, try:
            | x y [for: x from: @(2 3) with: y from: @(4 5) do: [(x,y) print]]
          which invokes the new method Context for:from:with:from:do:
        FieldReference value←x and ObjectReference value←x now return x;
            ObjectReference object: x has been added; it returns self
        No one seemed to be using the global variable Counter, so it is gone.
        Classes Form, Path, & Image defined but don't use them yet.
        Ancient safing bug fixed.
        Still garbage left over from NotifyWindows and BrowseWindows.
5.3h- (June 6, Steve W, Diana M)
        Many small to medium fixes.
        Paragraph editor now a subclass of TextSelection.
        Some new classes to support development of new
        text/graphics package.
5.3g- (May 23, Steve W, Dan I)
        Substantial changes to press file and galley editing.
        File fixes and exponential and log functions added to Float.
        New class FieldNameCollector (used by Class instvars)
          collects strings without uniquing.
    "
                                                    workspace
    [user notify: 'Not meant to be executed']
                                                              "
```

XEROX - Learning Research Group

```
user restore
user quit
Changes init
user restart

(dpo file: 'changes.st') filout.
(dpo file: 'FiledoutFile.st') filin.
(dpo file: 'old.st') newChars; filin.
(dpo file: 'TextFile') edit
NotifyFlag ← true.
dpo list: '*.st'
Directory compressAll.
File noChanges
user screenextent: 640@580 tab: 0@50.
BitRect new fromuser; edit
user schedule: (defaultBitRectEditor newframe)

| each. AllClassNames transform: each to: (Smalltalk@each) compressAll.

"

SystemOrganization classify: #UserView under: 'Kernel Classes'.
```

"VariableLengthClass"

```
Class new title: 'VariableLengthClass'
   subclassof: Class
   fields: ''
   declare: '';
   veryspecial: 20;
   asFollows_!
```

I am a class whose instances have numbered elements instead of named fields.

Initialization

```
classInit        "gets propagated to a dummy instance"
   [(self new: 1) classInit]
```

Instance access

```
allInstances [user notify: 'use allInstances: instead to specify the length range'
   "the length ranges are 0,1,2,3,4,5,6,7,8 individually and groups 9 (to 16), 17 (to
32), 33 (to 64), 65, 129, 257, 513, 1025, 2049, and 4197"]
allInstances: len | indx vec PCLs i "returns a vector containing all instances of
this class and length mixed with nils"
   ["for large lengths, instances come in groups with lengths within a single
power of 2"
   PCLs ← Vmem pclassesOf: self length: len.  "vector of PCLs"
   vec ← Vector new: 128*PCLs length.
   for: i to: PCLs length do:
      [(vec:[i-1*128+1 to: i*128]) all← PCLs:i].
   thisContext destroyAndReturn:
      (self fromFreelist: (Vmem freelistOffset: len) fill: vec)]
copy: inst | t i
   [t ← self new: inst length.
   for: i to: inst length do:
      [t:i ← inst:i]
   ↑t]
howMany: len | v "how many instances of this class and length are in use
now?"
   [v ← self allInstances: len.
   thisContext destroyAndReturn: v length - (v count: nil)]
new
   [user notify: 'use new: <Integer=length> here.']
new: length
   [length>16384 ⇒[user notify: length asString+
                                    ' is too big a String']
   length>8192 ⇒[user notify: length asString+
                                    ' is too big a Vector']
   length<0 ⇒[user notify: length asString+
                                    ' -- negative length is invalid']
   ↑self new: length asInteger] primitive: 29
recopy: inst | t i
   [t ← self new: inst length.
```

```
    for: i to: inst length do:
        [to i ← (inst to i) recopy]
    ↑t]
↵
SystemOrganization classify: ↗VariableLengthClass under: 'Kernel Classes'.↵
```

```
Class new title: 'Date'
   subclassof: Number
   fields: 'day year'
   declare: 'monthnames secsinday ';
   asFollows_!
```

Implements dates. (Steve Weyer)

Initialization

```
classInit [
   monthnames ← ↗(
      January February March April May June
      July August September October November December).
   secsinday ← 24*60*60]
```

Setting state

```
day: day month: month year: year [
   [year < 100⇒ [year ← 1900 + year]].
   [month ← self whichmonth: month⇒ []
   user notify: 'illegal month'].
   [day < 1 or: day > (self daysinmonth: month)⇒ [
      user notify: 'illegal day in month']].
   day ← day + (self monthday: month)]
day: day year: year | d
   [while: day > (d ← self daysinyear) do: [
      year ← year + 1.
      day ← day - d].
   while: day ≤ 0 do: [
      year ← year - 1.
      day ← day + self daysinyear].
   ]
default "Make self be current date"
   [self fromSeconds: user totalsecs]
fromSeconds: sec | d [
   d ← (sec / secsinday) asSmall. "Ignore time part.
   There are 1461 days in a 4-year cycle.
   2000 is a leap year, so no extra correction is necessary.
   day:year: will fix things up"
   self day: 1+ (d \ 1461) year: 1901+ (d/1461 *4)]
```

Aspects

```
asSeconds "Seconds since the beginning of time (local time)" [
   ∩secsinday * (self - (Date new day: 1 year: 1901))]
day [∩day]
dayinmonth [∩day - (self monthday: self month)]
dayinyear [∩day]
daysinmonth [∩self daysinmonth: self month]
daysinmonth: m [
   ∩↗(31 28 31 30 31 30 31 31 30 31 30 31)∘m + [m=2⇒ [self leap] 0]]
```

```
daysinyear [↑365 + self leap]
daysleft [↑self daysinyear - day]
leap [
   year \ 4 = 0⇒ [
      year \ 100 = 0⇒ [year \ 400 = 0⇒ [↑1] ↑0]
      ↑1]
   ↑0]
month | m leap [
   leap ← self leap.
   for: m from: 12 to: 1 by: ¯1 do: [
      (↱(0 31 59 90 120 151 181 212 243 273 304 334)∘m +
         [m > 2⇒ [leap] 0] "self monthday: m") < day⇒ [↑m]].
   user notify: 'illegal month']
monthday: m "Return first day-in-year of m'th month"
   [↑↱(0 31 59 90 120 151 181 212 243 273 304 334)∘m +
      [m > 2⇒ [self leap] 0]]
monthname [↑monthnames∘self month]
weekday [
   ↑↱(Tuesday Wednesday Thursday Friday Saturday Sunday Monday)
      ∘self weekdayIndex]
weekdayIndex | a d [
   [day ≤ (self monthday: 3)⇒ [
      a ← year-1.
      d ← 306]
   a ← year.
   d ← ¯59 - self leap].
   "Tuesday=1,..., Monday=7"
   ↑d + day + a + (a/4) + (a/400) - (a/100) \ 7 + 1]
whichmonth: m | a "M may be a (partial) month name, or a number. Return
the month number, or false" [
   m is: String⇒ [
      m ← m + '*'.
      for: a to: 12 do: [
         "first partial match"
         m match: monthnames∘a⇒ [↑a]].
      ↑false]
    ↑m between: 1 and: 12]
year [↑year]
```

Arithmetic

```
+ days
   [↑Date new day: day+days year: year]
- date
   [date is: Date⇒ [
      ↑(year-1 / 4) - (date year / 4) +
      day + date daysleft + (year-1 - date year * 365)]
   ↑0 - (date - self)]
previous: di [
   "e.g. previous: 6 (Sunday) returns Date which is previous closest Sunday.
   note: di=self weekdayIndex returns self+0"
   ↑self + (0 - (7 + self weekdayIndex - di \ 7))]
```

Printing and reading

```
from: s [self readfrom: s asVector "asSet" viewer format: nil]
printon: strm [self printon: strm format: ↗(1 2 3 040 3 1)]
printon: strm format: f | i m [
   "f is print format.
   1-3   positions to print day,month,year respectively
   4   character separator
   5   month format (1 month #, 2 first 3 chars, 3 entire name)
   6   year format (1 year #, 2 year #\100)"

   m ← self month.
   for: i to: 3 do: [
      [f∘i
      =1⇒ [day - (self monthday: m) printon: strm];
      =2⇒ [
         f∘5
         =1⇒ [m printon: strm];
         =2⇒ [strm append: monthnames∘m∘(1 to: 3)]
         strm append: monthnames∘m]
      ([f∘6=1⇒ [year] year\100]) printon: strm].
      i<3⇒ [strm next ← f∘4 "separator"]]]
readfrom: strm
   [self readfrom: strm format: ↗(1 2 3)]
readfrom: strm format: order | dmy i [
   strm ◁↗today⇒ [self default];
      ◁↗+⇒ ["pseudo positive infinite date" self day: 1 year: 1999];
      ◁↗-⇒ ["pseudo negative infinite date" self day: 1 year: 1901]
   [order ≡ nil⇒ [order ← ↗(1 2 3)]].
   dmy ← Vector new: 3.
   for: i to: 3 "dmy length" do: [dmy∘(order∘i) ← strm next].
   self day: dmy∘1 month: dmy∘2 year: dmy∘3]
┘
SystemOrganization classify: ↗Date under: 'Numbers'.┘
Date classInit┘
```

"Float"

```
Class new title: 'Float'
   subclassof: Number
   fields: ''
   declare: 'halfpi sqrt2 twopi fourthpi degreesPerRadian pi
radiansPerDegree ln2 ';
   bytesize: 16;
   veryspecial: 3;
   asFollows_!
```

These floating-point numbers are good for about 8 or 9 digits of accuracy, and the range is between plus and minus 10↑4000. Here are some valid floating-point examples:

8.0 13.3 0.3 2.5e6 1.27e¯300 ¯12.987654e2412

Mainly: 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
   [arg isNumber⇒ [↑self = arg asFloat] ↑false] primitive: 72
> arg
   [↑self>arg asFloat] primitive: 74
near: n [↑self near: n within: 1.0e¯4]
near: n within: eps [
   "for testing near equality, e.g. error convergence"
   ↑(self - n) abs ≤ eps]
negated [↑0.0-self]
sameAs: arg  "arg assumed to be of same class as self"
   [↑self=arg]
\ arg    "By analogy with integers"
   [self<0.0⇒[↑(self/arg) ipart+1.0*arg+self]
   ↑self-((self/arg) ipart*arg)]
| arg    "By analogy with integers"
```

```
   [↑(self/arg) ipart*arg]

Conversion
asDegrees       "self assumed to be in radians"
   [↑self / radiansPerDegree]
asDirection [↑self cos ⊙ self sin]
asFloat
asInteger       "Return an Integer = self ipart"
   [↑self asLarge] primitive: 78
asLarge | me digits "convert to LargeInteger"
   [self<0⇒[↑(0.0-self) asLarge negated]
    digits ← Stream default.
    [self=0.0⇒[digits next← 0]
     me ← self ipart.
     while꞉ me≥1 do꞉
      [digits next ← (me\0200) asInteger.
        me ← me/0200]].
    ↑LargeInteger new bytes: digits contents neg: false]
asRadians       "self assumed to be in degrees"
   [↑self * radiansPerDegree]
copy [↑self]
fpart [user croak] primitive: 77
ipart       "Returns a Float with zero fractional part"
   [↑self-self fpart]
recopy [↑self]
round
   [↑(self + [self < 0⇒ [¯0.5] 0.5]) asInteger]

Math functions
cos "for angles in radians"
   [self<0.0⇒[↑(self+halfpi) sin]
   ↑(halfpi-self) sin]
exp | a n1 x x2 P Q [
   "see Computer Approximations, pp. 96-104, p. 205 (EXPB 1065)"

   self abs > 9212.0 "1.0e4001 ln"⇒ [user notify: 'exp overflow']
   x ← self / ln2.
   (n1 ← Float new "2.0 ipow: x asInteger")
      instfield: 1 ← x asInteger * 2.
   [(x ← x fpart) ≥ 0.5⇒ [
      n1 ← n1 * sqrt2.
      x ← x - 0.5]].
   x2 ← x*x.
   "compute 2.0 power: x"
   P ← Q ← 0.0.
   "↷(0.252504285255762419337 44e4 0.28875563776168927289e2) reverse copy"
   for꞉ a from: ↷(28.875564 2525.0429) do꞉ [
      P ← (P*x2) + a].
   "↷(0.72857336028361108885189e4 0.375021654220866600213e3 0.1e1) reverse
copy"
   for꞉ a from: ↷(1.0 375.02165 7285.7336) do꞉ [
```

```
      Q ← (Q*x2) + a].
   ^n1 * ((Q + (x*P))/(Q - (x*P)))]
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)]
ln | a x x2 n P [
   "see Computer Approximations, pp. 105-111, p. 227 (LOGE 2663)"

   self ≤ 0.0⇒ [user notify: 'ln not valid for ' + self asString]

   x ← self + 0.0.
   "exponent"
   n ← ln2 * (((x instfield: 1) / 2) asFloat - 0.5).
   "mantissa between 0.5 and 1.0".
   x instfield: 1 ← 0.
   x ← x * sqrt2.
   x ← (x - 1.0) / (x + 1.0).
   x2 ← x*x.
   P ← 0.0.
   "↗(0.200000000046727e1 0.666666635059382 0.4000059794795
      0.28525381498 0.2376245609) reverse copy"
   for: a from: ↗(0.23762456 0.28525381 0.40000598 0.66666664 2.0) do: [
      P ← (P*x2) + a].
   ^n + (x * P)]
log: base [^self ln / base asFloat ln]
neg "Obsolete - use negated, which is uniform for all Numbers"
   [^self negated]
sin | x x2 sum const  "for angles in radians"
   [self<0.0⇒[^self negated sin negated];          " normalize to 0≤self≤(pi/4) "
      >twopi⇒[^(self \twopi) sin];
      >pi⇒[^(self-pi) sin negated];
      >halfpi⇒[^(pi-self) sin]
   sum ← x ← self.
   x2 ← x*x.
   for: const from:        "Now compute the series"
      ↗(¯0.1666666664 0.0083333315 ¯1.98409e¯4 2.7526e¯6 ¯2.39e¯8)
      do: [sum ← const*(x ← x*x2)+sum]
   ^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]
tan | x x2 sum const  "for angles in radians"
   [self<0.0⇒[^self negated tan negated];          " normalize to 0≤self≤(pi/4) "
      >pi⇒[^(self \pi) tan];
      >halfpi⇒[^(self-halfpi) tan negated];
```

```
    >fourthpi⇒[↑1.0/(fourthpi-self) tan]
  sum ← x ← self.
  x2 ← x*x.
  for: const from:        "Now compute the series"
    ↗(0.3333314036 0.1333923995 0.0533740603 0.0245650893 0.0029005250
0.0095168091)
    do: [sum ← const*(x ← x*x2)+sum]
  ↑sum]
```

Printing

```
absprinton: strm digits: digits   "print me using digits significant figures"
    | fuzz x exp q i
  ["x is myself normalized to [1.0, 10.0), exp is my exponent"
  exp ← [self<1.0⇒ [0-(10.0/self epart: 10.0)] self epart: 10.0].
  x ← self/(10.0 ipow: exp).
  "round the last digit to be printed"
  fuzz ← 10.0 ipow: 1-digits.  x ← 0.5*fuzz+x.
  "check if rounding has unnormalized x"
  [x≥10.0⇒[x←x/10.0.  exp←exp+1]].
  [exp<6 and: exp>¯4⇒
    [q ← 0.          "decimal notation"
    exp<0⇒ [strm append: '0.0000'∘(1 to: 1-exp)]].
  q ← exp. exp ← 0].        "scientific notation"
  "use fuzz to track significance"
  while: x≥fuzz do:
    [i ← x asInteger.  strm next ← 060+i.
    x ← x-i * 10.0.  fuzz ← fuzz*10.0.
    exp ← exp-1.  exp=¯1⇒ [strm append: '.']].
  "append additional zeros if necessary"
  while: exp≥¯1 do:
    [strm next ← 060.
    exp ← exp-1.  exp=¯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 printon: strm digits: 8]
printon: strm digits: digits   "print me using digits significant figures"
  [self>0.0⇒[self absprinton: strm digits: digits]
  self=0.0⇒[strm append: '0.0']
  strm append: '¯'. (0.0-self) absprinton: strm digits: digits]
roundTo: d
  [↑(self/d+[self<0.0⇒[¯0.5]0.5]) ipart*d]
```

Initialization

```
classInit [
  "constants from Computer Approximations, pp. 182-183"
  "3.14159265358979323846264338327950288"
```

```
    pi ← 3.1415927.

    "pi/2 = 1.57079632679489661923132169163975144"
    halfpi ← 1.5707963.

    "pi/4 = 0.78539816339744830961566084581987572"
    fourthpi ← 0.78539816.

    "pi*2 = 6.28318530717958647692528676655900576"
    twopi ← 6.2831853.

    "pi/180 = 0.01745329251994329576923690768488612"
    radiansPerDegree ← 0.017453293.
    degreesPerRadian ← 180.0/pi.

    "2.0 ln = 0.69314718055994530941723212145817657"
    ln2 ←0.69314718.

    "2.0 sqrt = 1.41421356237309504880168872420969808"
    sqrt2 ← 1.4142136]
↵
SystemOrganization classify: #Float under: 'Numbers'.↵
Float classInit↵
```

"Integer"

```
Class new title: 'Integer'
   subclassof: Number
   fields: ''
   declare: 'digitbuffer ';
   bytesize: 16;
   veryspecial: 1;
   sharing: BitMasks;
   sharing: ByteCodes;
   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 | t
   [t ← arg asInteger.
    t isLarge⇒[↑true]
    ↑self ≤ t]
≠ arg | t
   [arg isNumber⇒
    [t ← arg asInteger.
     t isLarge⇒[↑true]
     ↑self ≠ t]
    ↑true]
≥ arg | t
   [t ← arg asInteger.
    t isLarge⇒[↑false]
    ↑self ≥ t]
* arg | t
   [arg is: Integer⇒[↑self asLarge*arg]
    t ← arg asInteger.
    t isLarge⇒[↑self asLarge*arg]
    ↑self*t] primitive: 21
+ arg | t
   [arg is: Integer⇒[↑self asLarge+arg]
    t ← arg asInteger.
    t isLarge⇒[↑self asLarge+arg]
    ↑self + t]
- arg | t
   [arg is: Integer⇒[↑self asLarge-arg]
    t ← arg asInteger.
    t isLarge⇒[↑self asLarge-arg]
    ↑self - t]
/ arg
   [0=arg⇒[user notify: 'Attempt to divide by 0']
    arg isLarge⇒[↑self asLarge/arg]
    ↑self / arg asInteger] primitive: 22
< arg | t
```

```
   [t ← arg asInteger.
    t isLarge⇒[↑true]
    ↑self < t]
= arg | t
   [arg isNumber⇒
    [t ← arg asInteger.
     t isLarge⇒[↑false]
     ↑self = t]
    ↑false]
> arg | t
   [t ← arg asInteger.
    t isLarge⇒[↑false]
    ↑self > t]
negated [↑0-self]
sameAs: arg  "arg assumed to be of same class as self"
   [↑self=arg]
\ arg    "mod"
   [0=arg⇒[user notify: 'Attempt to divide by 0']
    arg isLarge⇒[↑self asLarge/arg]
    ↑self \ arg asInteger] primitive: 26
| arg    "truncate"
   [↑self/arg*arg]
```

Bit Manipulation

```
allmask: b [↑b = (self land: b)]
anymask: b [↑0 ≠ (self land: b)]
bits: int [
   "int is an Interval:  0 is leftmost bit, 15 is rightmost"
   ↑self field: (int length "width" * 16) +
      15 - int stop "displacement from right"]
even [↑(self land: 1) = 0]
field: fld | t
   [t ← fld asInteger.
    t isLarge⇒[user notify: 'Field descriptor too large']
    ↑self field: t] primitive: 36
field: fld ← val | t
   [t ← fld asInteger.
    t isLarge⇒[user notify: 'Field descriptor too large']
    ↑self field: t ← val asSmall] primitive: 37
hash "used to find large integers in dictionaries"
   [↑self]
hibit | i
   [for: i to: 16 do:
      [(self land: (biton∘(17-i)))≠0⇒[↑17-i]]
   ↑0]
land: arg
   [↑self land: arg asSmall] primitive: 23
lor: arg
   [↑self lor: arg asSmall] primitive: 24
lshift: arg
   [↑self lshift: arg asSmall] primitive: 25
lxor: arg
```

```
    [↑self lxor: arg asSmall] primitive: 35
nomask: b [↑0 = (self land: b)]
```

Conversion

```
asFloat [user croak] primitive: 34
asInteger [↑self]
asLarge | me digits "convert to LargeInteger"
   [self<0⇒
      [self=¯32768⇒[↑'¯32768' asLarge]
       ↑(0-self) asLarge negated]
    digits ← Stream default.
    [self=0⇒[digits next← 0]
     me ← self.
     while: me>0 do:
      [digits next ← me land: 0177.
       me ← me lshift: ¯7]].
    ↑LargeInteger new bytes: digits contents neg: false]
asObject [user croak] primitive: 81
asSmall
inString | t
   [t ← String new: 1. to: 1 ← self. ↑t]
oneToMeAsStream          "used by for-loops"
   [↑Stream new of: (Interval new from: 1 to: self by: 1)]
unsigned [
   self < 0⇒ [↑65536.0 + self asFloat]
   ↑self asFloat]
```

Subscripts

```
cansubscript: a
   [↑self≥1 and: self≤a length]
instfield:    i    "small integer gives trouble"
   [i = 1 ⇒[↑self] user notify: 'arg too big']
subscripts: a
   [self cansubscript: a⇒[↑a∘self]
   user notify: 'Subscript out of bounds: ' + self asString]
subscripts: a ← val | t
   [self cansubscript: a⇒
      [t ← val asInteger.
       (a is: String) and: (t isnt: Integer)⇒
         [user notify: 'Improper store (non-char into String?)']
       ↑a∘self ← t]
   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⇒[self=¯32768⇒[strm append: '¯32768']
         strm append:'¯'. (0-self) printon: strm base: 10]
   self printon: strm base: 10]
printon: strm base: b | rem i x
   [[0>(x←self)⇒[i ← 1.
        digitbuffer◦1 ← 040000\b*2+self-0100000\b. "get it?"
        x ← (040000/b*2+(self-0100000/b))]
   i ← 0].
   while: x≥b do:
      [digitbuffer◦(i←i+1) ← x\b. x ← x/b].
   strm next ← 060+x.
   while: i≠0 do:
      [strm next ← 060+(digitbuffer◦i). i ← i-1].
   ]
```

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]
```

Initialization

```
classInit "Initialize the digit buffer"
   [digitBuffer ← String new: 16]
```

Compiler Bytecodes

```
asRemoteCode: generator
   [self<256⇒ [↑super asRemoteCode: generator]
   (self land: 0177400)≤codeLoadTemp⇒ [↑ParsedFieldReference new var: self];
      =codeLoadLitInd⇒ [↑ParsedObjectReference new var: self]
   ↑super asRemoteCode: generator]
bfpSize
   [↑[self<0⇒ [2]; >8⇒ [2] 1]]
emitBfp: code on: stack
   [stack pop: 1.
   0=self⇒ [code next ← toPop]
   1≤self and: self≤8⇒ [code next ← self+toShortBfp-1] "short bfp"
   code emitLong: toLongBfp by: self]
emitBytes: code | c t
   [self<256⇒ [code next ← self]
   c ← self lshift: ¯8. t ← self land: 0177.
   ↱(16 16 32 48 48)∘c > t⇒ [code next ← ↱(0 16 32 64 208)∘c+t]
   code next ← toLoadFieldLong+c-1; next ← t]
emitForValue: code on: stack
   [[self=toSuper⇒ [code next ← toLoadSelf] self emitBytes: code].
   stack push: 1]
emitJmp: code on: stack
   [0=self⇒ []
   1≤self and: self≤8⇒ [code next ← self+toShortJmp-1] "short jmp"
   code emitLong: toLongJmp by: self]
emitsLoad
   [self<256⇒ [↑self<toSmashPop] ↑self<codeSendLit]
emittedVariable
   [[self<256⇒[self≤toSuper] self<codeSendLit]⇒[] ↑false]
firstPush
isField
   [↑self≥codeLoadField and: self<codeLoadTemp]
jmpSize
   [↑[self=0⇒ [0]; <0⇒ [2]; >8⇒ [2] 1]]
sizeForValue
   [self<256 or:    ↱(16 16 32 48 48)∘(self lshift: ¯8) > (self land: 0177)⇒ [↑1]
   ↑2]
```

As yet unclassified

```
asInt32 [↑ Int32 new high: 0 low: self]
between: min and: max
   [↑self≥min and: self≤max]
```

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

"LargeInteger"

```
Class new title: 'LargeInteger'
   subclassof: Number
   fields: 'bytes     "The String of digits (between 0 and 127)"
            neg    "The sign" '
   declare: '';
   asFollows_!
```

LargeInteger is a class of Numbers with integral values of arbitrary precision. The values are stored as a String of digits between 0 and 127. Arithmetic is done by performing regular Integer arithmetic (¯32768 to 32767) on the digits. The digit size of 127 instead of 255 was chosen so that a multiplication of two digits would always remain in the regular Integer range. The first element of bytes contains the lowest precision digit.
As well as standard arithmetic functions, the LargeIntegers are equipped to perform various tests of their own primality

Access

```
bit: index | byte
   "Return bit number i in the binary representation of this number. Bit
number 1 is the low order bit"
   [byte ← bytes∘(1+((index-1)/7)).
    ↑(byte lshift: (0-((index-1)\7))) land: 1]
bits
   "Return the index of the high order bit of the binary representation of this
number"
   [↑bytes last hibit+(7*(bytes length-1))]
bytes "Return the string of digits"
   [↑bytes]
bytes: bytes neg: neg "Initialize this LargeInteger"
neg "Return the sign of this LargeInteger"
   [↑neg]
```

Arithmetic

```
* arg | prod prodin prodout mcand i x carry plier p
   [arg←arg asLarge. mcand ← arg bytes.
   prod ← String new: bytes length+mcand length. prod all← 0.
   prodin ← prod asStream. prodout ← prod asStream.
   for: i to: bytes length do:
      [prodin reset; skip: i-1. prodout reset; skip: i-1.
      plier ← bytes∘i. carry ← 0.
      for: x from: mcand do:
         [p ← x*plier+carry+prodin next.
         carry ← p lshift: ¯7. prodout next← p land: 0177].
      until: carry=0 do:
         [p ← carry+prodin next. carry ← p lshift: ¯7.
         prodout next← p land: 0177]].
   [prod last=0→[prod ← (prod∘(1 to: prod length-1)) copy]].
   ↑LargeInteger new bytes: prod neg: neg≠arg neg]
+ arg | shorter longer sl s sum i carry
```

```
    [arg←arg asLarge.
    neg⇒[arg neg⇒[↑(self negated + arg negated) negated]↑arg - self negated]
    arg neg⇒[↑self - arg negated]
    longer ← bytes. shorter ← arg bytes.
    [shorter length>longer length⇒[longer ← shorter. shorter ← bytes]].
    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>127⇒[s←s-128. 1] 0].
       sum next← s]
    [carry>0⇒[sum next← 1]].
    ↑LargeInteger new bytes: sum contents neg: false]
- arg | shorter longer sl s sum i borrow
    [arg←arg asLarge.
    neg⇒[arg neg⇒[↑arg negated - self negated]
          ↑(arg + self negated) negated]
    arg neg⇒[↑self + arg negated] arg>self⇒[↑(arg - self) negated]
    longer ← bytes. shorter ← arg bytes.
    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+128. ¯1] 0]. sum next← s].
    while: sum last=0 do: "trim leading zeroes"
       [sum skip: ¯1. sum empty⇒[↑0 asLarge]].
    ↑LargeInteger new bytes: sum contents neg: false]
/ arg | dividend m divin divout divisor n d1 quotient j q s carry digit
    [arg ← arg asLarge. n ← arg norm.
     dividend←self normalize: n. m←dividend length.
     divin←dividend asStream. divout←dividend asStream.
     divisor←arg normalize: n. n←divisor length. d1←divisor∘(n-1).
     n=2⇒[↑self digitDivide: arg bytes∘1]
     quotient←(String new: m-n+1) asStream.
     for: j from: (m to: n by: ¯1) do:
       [carry ← ((dividend∘j)*128)+(dividend∘(j-1)).
        [dividend∘j=d1⇒[q←127] q←carry/d1].
        while: divisor∘(n-2)*q>((carry-(q*d1))*128+(dividend∘(j-2))) do:
          [q←q-1]
        divin reset; skip: j-n. divout reset; skip: j-n. carry←0.
        for: digit from: divisor do:
          [s←divin next+carry-(q*digit). carry←s/128.
           (divout next← s\128)≠0 and: s<0⇒[carry←carry-1]]
        [carry=¯1⇒
          [divin reset; skip: j-n. divout reset; skip: j-n. carry←0. q←q-1.
           for: digit from: divisor do:
             [s←divin next+carry+digit. carry←[s>127⇒[s←s-128. 1] 0].
              divout next← s]]].
        quotient next← q]
     ↑LargeInteger new bytes: quotient contents reverse neg: arg neg≠neg]
< arg [(self compare: arg)=1⇒[↑self] ↑false]
= arg
```

```
    [arg isNumber⇒
      [(self compare: arg)=2⇒[↑self] ↑false]
     ↑false]
> arg [(self compare: arg)=3⇒[↑self] ↑false]
abs "Return the positive magnitude (absolute value) of this LargeInteger"
    [↑LargeInteger new bytes: bytes neg: false]
allmask: b [↑b = (self land: b)]
anymask: b [↑0 ≠ (self land: b)]
compare: arg | i a
    [arg←arg asLarge.
    neg⇒
      [arg neg⇒[↑arg comparemag: self] ↑1]
    arg neg⇒[↑3]
    a ← arg bytes.
    bytes length>a length⇒[↑3]; <a length⇒[↑1]
    for: i from: (bytes length to: 1 by: ¯1) do:
      [(bytes:i)>(a:i)⇒[↑3];
            <(a:i)⇒[↑1]]
    ↑2]
comparemag: arg | i a
    [a ← arg bytes.
     bytes length>a length⇒[↑3]; <a length⇒[↑1]
     for: i from: (bytes length to: 1 by: ¯1) do:
      [(bytes:i)>(a:i)⇒[↑3];
            <(a:i)⇒[↑1]]
     ↑2]
digitDivide: divisorDigit | carry quotient i test nextDigit
    [carry ← 0.
     quotient ← (String new: bytes length) asStream.
     for: i from: (bytes length to: 1 by: ¯1) do:
      [test ← (carry lshift: 7)+(bytes:i).
       nextDigit ← test/divisorDigit. nextDigit>128⇒[user notify: 'digit>128']
       carry ← test-(nextDigit*divisorDigit).
       quotient next ← nextDigit]
     quotient ← quotient contents.
     [quotient:1=0 and: quotient length>1⇒
      [quotient ← (quotient:(2 to: quotient length)) reverse]
      quotient ← quotient reverse].
     ↑LargeInteger new bytes: quotient neg: neg]
digitMod: divisorDigit | carry i test nextDigit
    [carry ← 0.
     for: i from: (bytes length to: 1 by: ¯1) do:
      [test ← (carry lshift: 7)+(bytes:i).
       nextDigit ← test/divisorDigit. nextDigit>128⇒[user notify: 'digit>128']
       carry ← test-(nextDigit*divisorDigit)]
     ↑carry asLarge]
divideAndCarry: divisorDigit | carry quotient i test nextDigit
    [carry ← 0.
     quotient ← (String new: bytes length) asStream.
     for: i from: (bytes length to: 1 by: ¯1) do:
      [test ← (carry lshift: 7)+(bytes:i).
       nextDigit ← test/divisorDigit. nextDigit>128⇒[user notify: 'digit>128']
```

```
        carry ← test-(nextDigit*divisorDigit).
        quotient next ← nextDigit]
    quotient ← quotient contents.
    [quotient∘1=0⊃[quotient ← (quotient∘(2 to: quotient length)) reverse]
     quotient ← quotient reverse].
    ↑carry,(LargeInteger new bytes: quotient neg: neg)]
drop: ndigits
   [↑LargeInteger new bytes: bytes∘(ndigits+1 to: bytes length) neg: neg]
field: n
   [↑self asSmall field: n]
field: n ← val
   [↑self asSmall field: n ← val]
land: n
   [↑self asSmall land: n]
lshift: n
   [↑self asSmall lshift: n]
negated
   [↑LargeInteger new bytes: bytes neg: neg≡false]
nomask: b [↑0 = (self land: b)]
norm
   [↑7-bytes last hibit]
normalize: n | norm digitsout digit carry
   [norm ← String new: bytes length+1.
    n=0⊃
      [bytes copyto: norm. norm last ← 0.
       ↑norm]
    digitsout ← norm asStream. carry ← 0.
    for: digit from: bytes do:
      [norm ← digit lshift: n.
       digitsout next ← (norm+carry) land: 127.
       carry ← digit lshift: n-7]
    digitsout next ← carry.
    ↑digitsout contents]
unnormalize: bytes with: n neg: neg | i carry
   [[n≠0⊃
       [carry ← bytes∘1 lshift: 0-n.
        for: i from: (2 to: bytes length) do:
          [bytes∘(i-1) ← ((bytes∘i lshift: 7-n)+carry) land: 127.
           carry ← bytes∘i lshift: 0-n]
        bytes last ← carry]].
    bytes last≠0⊃[]
    i←bytes length.
    while: bytes∘i=0 do: [i←i-1. i=0⊃[↑0 asLarge]].
    bytes←(bytes∘(1 to: i)) copy]
\ arg | dividend m divin divout divisor n d1 norm j q s carry digit
   [arg ← arg asLarge. norm ← arg norm.
    dividend←self normalize: norm. m←dividend length.
    divin←dividend asStream. divout←dividend asStream.
    divisor←arg normalize: norm. n←divisor length. d1←divisor∘(n-1).
    n>m⊃[↑self].
    [n=m⊃[self<arg⊃[↑self]]].
    n=2⊃[↑self digitMod: arg bytes∘1]
```

```
    bytes length=1⇒[↑self]
    for: j from: (m to: n by: ¯1) do:
      [carry ← ((dividendoj)*128)+(dividendo(j-1)).
       [dividendoj=d1⇒[q←127] q←carry/d1].
       while: divisoro(n-2)*q>((carry-(q*d1))*128+(dividendo(j-2))) do:
         [q←q-1]
       divin reset; skip: j-n. divout reset; skip: j-n. carry←0.
       for: digit from: divisor do:
         [s←divin next+carry-(q*digit). carry←s/128.
          (divout next← s\128)≠0 and: s<0⇒[carry←carry-1]]
       [carry=¯1⇒
         [divin reset; skip: j-n. divout reset; skip: j-n. carry←0. q←q-1.
          for: digit from: divisor do:
            [s←divin next+carry+digit. carry←[s>127⇒[s←s-128. 1]0].
             divout next← s] carry≠1⇒[user notify: 'no carry on add back']]]]
    ↑LargeInteger new unnormalize: (dividendo(1 to: n)) copy with: norm
      neg: arg neg≠neg]
```

Conversion

asFloat *"Built for comfort, not for speed"*

```
    [↑self asString asFloat]
```

asInteger | t i

```
    [bytes length>3⇒[↑self]
    t ← 0.
    for: i from: bytes length to: 1 by: ¯1 do:
      [t ← t*0200+(bytesoi)].
    neg⇒[↑0-t]
    ↑t]
```

asLarge

asSmall | t

```
    [t ← [bytes length≥3⇒[(byteso3 land: 1) lshift: 14]0].
     t ← t+[bytes length≥2⇒[(byteso2 lshift: 7)+(byteso1)]bytesо1].
     bytes length≥3⇒[(bytesо3 land: 2)=0⇒[↑t] ↑t+0100000] ↑t]
```

isLarge

Printing

printon: strm | p c

```
    [self=0⇒[strm append: '0']
     [neg⇒[strm append: '¯'. c← self negated]c←self].
     p←Stream default.
     while: c>0 do:
       [c←c divideAndCarry: 10.
        p next← co1+060.
        c←co2]
     strm append: p contents reverse]
```

spaceprinton: strm | p i c

```
    [self=0⇒[strm append: '0']
     [neg⇒[strm append: '¯'. c← self negated]c←self].
     p←Stream default. i←0.
     while: c>0 do:
       [c←c divideAndCarry: 10.
```

```
        p next← co1+060.
        c←co2.
        i←i+1. i=5⇒[p space. i←0]]
    strm append: p contents reverse]
┘
SystemOrganization classify: ↗LargeInteger under: 'Numbers'.┘
```

"Number"

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

Numbers in general

Arithmetic

```
< n [↑self - n < 0]
= n [↑self - n = 0]
> n [↑self - n > 0]
abs [self<0⇒[↑self * ¯1]]
between: min and: max
   [↑[min≤self and: self≤max]]
compare: i
   [self < i⇒ [↑1]
   self = i⇒ [↑2]
   ↑3]
factorial "I only work for positive integer values"
   [self=0⇒[↑1]
   ↑self * (self-1) factorial]
log2 | i cnt "floor of log base 2"
   [self < 0 ⇒[↑(self * ¯1) log2]
   self < 1 ⇒[↑((self/self) / self) log2 * ¯1]
   i ← 1. cnt ← 0.
   while: self ≥ i do: [i ← i+i. cnt ← cnt+1].
   ↑cnt-1]
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

```
cansubscript: a
```

```
    [↑self asInteger cansubscript: a]
subscripts: a
    [↑a∘self asInteger]
subscripts: a ← val
    [↑a∘self asInteger ← val]
```

Intervals, Points

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

Compatibility

```
isLarge
    [↑false]
isNumber
⌟
SystemOrganization classify: ↶Number under: 'Numbers'.⌟
```

"Time"

```
Class new title: 'Time'
   subclassof: Object
   fields: 'h m s'
   declare: '';
   asFollows_!
```

Implements times of day. Still needs a lot of work. (Steve Weyer)

```
Setting state
default "Makes self be the current time"
   [self fromSeconds: user totalsecs]
fromSeconds: sec [
   "Seconds since midnight, ignores date part"
   sec ← sec \ 86400.
   h ← (sec / 3600) asSmall.
   sec ← (sec \ 3600) asSmall.
   m ← sec / 60.
   s ← sec \ 60]
hours: h
minutes: m
seconds: s

Printing
printon: strm "Format is h:mm:ss am/pm" [
   strm print: [h>12⇒[h-12]; <1⇒[12] h];
      append: [m < 10⇒ [':0'] ':']; print: m;
      append: [s < 10⇒ [':0'] ':']; print: s;
      space append: [h<12⇒['am'] 'pm']]

Aspects
asSeconds [↑3600 * h + (60*m+s)]
hours [↑h]
_!
SystemOrganization classify: ↗Time under: 'Numbers'._!
```

"Array"

```
Class new title: 'Array'
   subclassof: Object
   fields: ''
   declare: '';
   asFollows_!
```

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

Reading and Writing

```
∘ x
   [↑x subscripts: self]
∘ x ← val
   [↑x subscripts: self ← val]
< v "for sorting vectors by first element"
   [↑(self∘1)<(v∘1)]
= arg | x
   [arg isArray⇒
      [self length ≠ arg length⇒ [↑false]
      for: x to: self length do:
         [(self∘x) = (arg∘x)⇒ [] ↑false]
      ↑true]
   ↑false]
> v "for sorting vectors by first element"
   [↑(self∘1)>(v∘1)]
all ← val | i
   [for: i to: self length do:
      [self∘i ← val]]
empty [↑self length=0]
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 s [
   x ← self species new: self length + arg length.
   self copyto: (s ← x asStream).
   arg copyto: s.
   ↑x]
copy
   [↑self copyto: (self species new: self length)]
copy: a to: b
   [↑self copy: a to: b to: (self species new: b-a+1)]
```

```
copy: a to: b to: t | i s me
   [s ← t asStream.
   me ← Stream new of: self from: a to: b.
   for: i from: a to: b do:          "general code wont stop at false"
      [s next ← me next]
   ↑t]
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)]
insertNonDescending: x          "self is assumed to be sorted"
   [↑self insertSorted: x]
insertSorted: x      | a c i      "self is assumed to be sorted"
   [i ← self findSorted: x.
   c ← (a ← self species new: self length+1) asStream.
   self∘(1 to: i-1) copyto: c. c next ← x. self∘(i to: self length) copyto: c.
   ↑a]
notNil | t i  "copy self (which contains no falses) removing all nils"
   [t ← (self species new: (self length-(self count: nil))) asStream.
   for: i from: self do: [i≡nil ⇒[] t next← i].
   ↑t asArray]
replace: a to: b by: s | x xs
   [x ← self species new: self length+s length -(1+b-a).
   xs ← x asStream.
   self∘(1 to: a-1) copyto: xs.
   s copyto: xs.
   self∘(b+1 to: self length) copyto: xs.
   ↑x]
without: index | s me i  "if index in range, return self without oindex"
   [index canSubscript: self⇒
      [s ← (self species new: self length-1) asStream.
      me ← self asStream.
      for: i to: self length do: [i=index⇒ [me next] s next ← me next].
      ↑s asArray]]
```

Searching

```
all: variable suchThat: expr | s i x  "a copy of some of me"
   [s ← (self species new: self length) asStream.
   for: i to: self length do:
      [x ← self∘i. variable value ← x.
      expr eval⇒ [s next ← x]].
```

```
    ↑s contents]
count: x | i n
    [n←0.
    for: i to: self length do:
        [x=(self∘i)⇒ [n←n+1]].
    ↑n]
findS: x suchThatS: predicate | i
    [for: i to: self length do:
        [x value ← self∘i. predicate eval⇒ [↑i]].
    ↑0]
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]
findSorted: x | lo mid hi          "self is assumed to be sorted"
    [hi ← self length+1. lo ← 1.
    while: lo < hi do:          "binary search"
        [self∘(mid←lo+hi/2) > x⇒[hi ← mid] lo ← mid+1].
    ↑hi]        " self∘hi > x, or, x should be inserted before self∘hi ".
firstS: x suchThatS: predicate | i
    [for: i to: self length do:
        [x value ← self∘i. predicate eval⇒ [↑self∘i]].
    ↑false]
has: x
    [self length=1⇒ [↑self∘1 = x]
    ↑(self find: x)≠0]
```

Permutation

```
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.]
swap: i with: j | t
    [t ← self∘i. self∘i ← self∘j. self∘j ← t]
```

Conversion

```
asSet [∩Set new of: self to: self length]
asStream
    [∩Stream new of: self]
frequencies | d x c  "return a sorted vector ((freq item) (freq item) ...)"
    [d ← Dictionary new init: 64.
    for: x from: self do:
        [c ← d lookup: x⇒ [d∘x ← c+1] d insert: x with: 1].
    ∩d asInvertedVector sort]
transform: each to: expr | s i  "a copy of me with each element transformed"
    [s ← (self species new: self length) asStream.
    for: i to: self length do:
        [each value ← self∘i. s next ← expr eval].
    ∩s asArray]
viewer [∩SetReader new of: self]
```

Mapping

```
canSubscript: a | i
    [for: i from: self do: [i canSubscript: a⇒ [] ∩false]]
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

isArray

isIntervalBy1

```
    [↑false]
```

species

```
    [↑Vector]
```

Comparing

hash *"make sure = arrays hash =ly"*

```
    [self length=0⇒[↑17171]
    ↑(self∘1) hash + (self∘self length) hash]
```

```
SystemOrganization classify: ↶Array under: 'Basic Data Structures'.
```

"FieldReference"

```
Class new title: 'FieldReference'
   subclassof: Object
   fields: 'object offset'
   declare: '';
   asFollows↵
```

I reference a field of an instance

Initialization
object: object offset: offset

Indirection
eval
```
   [↑object instfield: offset]
```
value [↑object instfield: offset]
value ← value
```
   [object instfield: offset ← value. ↑value]
↵
SystemOrganization classify: ↷FieldReference under: 'Basic Data
Structures'.↵
```

"Interval"

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

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

Initialization

```
from: start to: stop by: step
   [length ← 1+(stop-start/step).
   step<0⇒[start<stop⇒[length← 0]]
   stop<start⇒[length← 0]
   ]
```

Reading and Writing

```
o x
   [x is: Integer⇒[x<1⇒ [↑nil]
      x>length⇒ [↑nil]
      ↑start+(step*(x-1))]
   ↑super o x]
o x ← val
   [user notify: 'Intervals are not for writing into']
length [↑length]
start [↑start]
stop [↑stop]
```

Compatibility

```
canSubscript: a
   [↑length≤0 or: ((start canSubscript: a) and: (length-1*step+start
canSubscript: a))]
isIntervalBy1
   [↑step=1]
```

Random Numbers

```
random        "See Lehmers linear congruential method, Knuth Vol. 1:
   modulus m=2↑16
   a=27181 odd, and 5 = a mod 8
   c=13849 odd, and c/m around 0.21132"
   [step← (13849 + (27181*step)) asSmall.
   ↑(start + ((length asFloat*(32768.0+step))/65536.0)) asSmall]
randomInit [self randomInit: mem o 0430]
randomInit: x        "Call with const to get repeatable sequence"
   [step← x.        "step holds the current state"
   start is: Float⇒[length←stop-start]          "for Float intervals"]
↵
SystemOrganization classify: ↗Interval under: 'Basic Data Structures'.↵
```

"ObjectReference"

```
Class new title: 'ObjectReference'
   subclassof: Object
   fields: 'object'
   declare: '';
   asFollows↵
```

I am an indirect reference

Initialization
object: object

Indirection
eval
   [^object]
value [^object]
value ← object
   [^object]

Conversion
printon: strm
   [strm append: '->'; print: object]
↵
SystemOrganization classify: ↷ObjectReference under: 'Basic Data Structures'.↵

"PQueue"

```
Class new title: 'PQueue'
   subclassof: Stream
   fields: 'readposition'
   declare: '';
   asFollows
```

A PQueue is a First In First Out list of objects implemented as an array and a read pointer and write pointer. PQueue is a subclass of Stream and uses Stream's standard method for inserting a new item (next←, i.e. Stream's position is the write pointer). A PQueue also has a read pointer which it uses for accessing objects with the messages next or dequeue (which are identical). All messages to a PQueue that change its state are declared as critical sections to avoid race conditions

FIFO access

```
dequeue: num | n
   [Top critical:
      [position-readposition < num ⇒ [n ← false]
       n ← (array:(readposition+1 to: readposition + num)) copy.
       readposition ← readposition + num].
    ↑n]
length | l
   [Top critical: [l ← position-readposition]. ↑l]
myend [↑true]
next | n
   [Top critical:
      [readposition≥position⇒ [readposition←position←0. n ← false]
       n ← array:(readposition ← readposition+1)].
    ↑n] primitive: 98
pastend ← x      | n i    "simple arg"
   [Top critical:
      [position≥limit⇒
         [readposition=0⇒[super pastend ← x]
          n ← position-readposition.
          for: i to: n do: [array:i ← array:(readposition+i)].
          readposition ← 0. position ← n.
          self next ← x]
       array:(position ← position+1) ← x].
    ↑x]
peek | n
   [Top critical:
      [readposition≥position⇒ [readposition←position←0. n ← false]
       n ← array:(readposition + 1)].
    ↑n]
skip: x
   [Top critical: [readposition ← readposition+x]]
```

LIFO access

```
push: x          "treat as LIFO queue"
```

```
    [Top criticals
      [readposition > 0 ⇒
        [array∘readposition ← x.
         readposition ← readposition - 1]          "readpositon > 0, just jam it in"
       self insert: x]]                            "otherwise insert on front"
```

Stream protocol

```
contents | n
    [Top criticals [n ← (array∘(readposition+1 to: position)) copy]. ↑n]
empty | l
    [Top criticals [l ← readposition≥position]. ↑l] primitive: 99
end | n
    [Top criticals [n ← readposition≥position]. ↑n]
of: array
    [Top criticals [position ← 0. readposition ← 0. limit ← array length]]
of: array from: position to: limit
    [user notify: 'of:from:to: is not appropriate for PQueues']
reset
    [Top criticals [readposition ← position ← 0]]
⌟
SystemOrganization classify: ↗PQueue under: 'Basic Data Structures'.⌟
```

"Queue"

```
Class new title: 'Queue'
   subclassof: Stream
   fields: 'readposition'
   declare: '';
   asFollows
```

A Queue is a First In First Out list of objects implemented as an array and a read pointer and write pointer. Queue is a subclass of Stream and uses Stream's standard method for inserting a new item (next←, i.e. Stream's position is the write pointer). A Queue also has a read pointer which it uses for accessing objects with the messages next or dequeue (which are identical)

FIFO access

```
deQ1 | n    "A noninterruptable dequeue"
   [Top critical: [n ← self dequeue].
    ↑n]
dequeue
   [readposition≥position⇒ [readposition←position←0. ↑false]
    ↑array∘(readposition ← readposition+1)]
dequeue: num | n
   [position-readposition < num ⇒ [↑false]
    n ← (array∘(readposition+1 to: readposition + num)) copy.
    readposition ← readposition + num.
    ↑n]
enQ1: n    "A noninterruptable enqueue"
   [Top critical: [super next← n].
    ↑n]
length
   [↑position-readposition]
next
   [readposition≥position⇒ [readposition←position←0. ↑false]
    ↑array∘(readposition ← readposition+1)]
peek
   [readposition≥position⇒ [readposition←position←0. ↑false]
    ↑array∘(readposition + 1)]
skip: x
   [readposition ← readposition+x]
```

LIFO access

```
push: x              "treat as LIFO queue"
   [readposition > 0 ⇒
   [array∘readposition ← x. readposition ← readposition - 1]          "readpositon >
0, just jam it in"
   self insert: x]                                                   "otherwise insert on
front"
```

Stream protocol

```
contents
```

```
    [↑(array∘(readposition+1 to: position)) copy]
empty
    [↑readposition≥position]
end
    [↑readposition≥position]
of: array
    [position ← 0. readposition ← 0. limit ← array length]
of: array from: position to: limit
    [user notify: 'of:from:to: is not appropriate for Queues']
pastend ← x | n
    [readposition=0⇒[↑super pastend ← x]
     n ← position-readposition.
     array∘(1 to: n) ← array∘(readposition+1 to: position).
     readposition ← 0.
     position ← n.
     ↑self next ← x]
reset
    [readposition ← position ← 0]
┘
SystemOrganization classify: ↗Queue under: 'Basic Data Structures'.┘
```

"Set"

```
Class new title: 'Set'
   subclassof: Stream
   fields: 'views'
   declare: '';
   asFollows↵
```

For storing/collecting, read by a SetReader.
Use no messages from Stream except of:, empty, next←, contents, space, nextword←

Initialization

```
default [self vector: 8]
of: array to: position [limit ← array length]
string: limit [self of: (String new: limit)]
vector: limit [self of: (Vector new: limit)]
```

Index operations

```
oi [↑array∘(self checkIndex: i)]
oi ← val [↑array∘(self checkIndex: i) ← val]
deleteI: i | r v [
   v ← self∘i.
   r ← self viewRange: i+1 to: position.
   position ← i-1.
   self append: r.
   array∘(position+1) ← nil.
   [views≡nil⇒ []
   for: r from: views do: [r of: array to: position; changed: i by: ¯1]].
   ↑v]
insertI: i value: v | r [
   [i > position⇒ [self next ← v]
   "old contents in set"
   r ← [position = limit⇒ [self grow] self growby: 0].
   self append: (r viewRange: 1 to: i-1);
      next ← v;
      append: (r viewRange: i to: r length).
   views≡nil⇒ []
   for: r from: views do: [r of: array to: position; changed: i by: 1]].
   ↑v]
```

Value operations

```
append: x  [for: x from: x do: [self next ← x]]
delete: x | i [
   for: i to: position do: [
      array∘i ≡ x⇒ [↑self deleteI: i]].
   ↑false]
find: v | i [
   for: i to: position do: [array∘i = v⇒ [↑i]].
   ↑0]
```

Viewing

```
asSet
asStream [↑self viewer]
copy [↑self viewer copy]
initView: v [↑v of: array to: position]
length [↑position]
notViewed: v [
   views delete: v;
      empty⇒ [views ← nil]]
species [↑array species]
viewer [
   ↑SetReader new of: array from: 1 to: position
   "self viewRange: 1 to: position"]
viewer: v [
   [views≡nil⇒ [views ← Set default]].
   views next ← v]
viewRange: i to: j [
   ↑"self viewer:" (
   SetReader new of: array from: (i "max: 1") to: (j "min: position"))]
```

Private

```
checkIndex: i [
   i between: 1 and: position⇒ [↑i]
   ↑user notify: 'illegal index']
grow [
   "self grown and reset. returns another Set with old contents"
   ↑self growby: (10 max: limit/4)]
growby: n | old [
   "grow and reset self. return old Set for copying"
   old ← Set new of: array to: position.
   self of: (array species new: limit+n) to: 0.
   ↑old]
next [user notify: 'no direct reading of a Set']
pastend ← x [
   ↑[self append: self grow; next ← x]]
sort | pos sort [
   "no readers"
   array sort: 1 to: position;
      o1 is: Interval⇒ [
      pos ← 1.
      while: pos < position do: [
         (sort ← array∘pos compare: array∘(pos+1))
            =1⇒ ["<" pos ← pos+1];
            =3⇒ [">"
               "array swap: pos with: pos+1"
               user notify: '3> error']
         [sort
            =2⇒ ["second = or contained in first"];
            =4⇒ ["included"
               array∘pos ← array∘(pos+1)];
            =5⇒ ["first merge with second"
               array∘pos ← array∘pos merge: array∘(pos+1)]
```

```
            "=6 second merge with first"
               "array∘pos ← array∘(pos+1) merge array∘pos"
               user notify: '6> error'].
         self deletel: pos+1 ]]]
```

```
SystemOrganization classify: ↱Set under: 'Basic Data Structures'.
```

"SetReader"

```
Class new title: 'SetReader'
   subclassof: Stream
   fields: ''
   declare: '';
   asFollows↵
```

Read a Set; no edits occur to set. (see Steve for ISetReader (interruptible))
Inherit of:from:to:, next, next:, end, pastend, skip:, ◁, asStream, viewer

Initialization

```
of: array from: position for: n [
   position ← position-1.
   limit ← position+n]
```

Reading

```
asSet [↑self copy]
copy "yield contents all at once as a Set" [
   ↑[Set new of: (array species new: limit-position); append: self]]
length [
   "how much left"
   ↑limit-position]
↵
SystemOrganization classify: ↷SetReader under: 'Basic Data Structures'.↵
```

"Stream"

```
Class new title: 'Stream'
   subclassof: Object
   fields: 'array position limit'
   declare: '';
   asFollows↵
```

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

Initialization

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

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]
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]
into: x | i          "Array result"
   [for: i to: x length do:
      [x∘i ← self next].
   ↑x]
next    "simple result"
   [self myend⇒ [↑self pastend]
   ↑array∘(position ← position+1)] primitive: 17
next: x | t    [    "Array result"
   t ← array species new: x.
   for: x to: x do: [t∘x ← self next].
   ↑t]
```

```
nextNumber [↑self nextNumber: 1]
nextNumber: n | val t [
   "return n words as a positive Integer or LargeInteger"
   val ← 0.
   for: n to: n*2 do: [t ← self next⇒ [val ← 256 * val + t] ↑false].
   ↑val]
nextNumber: n ← val [
   "write n words of a positive Integer or LargeInteger"
   "kludge for case of n=1 and 2"
   [n=2⇒ [
      self nextword ← [
         (val is: Integer) or: val < 65536⇒[0] (val/65536) asSmall]]].
   self nextword ← val asSmall]
nextNumber← val [self nextword ← val asSmall "nextNumber: 1 ← val"]
nextPoint | x [
   x ← self nextword.
   ↑Point new x: x y: self nextword]
nextPoint←p [
   self nextword ← p x;
      nextword ← p y]
nextString | len [
   ↑self next: [
      (len ← self next)
         <192⇒[len]    "up to 191 chars (BCPL compat)"
      len-192*256 + self next]]        "up to 16383 chars"
nextString← s | len [
   [(len ← s length) < 192⇒[self next← len]
   self next← len/256+192; next← len\256].
   self append: s.
   ↑s]
nextword | hi lo
   [hi ← self next⇒
      [lo ← self next⇒
         [↑(hi lshift: 8)+lo]
      ↑false]
   ↑false]
nextword← val
   [self next← val lshift: ¯8.
   self next← val land: 0377. ↑val]
next ← x       "simple arg"
   [self myend⇒ [↑self pastend ← x]
   ↑array∘(position ← position+1) ← x] primitive: 18
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]
```

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

Test and alter position

```
empty          "for"
   [↑position=0]
end
   [↑position≥limit]
limit
   [↑limit]
loc    "synonym for compiler"
   [↑position]
myend
   [↑position≥limit]
pastend
   [↑false]
pastend ← x
   [array ← array grow. limit ← array length.
   ↑self next ← x]
position
   [↑position]
positioneven: nul
   [position allmask: 1⇒ [self next ← nul]]
position← position
reset
   [position ← 0]
settoend [position← limit]
skip: x
   [position ← position+x]
skipwords: w [self skip: 2*w]
wordposition [↑self position/2]
wordposition← w [self position← w*2]
```

Static reading and writing

```
o x
   [↑array o x]
o x ← val
   [↑array o x ← val]
contents | a i "↑(array o(1 to: position)) copy -- written out for speed"
   [a← array species new: position.
   for: i to: position do:
      [a o i← array o i].
   ↑a]
first
   [position ≠ 0 ⇒ [↑array o 1] ↑nil]
insert: x     | a     i                "treat as LIFO queue, insert in front"
   [position = 0 ⇒ [↑self next ← x]                    "if at the beginning, next←
```

```
works"
   a ← array species new: (limit ← position+1 max: limit).        "otherwise make a
new one big enough"
   a next ← x.                                                    "jam in x"
   for: i to: position do: [a next ← array∘i].                    "copy into the old guy"
   array ← a.                                                     "make new one be array"
   ↑x                                                             "and return x"]
last
   [position≠ 0 ⇒ [↑array∘position] ↑nil]
last: n
   [↑(array∘(position-n+1 to: position)) copy]
rest [↑(array∘(position+1 to: limit)) copy]
```

Character printing

```
cr
   [self next ← 015]
print: obj
   [obj printon: self]
semicrtab
   [self append: ';
   ']
space
   [self next ← 040]
tab
   [self next ← 011]
```

Coercions

```
asArray
   [↑array]
asStream
asText [↑self contents asText]
asVector "Convert a string to a vector of tokens"
   [↑(Reader new of: self) read]
viewer [↑Stream new of: array from: 1 to: position]
```

Compiler object code

```
emitLong: jmpOrBfp by: dist
   [[dist<0⇒ [dist←dist+1024]; >1023⇒ [dist←¯1] jmpOrBfp←jmpOrBfp+4].
   dist<0⇒ [user notify: 'A block compiles more than 1K bytes of code']
   self next ← dist/256 + jmpOrBfp. self next ← dist\256]
┘
SystemOrganization classify: ↗Stream under: 'Basic Data Structures'.┘
```

"String"

```
VariableLengthClass new title: 'String'
   subclassof: Array
   fields: ''
   declare: '';
   bytesize: 8;
   asFollows_!
```

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

Reading and Writing

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

Copying and Altering

```
+ arg
   [↑self concat: arg]
findString: str startingAt: start | i t
   [str length=0⇒[↑0] t← str↓1.
   for: i from: start to: self length-str length+1 do:
      [self↓i=t⇒[self↓(i to: i+str length-1)=str⇒[↑i]]]
   ↑0]
recopy
   [↑self copy]
replace: a to: b by: s | t [
   Stream new
      of: (t ← String "self species" new: self length+s length - (1+b-a));
      append: self↓(1 to: a-1);
      append: s;
      append: self↓(b+1 to: self length).
   ↑t]
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]
```

Comparison

```
- s | i c ldiff     [
   "Return a negative, zero, or positive integer as I compare < = or > s"
```

```
   "The collation sequence is ascii with case differences ignored."
   for: i to: [
      (ldiff ← self length-s length) < 0⇒ [self length] s length] do: [
      (c← UpperCase∘(self∘i + 1) -(UpperCase∘(s∘i + 1)))
         ≠0⇒ [↑c*2]].
   ↑ldiff*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]
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 | l m
   [[(l← m← self length)≤2⇒
      [l=2⇒[m←3]; =1⇒[↑((self∘1) land: 0177)*0152] ↑052525]].
    ↑(self∘1)*060+(self∘(m-1)+l)]
match: text | star pound pattern scanning p t back [
   star ← 052 "*".  pound ← 043 "#".
   pattern ← self asStream.  text ← text asStream.
   scanning ← false.
   while: true do: [
      (p ← pattern next)
         =star⇒ [pattern end⇒ [↑true] scanning ← pattern position]
      (t ← text next)
         ≡false⇒ [↑t≡p]
      p≡false⇒ [
         scanning⇒ [
            back ← scanning - pattern position.
            pattern skip: back. text skip: back]
         ↑false]
      UpperCase∘(t+1) = (UpperCase∘(p+1)) or: p=pound⇒ []
      scanning⇒ [
         back ← scanning - pattern position.
         pattern skip: back. text skip: back+1]
      ↑false]]
systemRehash | dicts d left loop
   [String understands: '

hash | l m
   [[(l← m← self length)≤2⇒
      [l=2⇒[m←3]; =1⇒[↑((self∘1) land: 0177)*0152] ↑052525]].
    ↑(self∘1)*060+(self∘(m-1)+l)]

'.   ↷a rehash.
```

```
    dicts ← (HashSet allInstances+Dictionary allInstances
              +SymbolTable allInstances) notNil.
    for: d from: dicts do:
      [left ← d objects asStream. loop ← left next.
       while: loop do:
         [loop is: String⇒[d rehash. loop ← false]
          loop ← left next]]]
```

Conversion

```
asBCPL: len | s i
   [s ← String new: len.
   so1← self length.
   for: i to: self length do: [so(i + 1) ← selfoi].
   for: i from: (self length+2 to: s length) do: [soi ← 0].
   ↑s]
asBytes | s c
   [s ← Stream default.
   for: c from: self do:
      [s append: c base8; space]
   ↑s contents]
asDecimalDigits "Not asInteger, because the result may be a Float if it's too big"
      | strm sign c val
   [strm← Stream new of: self.
   sign← strm◁025.
   val← [self length>4⇒[0.0]0].
   for: c from: strm do:
      [c<060 or: c>071⇒[user notify: self + ' isn''t a valid integer']
      val← val*10+(c-060)]
   sign⇒[↑val*¯1]
   ↑val]
asEntityInFont: fontname [
   ↑TextEntity new text: self style: [Style new withfont: fontname]]
asEntityInStyle: style [↑TextEntity new text: self style: style]
asFileName
   [↑File new namecheck: self fixing: true]
asFloat | strm int frac exp
   [strm← Stream new of: self.
   int← strm upto: 056.
   frac← strm upto: 0145.
   exp← strm rest asInteger - frac length.
   int← (int concat: frac) asDecimalDigits asFloat.
   exp=0⇒[↑int];
      >0⇒[↑int*(10.0 ipow: exp)].
   ↑int/(10.0 ipow: 0-exp)
   ]
asInteger | strm sign base maxdigit c val
   [strm← Stream new of: self.
   sign← [strm◁025⇒[¯1]1].
   base← [strm◁060⇒[8]10]. maxdigit← 060+base.
   val← [self length>4⇒[0.0]0].
   for: c from: strm do:
      [c<060 or: c≥maxdigit⇒[user notify: self + ' isn''t a valid Integer']
```

```
      val← val*base+(c-060)]
   "Some special maneuvering to keep 01dddddd and ¯32768 (and nothing else)
from overflowing."
   [val>077777⇒[base=8⇒[sign=1⇒[val<65536.0⇒[↑(val-65536.0) asInteger]]]]].
   ↑(val*sign) asInteger]
asLarge "convert to a LargeInteger"
   | neg i large large10
   [[self∘1=025⇒[neg←true] neg←false].
    large ← 0 asLarge. large10 ← 10 asLarge.
    for: i from: [neg⇒[2]1] to: self length do:
      [large ← (large*large10)+(self∘i-060)].
    neg⇒[↑large negated] ↑large]
asParagraph
   [↑Paragraph new text: self alignment: 0]
asText [↑[self is: String⇒ [self]; asString]]
asUppercase | s c
   [s ← Stream default.
   for: c from: self do:
      [s next ← UpperCase∘(c+1)]
   ↑s contents]
asVector
   [↑self asStream asVector]
base8: i "word: i in base 8 as a String"
   [↑(self word: i) base8]
hasBeenUniqued
   [↑↗a hasInterned: self]
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]
unique | u          "copy and intern"
   [↑↗a intern: self]
```

Compatibility

```
species
   [↑String]
```

System primitives

```
lock [] primitive: 31
unlock [] primitive: 32
┘
SystemOrganization classify: ↗String under: 'Basic Data Structures'.┘
```

"Substring"

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

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

Initialization

```
data: data map: map
```

Reading and Writing

```
o x
   [↑data o (map o x)]
o x ← val
   [↑data o (map o x) ← val]
length
   [↑map length]
map
   ["Return my map."
   ↑map]
```

Copying and Altering

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

Conversion

```
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]
asText [↑self copy asText]
```

Compatability

```
species
   [↑data species]
↵
SystemOrganization classify: ↷Substring under: 'Basic Data Structures'.↵
```

"UniqueString"

```
VariableLengthClass new title: 'UniqueString'
  subclassof: String
  fields: ''
  declare: '';
  bytesize: 8;
  asFollows↵
```

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

Initialization

```
classInit | i a v          "make up table of 1-char atoms"
   [v ← Vector new: 128. a ← String new: 1.
   for: i to: 128 do:
      [a↑1 ← i-1. v↑i ← a unique]
   UST1 ← v]
hasInterned: s | h i v n
      "↑false if String s hasnt been interned, else ↑s unique"
   [   [s length=1⇒[s↑1<128⇒[↑UST1↑(s↑1+1)]]].
   h← s hash.
   v ← USTable↑(h\USTable length+1).
   i← h\v length+1.
   for: n to: v length do:
      [v↑i≡nil⇒    [↑false]
      [s length=(v↑i) length⇒ [s=(v↑i)⇒[↑v↑i]]].
      i← [i=v length⇒[1] i+1]]
   user notify: 'USTable is jammed']
intern: s | h i j v n
   [   [s length=1⇒[s↑1<128⇒[↑UST1↑(s↑1+1)]]].
   h← [s is: String⇒[s hash] s stringhash].
   v ← USTable↑(h\USTable length+1).
   i← h\v length+1.
   for: n to: v length do:
      [v↑i≡nil⇒                          "empty slot"
         [n← ¯4. for: j from: v do: [j≡nil⇒ [n ← n+4]].
         n < v length⇒                   "grow bucket if > 3/4 full"
            [USTable↑(h\USTable length+1) ← Vector new: 2*v length.
            for: n from: v do:           "rehash all its contents"
               [n≡nil⇒ [] self intern: n]
            ↑self intern: s]
         ↑v↑i ← [s is: UniqueString⇒[s]          "install new entry"
                  (UniqueString new: s length) str: s]]
      [s length=(v↑i) length⇒ [s=(v↑i)⇒[↑v↑i]]].
      i← [i=v length⇒[1] i+1]]
   user notify: 'USTable is jammed']
rehash | oldTable v i
   [oldTable ← USTable.
    USTable ← Vector new: oldTable length.
    for: i to: USTable length do:
      [USTable↑i ← Vector new: 4].
    for: v from: oldTable do:
```

```
      [for: i from: v do:
         [i≡nil⇒[] self intern: i]]]
str: s | j
   [for: j to: s length do:
      [super:j ← s:j]
   ↑self]
unique
```

Reading and Writing

```
:x ← val
   [user notify: 'UniqueStrings are not for writing into']
```

Selectors

```
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]
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, dots, and arrows"
   for: i to: len do: [self:i=072⇒ [n←n+1]; =03⇒ [n←n+1]; =0137⇒[n←n+1];
=07⇒[n←n+1]].
   ↑n]
```

Comparison

```
= x [↑self≡x]
hash [] primitive: 46
stringhash
   [↑super hash]
```

Compatibility

```
copy [↑self]
recopy [↑self]
species
   [↑String]
```

Conversion

```
asString
[↑super copy]
printon: strm
   [strm append: self]
```

⌟

```
SystemOrganization classify: ↑UniqueString under: 'Basic Data Structures'.↵
UniqueString classInit↵
```

"Vector"

```
VariableLengthClass new title: 'Vector'
   subclassof: Array
   fields: ''
   declare: '';
   asFollows↵
```

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

Microcode Primitives

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

Reading and Writing

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

Copying and Altering

```
, x | v
   [v ← self growby: 1.      "use a stream if youre in a hurry"
   v last ← x. ↑v]
```

Searching

```
max| biggest i
   [biggest ← self o 1.      "return largest value in a vector"
   for: i to: self length do:
      [(self o i) > biggest ⇒[biggest ← self o i]].
    ↑biggest]
```

Conversion

```
asVector
printon: strm | i
   [strm append: '('.
   for: i to: self length do:
      [strm print: self o i; space]
   strm append: ')' ]
```

System primitives

```
nail [user croak] primitive: 31          "Nail me in core and return my core
address"
unNail [user croak] primitive: 32        "Release me from being nailed"
```

Compiler argument list

```
argsOff: stack
   [stack pop: self length]
emitForValue: code on: stack  | x
   [for: x from: self do: [x emitForValue: code on: stack]]
firstPush
   [↑(self-1) firstPush]
remote: generator  | x
   [for: x from: self do: [x remote: generator]]
sizeForValue  | size x
   [size ← 0. for: x from: self do: [size ← size+ x sizeForValue]. ↑size]
┘
SystemOrganization classify: ↗Vector under: 'Basic Data Structures'.┘
```

"ClassOrganizer"

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

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

Initialization

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

Access to parts

```
asStream | v t
   [v ← Vector new: 0.
   for: t from: groupVector do:
      [v ← v concat: t]
   ↑v asStream]
categories [↑commentVector]
category: str | i
   [i ← commentVector find: str.
   i=0⇒[user notify: 'No such category: '+str]
   ↑groupVector•i]
classify: selector under: heading | s h n
   [selector is: Vector⇒
      [for: s from: selector do:
         [self classify: s under: heading]]
   s ← commentVector find: heading.
   s>0 and: (groupVector•s has: selector)⇒[↑self]
   [h ← self invert: selector⇒
      [heading=default⇒[↑self]
       n ← commentVector find: h.
       groupVector•n ← groupVector•n delete: selector]].
   [s=0⇒ [s ← self insert: heading]].
   groupVector•s ← groupVector•s insertSorted: selector.
   n ← commentVector find: default.
   n>0 and: (groupVector•n) length=0⇒
      [self deleteCategory: n]]
compress
   [globalComment ← self globalComment asParagraph compressIntoString]
delete: selector | i        "delete this from all categories"
   [for: i to: groupVector length do:
      [groupVector•i has: selector⇒
```

```
        [groupVector↓i ← (groupVector↓i) delete: selector.
        (groupVector↓i) length=0 and: commentVector↓i=default⇒
          [self deleteCategory: i]]
    ]]
deleteCategory: index
  [groupVector ← groupVector without: index.
  commentVector ← commentVector without: index]
globalComment
  [↑(Paragraph new unpackFromString: globalComment) text]
globalComment ← str
  [globalComment ← str asParagraph packIntoString]
has: sel | t
  [for: t from: groupVector do:
    [t has: sel⇒[↑true]]
  ↑false]
insert: heading | di dgroup hi "force default category to end, delete if empty"
  [[(di←commentVector find: default)>0⇒ [dgroup ← groupVector↓di]].
  commentVector ← (commentVector without: di), heading.
  groupVector ← (groupVector without: di), (Vector new: 0).
  hi ← commentVector length.
  di=0 or: dgroup length=0⇒ [↑hi]
  commentVector ← commentVector, default.
  groupVector ← groupVector, dgroup.
  ↑hi]
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: self 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.
  self 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]
  ]]
```

```
SystemOrganization classify: ↗ClassOrganizer under: 'Sets and
Dictionaries'.
```

ClassOrganizer classInit↵

"Dictionary"

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

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

Initialization

```
copyfrom: dict
   [self objects ← dict objects copy.
   values ← dict values copy]
init: size
   [values ← Vector new: size. super init: size]
```

Searching

```
o name
   [↑values o (self findorerror: name)]
o name ← val
   [↑values o (self findorerror: name) ← val]
lookup: name | x
   [x ← self find: name⇒ [↑values o x] ↑false]
```

Inserting and Deleting

```
clean | name      "release unreferenced entries"
   [for: name from: self do:          "slick, huh"
      [(self o name) refct = 1 ⇒ [self delete: name]]]
delete: name
   [name is: Vector⇒[super delete: name]
   values o (self findorerror: name)← nil.
   super delete: name]
insert: name with: value
   [self insert: name.
   values o (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 o i with: vals o i]]
with: names values: vals | i
   [for: i to: names length do:
      [self insert: names o i with: vals o i]]
```

Private

```
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 o name]       "hash each entry into it"
```

```
    self copyfrom: copy]
rehash | i copy
    [copy ← Dictionary new init: self size.        "create a copy"
    for: i to: objects length do:
        [objects:i≡nil:⇒[]
         copy insert: objects:i with: values:i]        "hash each entry into it"
    self copyfrom: copy]
swap: i with: j
    [values swap: i with: j.
    super swap: i with: j]
values [↑values]
```

Inversion

```
asInvertedVector | s i v  "in form ((value, object), ...)"
    [s ← (Vector new: objects length) asStream.
    for: i to: objects length do:
        [objects:i≡nil:⇒ []
        v ← Vector new: 2. v:1←values:i. v:2←objects:i.
        s next ← v].
    ↑s contents]
invert
    [↑self invertto: (Dictionary new init: objects length)]
invert: obj | i
    [for: i to: values length do:
        [values:i=obj:⇒ [↑objects:i]]
    ↑false]
invertto: dict | i
    [for: i to: objects length do:
        [objects:i≡nil:⇒ []
        dict insert: values:i with: objects:i].
    ↑dict]
!
SystemOrganization classify: #Dictionary under: 'Sets and Dictionaries'.!
```

"HashSet"

```
Class new title: 'HashSet'
   subclassof: Object
   fields: 'objects'
   declare: '';
   asFollows↓
```

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

Initialization

```
copy "↑ a copy of me"
   [↑self class new copyfrom: self]
copyfrom: hset "take on state of hset"
   [objects ← hset objects copy]
init
   [self init: 4]
init: size
   [objects ← Vector new: (size max: 2)]
```

Access to parts

```
asStream
   [↑self contents asStream]
contents | obj strm
   [strm ← (Vector new: objects length) asStream.
   for: obj from: objects do:
      [obj≡nil⊃[] strm next← obj]
   ↑strm contents]
size [↑objects length]
```

Searching

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

Insertion and deletion

```
delete: obj | i j l
   [obj is: Vector⊃[for: i from: obj do: [self delete: i]]
   i← self findorerror: obj.
   objects∘i← nil.
```

```
    l← objects length.
    until: objects∘(i← [i=l⇒[1] i+1])≡nil do:
        [i=(j← self findornil: objects∘i)⇒[]
        self swap: i with: j]
    ]
insert: obj | i
    [self findorinsert: obj. ↑obj]
insertall: objs | x
    [for: x from: objs do: [self insert: x]]
```

Growing and shrinking

```
packprobes | tot n l i obj          "↑(fullness, avg #probes)"
    [tot ← n ← 0. l ← objects length.
    for: i to: l do:
        [(obj← objects∘i)≡nil⇒[]
        tot ← i-obj hash\l+tot.  n← n+1]
    n=0⇒[↑(1,1)]
    ↑((n asFloat/l) , (tot asFloat/n))]
"Class md packprobes(0.4921875 2.53968255 )"
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]
```

Private

```
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]
objects [↑objects]
objects← objects
rehash | i copy
    [copy ← HashSet new init: self size.        "create a copy"
```

```
    for: i to: objects length do:
        [objects:o i≡nil:⊃[]
         copy insert: objects:o i]          "hash each entry into it"
    objects ← copy objects]
sparse | i n
    ["↑true if (1 max: 1/4 of table) is nil"
    n ← objects length.
    for: i to: objects length do:
        [objects:o i≡nil:⊃[(n←n-4)≤0:⊃[↑true]]]
    ↑false]
swap: i with: j
    [objects swap: i with: j]
┘
SystemOrganization classify: ↗HashSet under: 'Sets and Dictionaries'.┘
```

"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"'
   declare: '';
   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

```
copyfrom: dict
   [self objects ← dict objects copy.
   methods ← dict methods copy.
   code ← dict code copy]
init: size
   [methods ← Vector new: size.
   code ← Vector new: size.
   super init: size]
```

Inserting and Deleting

```
close | i        "recycle all code and literals pointed to"
   [forS i to: methods length doS
      [methodsoi≡nil⇒[]
      self freeMethod: methodsoi]
   self init]
delete: name | i
   [i ← self findorerror: name.
   self freeMethod: methodsoi.
   methodsoi← codeoi← nil.
   super delete: name]
insert: name method: m literals: l
      code: c backpointers: b | i copy
   [i ← self find: name⇒          "if name is already there"
      [self freeMethod: methodsoi.
      self holdLiterals: l.
      methodsoi ← m. codeoi ← 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]
purge: sel ["demand purging invalidates checkpointing"]
```

Access to parts

```
code
    [∩code]
code: name
    [∩code∘(self findorerror: name)]
code: name ← str
    [∩code∘(self findorerror: name) ← str]
invert: method | i
    [for: i to: methods length do:
        [methods∘i≡method⇒ [∩objects∘i]].
    ∩false]
literals: name
    [∩self literalsIn: methods∘(self findorerror: name)]
method: name
    [∩methods∘(self findorerror: name)]
methodorfalse: name | i
    [i ← self find: name⇒[∩methods∘i] ∩false]
methods
    [∩methods]
```

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]
freeMethod: m | v c i t          "method pointed to by some vector (dict or keeper)
        and (upon entry) by m.  If any other owners, refct will be >2.
        *Expects Interpreter to nil args on callers stack*"
    [m refct>2⇒[MethodKeeper next← m]          "keep it"
    v← self literalsIn: m.          "free its literals"
    v length=0⇒[]
    c ← v nail.          "fasten seat belts"
    for: i to: v length do:          "lower refct of each literal"
        [t ← mem∘(c+i-1). v∘i ← nil. mem∘(c+i-1) ← t]
    v unNail]
freeMethods | v i          "Free kept methods no longer used"
    [v← MethodKeeper contents.
    MethodKeeper← (Vector new: 10) asStream.
    for: i to: v length do:
        [self freeMethod: v∘i]
    ]
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. memo@(m+i-1) ← -1. v@i ← t]
    v unNail]
holdMethods: v | i "a random insertion just to make it legal form"
    [for: i to: v length do:
        [self insert: i method: v@i literals: nil code: nil backpointers: nil]]
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]
```

Private

```
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]
swap: i with: j
    [methods swap: i with: j.
    code swap: i with: j.
    super swap: i with: j]
┘
SystemOrganization classify: ↶MessageDict under: 'Sets and Dictionaries'.┘
```

"SymbolTable"

```
Class new title: 'SymbolTable'
   subclassof: Dictionary
   fields: ''
   declare: '';
   asFollows_!
```

I associate each of my objects with an object reference

Access to parts

```
ref: name
   [↑super◦name]
ref: name ← val
   [↑super◦name ← val]
```

Searching

```
◦ name
   [↑(super◦name) value]
allCallsOn: selector from: classNames | className s w cl sel
   [[selector is: Vector⇒ [] selector ← selector inVector].
   s ← Stream default.
   user displayoffwhile:
      [for: className from: classNames do:
         [cl ← self◦className.
         for: sel from: selector do:
            [w ← cl whosends: sel. w length=0⇒ []
            s append: className; append: '⇒'; append: w asString; cr]]].
   ↑s contents]
allRefs "what methods reference my variables (I am probably 'Undeclared')"
   [↑self allRefsTo: self contents from: user classNames]
allRefsTo: symbol from: classNames | s
   [[symbol is: Vector⇒ [] symbol ← symbol inVector].
   ↑Smalltalk allCallsOn: (symbol transform: s to: (self ref: s)) from:
classNames]
invert: obj | i
   [for: i to: values length do:
      [nil≡(values◦i)⇒[]
      obj ≡ (values◦i) value ⇒[↑objects◦i]]
   ↑false]
invertRef: obj | i
   [for: i to: values length do:
      [obj≡(values◦i)⇒[↑objects◦i]]
   ↑false]
lookup: name | r
   [r←super lookup: name⇒[↑r value] ↑false]
lookupRef: name
   [↑super lookup: name]
```

Insertion

```
◦ name ← x
```

```
    [↑(super◦name) value ← x]
declare: name      "Take ref(s) and value(s) from Undeclared, if name(s) there"
    [self declare: name from: Undeclared]
declare: name as: x | a s
    [name is: Vector⇒
        [s ← x asStream. for: a from: name do: [self declare: a as: s next]]
    self declare: name.
    self◦name ← x]
declare: name from: symTab | a "take name(s), ref(s) and value(s) from
symTab"
    [name is: Vector⇒ [for: a from: name do: [self declare: a from: symTab]]
    self has: name⇒ []
    symTab has: name⇒
        [super insert: name with: (symTab ref: name).
        symTab delete: name]
    self insert: name with: nil]
define: name as: x        "synonym"
    [↑self declare: name as: x]
insert: name with: x
    [   [self has: name⇒[]
        super insert: name with: ObjectReference new].
    self◦name ← x]
insert: name withref: ref
    [super insert: name with: ref]
```

Growing and shrinking

```
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]
rehash | i copy
    [copy ← SymbolTable new init: self size.        "create a copy"
    for: i to: objects length do:
        [objects◦i≡nil⇒[]
        copy insert: objects◦i withref: values◦i]        "hash each entry into it"
    self copyfrom: copy]
┘
SystemOrganization classify: ↶SymbolTable under: 'Sets and Dictionaries'.┘
```

```
                                                    "SystemOrganizer"
Class new title: 'SystemOrganizer'
   subclassof: ClassOrganizer
   fields: ''
   declare: '';
   asFollows↵

Provides an organization for the classes in the system just as ClassOrganizer
organizes the messages within a class.  (In fact, the only difference is the
filout/printing messages.)

Filout and printing
filoutCategory: cat | f all a
   [user displayoffwhile:
      [f ← dpo file: (cat+'.st.') asFileName.
      all ← self superclassOrder: cat.
      f filoutclass: all.
      for: a from: all do: [(Smalltalk:a) noChanges]].
   ]
printAll | cat
   [for: cat from: commentVector do:
      [self printCategory: cat]]
printCategory: cat | f  "in alphabetical order"
   [user displayoffwhile:
      [f ← PressPrinter init of: (dpo file: (cat+'.press.') asFileName).
      f printclass: (self category: cat).
      f close]]
superclassOrder: cat | all lis a i c sup  "Arrange classes in superclass order so
they can be filed in"
   [lis← (self category: cat) copy. all ← (Vector new: lis length) asStream.
   while: lis length>0 do:
      [i← 1.
      until:
         [a← lis:i. sup← c← Smalltalk:a.
         until: "Make sure it doesn't have an as yet unprinted superclass"
            [sup← sup superclass.
            sup=nil⇒[true]
            lis has: sup title unique]
         do: [].
         sup=nil]
      do: [i← i+1].
      all next ← a.
      lis← lis delete: a].
   all contents]
↵
SystemOrganization classify: ↷SystemOrganizer under: 'Sets and
Dictionaries'.↵
```

"Cursor"

```
Class new title: 'Cursor'
   subclassof: Object
   fields: 'bitstr offset'
   declare: '';
   asFollows_!
```

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

Initialization

```
fromString: bitstr [self fromString: bitstr offset: 0⊙0]
fromString: bitstr offset: offset
fromtext: str [self fromtext: str offset: 0⊙0]
fromtext: str offset: offset | i s n c [
   "Not great, but compatible with printon."
   bitstr ← String new: 32.
   s ← str asStream.  s next.
   for: i to: 16 do:
      [n ← 0.
      while: ((c ← s next)=060 or: c=061) do:
         [n ← (n lshift: 1)+(c-060)].
      bitstr word: i ← n]]
offset: offset
```

Hardware cursor

```
frompage1        "load this cursor from the hardware locations"
   [bitstr ← String new: 32.
   BitBlt new forCursor; sourcebase← 0431; destbase ← bitstr; copy: storing]
```

Conversion

```
printon: strm | i
   [strm append: 'Cursor new fromtext: '''.
   for: i to: 16 do:
      [strm cr.
      (bitstr word: i) printon: strm base: 2]
   strm append: ''' offset: '; print: offset; append: '.']
```

Aspects

```
offset [↑offset]
```

Showing

```
show     [
   "copy this cursor into the page 1 hardware locations"
   BitBlt new forCursor; destbase← 0431; sourcebase ← bitstr; copy: storing.
   user currentCursor: self

   "the following statement will copy back if we ever need to"
   "BitBlt new forCursor; sourcebase← 0431; destbase ← bitstr; copy: storing"]
showwhile: expr | oldcursor value [
```

```
    oldcursor ← user currentCursor.
    self show.
    value ← expr eval.
    oldcursor show.
    ↑value]
```

Compatibility

topage1 [self show]

SystemOrganization classify: ↗Cursor under: 'Graphical Objects'.

"Form"

```
Class new title: 'Form'
   subclassof: Object
   fields: 'extent bits offset'
   declare: 'white reverse over black under ';
   asFollows_!
```

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

INIT

```
classinit
   ["sets up colors and effectos for BITBLT."
   black ← 0-1.
   white ← 0.
   over ← 0.
   under ← 1.
   reverse ← 2.
   ]
extent: extent |
   ["creates a virtual bit map with width = extent x and height = extent y."
   bits ← String new: (( extent x)+15)/16 * 2 * extent y.
   offset ← (0⊙0).
   self white.
   ]

fromImage: image
   ["creates a virtual bit map with width = (image width) and height = (image
height) with the bits in image."
   offset ← 0⊙0.
   extent ← image extent.
   bits← (image rectangle) bitsIntoString
   ]

fromrectangle: r
   ["creates a virtual bit map with width = (r width) and height = (r height)
with the bits in r."
   offset ← 0⊙0.
   extent ← r extent.
   bits← r bitsIntoString
   ]

   fromuser | r
   ["create a new Form whose rectangle is specified by the user. "
   r ← Rectangle new fromuser.
   offset ← 0⊙0.
   extent ← r extent.
   bits← r bitsIntoString
   ]
```

PATTERN ACCESS

```
bits
   ["return the string containing the bits)"
   ↑ bits
   ]
black | i
   ["sets all bits in the form to black ( to ones)"
   for: i to: bits length do: [ bits↓i ← 0-1]
   ]
gray | i
   ["sets all bits in the form to gray ( to gray)"
   for: i to: bits length do: [ bits↓i ← 025252]
   ]
white | i
   ["sets all bits in the form to white ( to zeros)"
   for: i to: bits length do: [ bits↓i ← 0]
   ]
```

MODULE ACCESS

```
extent
   ["return the extent (width⊙height) of the Form"
   ↑ extent
   ]
height
   ["return the height of the Form"
   ↑ extent y
   ]
offset
   ["return the offset of the Form"
   ↑ offset
   ]
offset: offset
   ["set the offset of the form "
   ↑ self
   ]
width
   ["return the width of the Form"
   ↑ extent x
   ]
```

FILING

```
read: filename | f strip w h form stripheight leftoverlines i
   ["Reads the Form from the disk in the format width,height,bits."
   f← (dp0 file: filename).
   w ← (f nextword).
   h ←(f nextword).
   extent ← w⊙h.
   w*h < 16000⇒
      [bits ← (Form new extent: extent) bits.
      f into: bits.
      f close.
      ]
```

```
    stripheight ← 50.
    leftoverlines ← h \ stripheight.
    form ← (Form new extent: w@stripheight) .
    strip ← form bits.
    for: i from: ( 1 to: h-stripheight by: stripheight) do:
       [   f into: strip.
          form copyto: (0@i) effect: over.
       ]
    f close.

    ]
write: filename | f
    ["Saves the Form in the format width,height,bits."
    f← (dpO file: filename).
    f nextword← (self width).
    f nextword← (self height).
    f append: bits.
    f shorten ; close.
    ]
```

DISPLAY

```
displayat:  path effect: effect clippedBy: cliprect| r i
    ["basic form display primitive"

    path is: Point ⇒ [ r ← Rectangle new origin:  path extent: (self extent).
                          r bitsFromString: bits mode: effect clippedBy: cliprect. ]
    path is: Path ⇒ [for: i to: path length do: [ self displayat: path@i
                                  effect: effect clippedBy: cliprect] ]
    ]
```

SystemOrganization classify: ↱Form under: 'Graphical Objects'.

"FormSet"

```
Class new title: 'FormSet'
   subclassof: Object
   fields: 'space spaceorigin image strike style styleindex formindex
offsettable'
   declare: 'imageindex bitmover ';
   asFollows_!
```

An object for holding sets of forms. The most conventional use will be as a repository of a set of forms typically identified as characters when seen on the display or on paper. The forms will be ascessible by passing a one-character string or a number.

ACCESS

```
asForm: formindex        | f
   ["returns the form indexed by formindex ."
   self checkindex.
   f ← Form new extent: (self width) ⊙ (self height) .
   bitmover destraster ← (f width + 15 / 16).
   bitmover destx ← 0. bitmover desty ← 0.
   bitmover sourcex ← (self originx). bitmover sourcey ← 0.
   bitmover width ← (self width).
   bitmover height ← (self ascent) + (self descent).
   bitmover sourceraster ← (self wordwidth).
   bitmover destbase ← (f bits).
   bitmover sourcebase ← strike.
   bitmover dstrike ← false.
   bitmover sstrike ← true.
   bitmover copy: storing.
   ↑ f
   ]
changeascentto: newascent
   [   "new ascent for FormSet"
   self deltaascent: (newascent - (self ascent))
   ]
changedescentto: newdescent
   [   "new ascent for FormSet"
   self deltadescent: (newdescent - (self descent))
   ]
changewidthof: formindex to: width
   [   "new width for form at index"
   self checkindex. self deltawidthof: formindex by: (width - (self width))
   ]
classInit
   ["Just initialize the bitmover for now."
   bitmover ← BitBlt init
   ]
copy: formindex      to: pt | f
   ["copies the form indexed by formindex to pt."
   self checkindex.
   "f ← Image new size: (self width) ⊙ (self height) at: pt."
```

```
    bitmover destraster ← ((user screenrect) width + 15 / 16).
    bitmover destx ← pt x. bitmover desty ← pt y.
    bitmover sourcex ← (self originx). bitmover sourcey ← 0.
    bitmover width ← (self width).
    bitmover height ← (self ascent) + (self descent).
    bitmover sourceraster ← (self wordwidth).
    bitmover destbase ← (memo 066).
    bitmover sourcebase ← strike.
    bitmover strike ← true.
    bitmover copy: oring.
    ↑ self widthof: formindex
    ]
copy: formindex      to: pt effect: effect | f
    ["copies the form indexed by formindex to pt."
    self checkindex.
    "f ← Image new size: (self width) @ (self height) at: pt."
    bitmover destraster ← ((user screenrect) width + 15 / 16).
    bitmover destx ← pt x. bitmover desty ← pt y.
    bitmover sourcex ← (self originx). bitmover sourcey ← 0.
    bitmover width ← (self width).
    bitmover height ← (self ascent) + (self descent).
    bitmover sourceraster ← (self wordwidth).
    bitmover destbase ← (memo 066).
    bitmover sourcebase ← strike.
    bitmover strike ← true.
    bitmover copy: effect.

    ]
copyrange: start to: stop from: sourceset startingat: deststart
        | savebackground savebits i f
    ["copy a range of forms from one set to another"
    user displayoffwhile:
    [
    [sourceset is: FormSet ⇒ []
        sourceset is: String ⇒ [sourceset ← FormSet new from: sourceset]
        user notify: 'Illegal sourceset -- not String or Formset.'
    ].
    savebackground ←
        Form new size: (sourceset maxwidth) by: (sourceset height).
    savebackground translate: 0@0; scale: 1.
    savebits ← savebackground bitsIntoString.

    for: i from: start to: stop do:
        [f ← sourceset copy: i to: 0@0. self include: deststart with: f.
        deststart ← deststart + 1].

    savebackground bitsFromString: savebits.
    ]
    ]
currentformindex
    ["return index of form last touched"
    ↑ formindex
```

```
    ]
currentformorigin | imageindex
    ["return index in image of form last touched"
     imageindex ← image find: formindex.
    ↑ space ptofchar: imageindex.
    ]
from: strike
    ["Make a formset out of string in strike format"
    self classInit. self install: strike
    ]
from: first to: last ascent: ascent descent: descent
    style: style styleindex: styleindex name: name
    ["Make an empty formset."
    self classInit.
    offsettable ← String new: (last - first + 3) * 2.
    offsettable all ← 0.
    offsettable word: (last - first + 3) ← 4.          "width of illegal form"
    strike ←
        String new: (9 "header" + ascent + descent "space for illegal form") * 2.
    strike all ← 0.
    self type: 0100000.                    "for the outside world"
    self first: first.
    self last: last.
    self wordwidth: 1.          "only illegal form"
    self ascent: ascent.
    self descent: descent.
    self maxwidth: 4.           "width of illegal form"

    strike ← strike concat: offsettable o (1 to: offsettable length).

    "mash in bits of illegal form"
    "leftside"
    bitmover destraster ← (self wordwidth).
    bitmover destx ← 0. bitmover desty ← 0.
    bitmover width ← 1. bitmover height ← (self ascent) + (self descent).
    bitmover destbase ← strike.
    bitmover strike ← true.
    bitmover fill: storing color: black.
    "rightside"
    bitmover destx ← 3.
    bitmover strike ← true.
    bitmover fill: storing color: black.
    "top"
    bitmover width ← 4. bitmover height ← 1. bitmover destx ← 0.
    bitmover strike ← true.
    bitmover fill: storing color: black.
    "bottom"
    bitmover desty ← (self ascent) + (self descent) - 1.
    bitmover strike ← true.
    bitmover fill: storing color: black.

    self install: strike.
```

```
    self updateseglength.
    [style ≡ nil ⇒ []
        style setfont: styleindex name: name fromstring: strike]
    ]
fromspace: pt to: dest
    ["get form selected from space"
    formindex ← image ∘ ((space charofpt: pt) min: 256).
    ↑ (self copy: formindex to: dest).
    ]
fromstyle: style styleindex: styleindex
    ["Make a formset out of string in strike format"
    self classInit. self install: style fonts∘(styleindex+1)
    ]
height
    ["return height of fromset"
    ↑self ascent + self descent
    ]
include: formindex with: form | newoffsettable newstrike i j
    ["Put a form into the formset"
    (formindex > (self first)) and:
        (formindex < (self last)) ⇒ [ self replace: formindex with: form]

    formindex < 0 ⇒ [user notify: 'Formindex < 0 illegal for formset.']
    formindex > 255 ⇒ [user notify: 'Formindex > 255 illegal for formset.']

    [formindex < (self first) ⇒
        [newoffsettable ←
            String new: ((self first) - formindex + (self abslength)) * 2.
        newoffsettable all ← 0.
        j ← (self first) - formindex + 1.
        for: i from: j to: ((newoffsettable length) / 2) do:
            [newoffsettable word: i ← strike word: offsettable + (i-j)].
        ]
        newoffsettable ←
            String new: ((self abslength) + formindex - (self last)) * 2.
        newoffsettable all ← 0.
        for: i from: 0 to: ((self length) - 1) do:
            [newoffsettable word: (i+1) ← strike word: offsettable + i].
        for: i from: (self length) to: ((newoffsettable length) / 2) do:
            [newoffsettable word: i ← strike word: offsettable + self length].
        newoffsettable word: ((newoffsettable length) / 2) ←
            strike word: (offsettable + self length + 1).
        ].
    newstrike ← String new: (9 "header" + ((self wordwidth) *
        ((self ascent + self descent)) "bits")) * 2.        "new space for bits"
    for: i to: 9 do:
        [newstrike word: i ← strike word: i].               "fill in header of new font"

    bitmover destraster ← (self wordwidth).
    bitmover destx ← 0. bitmover desty ← 0.
    bitmover sourcex ← 0. bitmover sourcey ← 0.
    bitmover width ← (self strikerightx).
```

```
    bitmover height ← (self ascent) + (self descent).
    bitmover sourceraster ← (self wordwidth).
    bitmover destbase ← newstrike.
    bitmover sourcebase ← strike.
    bitmover strike ← true.
    bitmover copy: storing.

    "copy the xtable"
    newstrike ←
        newstrike concat: newoffsettable o(1 to: newoffsettable length).
    self install: newstrike.
    [formindex < (self first) ⇒ [self first: formindex] self last: formindex].
    self replace: formindex with: form.
    ]
initspaceat: spaceorigin | i run para
    ["make a space for formset viewing"
    image ← String new: 256.
    image all ← 0.
    for: i from: ((self first) to: (self last)+1)
        do: [image o(i+1) ← i].
    run ← String new: 2.
    run word: 1 ← 16 * (styleindex) + 0177400.
    para ←
    Paragraph new text: image runs: run alignment: 0.
    [space ≡ nil ⇒ [] space erase].
    space← Textframe new para: para
        frame: (Rectangle new origin: spaceorigin extent:
        ((self last) - (self first) * (self maxwidth) / 8)
                                    ⊙
        ((self ascent) + (self descent) * 6))
        style: style.
    ]
makecu: name scale: scale
        | f i iform bits drast srast
    ["Put out strike in Carnegie-Mellon format.
        A typical call might look like:
            yourset ←
                FormSet new style: DefaultTextStyle styleindex: 0.

                yourset makecu: 'cream12' scale: 1.
            "

    user displayoffwhile:
    [
    f ← (dp0 file: name + '.cu.').
    f nextword ← self height * scale.
    f nextword ← self maxwidth * scale + 15 / 16.
    bits ← String new:
        ((self height * scale) * ((self maxwidth * scale + 15)/16))
            * 2.
    drast ← self maxwidth * scale + 15 / 16.
    srast ← (user screenrect width) + 15/16.
```

```
    for: i from: ((self first) to: (self last) by: 1) do:
        [iform ← self copy: i to: 0⊙0.
        [scale > 1 ⇒ [iform blowup: 0⊙0 by: scale]].
        f nextword ← i. f nextword ← self width*scale.

        bitmover destbase ← bits.
        bitmover destraster ← drast.
        bitmover destx ← 0. bitmover desty ← 0.
        bitmover sourcebase ← mem∘066.
        bitmover sourceraster ← srast.
        bitmover sourcex ← 0. bitmover sourcey ← 0.
        bitmover strike ← false.
        bitmover width ← (iform width) * scale.
        bitmover height ← (iform height) * scale.
        bits all ← 0.
        bitmover copy: storing.
        f append: bits].
    f shorten. f close.
    ]
    ]
newspace
    ["let user reshape/position space"
    space frame ← Rectangle new fromuser. self show.
    ]
replace: formindex with: form
    ["Replace form in set.  Check incoming form for compatibility with formset,
    and insert form into formset."
    self checkindex.
    [form width ≠ self width ⇒ [self changewidthof: formindex to: form width]].

    "copy bits of form into formset"
    bitmover destraster ← (self wordwidth).
    bitmover destx ← (self originx). bitmover desty ← 0.
    bitmover sourcex ← form origin x. bitmover sourcey ← form origin y.
    bitmover width ← (self width).
    bitmover height ← (self ascent) + (self descent).
    bitmover sourceraster ← ((user screenrect) width + 15 / 16).
    bitmover destbase ← strike.
    bitmover sourcebase ← (mem ∘ 066).
    bitmover strike ← true.
    bitmover copy: storing.
    ]
show
    ["show all the forms in the set"
    space outline. space show
    ]
space
    ["return textframe that is space of formset"
    ↑ space
    ]
spaceframe
    ["return frame of space"
```

```
    ↑(space frame)
    ]
spaceorigin: spaceorigin
    ["reposition the space"
    space erase. (space frame) origin ← spaceorigin. self show.
    ]
widthof: formindex
    ["return width of from at formindex"
    ↑self width
    ]

INTERNAL
abslength
    ["Return absolute length of formset, i.e. number of forms in set
        plus space for illegal character and its rightx"
    ↑ ((self last) - (self first) + 3)
    ]
checkindex
    ["check formindex for legality and make into number if necessary"
    [formindex is: String ⇒
        [formindex length > 1 ⇒ [user notify: 'formindex out of range for FormSet.']
        formindex ← formindex∘1]
    ].
    (formindex < (self first)) or: (formindex > (self last+1)) ⇒
        [user notify: 'formindex out of range for this FormSet.']
        ]
deltaascent: delta | newstrike
    ["ascent delta"
    [(self ascent) + delta < 0 ⇒[delta ← 0 - (self ascent)]].
    [delta > 0 ⇒
        ["grow"
        newstrike ← String new: (2 * (self wordwidth) * delta).
        newstrike all ← 0.                          "fill with white"
        newstrike ←             "add oldfont header and new space together"
            (strike∘(1 to: 18) concat: newstrike∘(1 to: newstrike length)).
        newstrike ←                                 "now add on rest of old font"
            (newstrike concat: strike∘(19 to: strike length)).
        ]
    "shrink"
    newstrike ← (strike∘(1 to: 18) concat:
        strike∘((19 + (0 - (2 * (self wordwidth) * delta)))
            to: strike length)).
    ].
    newstrike word: 6 ← ((self ascent) + delta).        "reset ascent word in font"
    self install: newstrike.                        "newstrike now font of interest"
    self updateseglength.
    [style ≡ nil ⇒ [ ]
        (style maxascent) < (self ascent) ⇒ [style maxascent: (self ascent)]].
    ]
deltadescent: delta | newstrike somespace
    ["descent delta"
    [(self descent) + delta < 0 ⇒ [ delta ← 0 - (self descent)]].
```

```
    [delta > 0 ⇒
        [somespace ← String new: 2 * (self wordwidth) * delta.
        somespace all ← 0.
        newstrike ←
          (strike o (1 to: offsettable - 1 * 2) concat: somespace).
        ]
    newstrike ←
      (strike o (1 to: ((offsettable - 1 * 2)
          + ((self wordwidth) * delta * 2)))).
    ].
    "copy the xtable"
    newstrike ←
      newstrike concat: strike o ((offsettable * 2 - 1) to: strike length).
    newstrike word: 7 ←
      ((self descent) + delta).          "reset descent word in font"
    self install: newstrike.              "updatedfont now font of interest"
    self updatemaxwidth.
    self updateseglength.
    [style ≡ nil ⇒ [ ]
      (style maxdescent) < (self descent) ⇒
        [style maxdescent: (self descent)]]
    ]
deltawidthof: index by: delta
    | newwordwidth newoffsettable newstrike normalizedindex normalizedlast i
    ["change width of form at index"
    [delta < 0 ⇒ [(delta abs) > (self width) ⇒ [delta ← 0-(self width)]]].
    newwordwidth ←
      [((self strikerightx) + 15 / 16) ≠
        (i ← ((self strikerightx) + delta + 15 / 16)) ⇒
        [ i ]
        (self wordwidth)].
    newoffsettable ← newwordwidth *
      ((self ascent + self descent)) "height" + 9 "header" + 1 "for 0 addressing".
    XeqCursor showwhile:
    [
    newstrike ← String new: (9 "header" + (newwordwidth *
      ((self ascent + self descent)) "bits")) * 2.          "grow/shrink the bits"
    newstrike all ← 0.
    for: i to: 8 do:
      [newstrike word: i ← strike word: i].                 "fill in header of new
font"
    newstrike word: 9 ← newwordwidth.                       "set raster in new font"

    "copy the xtable"
    newstrike ← newstrike concat: strike o ((offsettable * 2 - 1) to: strike length).

    "set up to copy up to old bits of form in formset"
    bitmover destraster ← newwordwidth.
    bitmover destx ← 0. bitmover desty ← 0.
    bitmover sourcex ← 0. bitmover sourcey ← 0.
    bitmover width ← (self originx) + (self width).
    bitmover height ← (self ascent) + (self descent).
```

```
    bitmover sourceraster ← (self wordwidth).
    bitmover destbase ← newstrike.
    bitmover sourcebase ← strike.
    bitmover strike ← true.
    bitmover copy: storing.

    "now copy remainder of font"
    bitmover destx ← (self originx) + (self width) + delta.
    bitmover width ← (self strikerightx) - (self originx) - (self width).
    bitmover sourcex ← (self originx) + (self width).
    bitmover strike ← true.
    bitmover copy: storing.

    "shift x-vals"
    normalizedindex ← formindex - (self first).
    normalizedlast ← (self last) - (self first).
    for: i from: ((normalizedindex + 1) to: (normalizedlast + 2 "max")) do:
        [newstrike word: (newoffsettable+i) ←
            delta + (newstrike word: (newoffsettable+i))].
    self install: newstrike.        "set up the updated copy of the formset"
    self updatemaxwidth.
    self updateseglength.
    ].
    ]
install: strike
    ["set up a new or refreshed strike"
    [style ≡ nil ⊃ [] style fonts ○ (styleindex + 1) ← strike].
    offsettable ← (self wordwidth) *
        ((self ascent) + (self descent)) + 9 "header" + 1 "for 0 addressing".
    ↑ strike]
length
    ["Return length of formset, i.e. number of forms in set"
    ↑ ((self last) - (self first) + 1)
    ]
originx
    ["Return origin x  of form at formindex"
    ↑ (strike word: (offsettable + formindex - (self first)))]
updatemaxwidth | newmaxwidth i
    ["update max width"
    newmaxwidth ← 0.
    for: i from:
        (offsettable to: offsettable + ((self last) - (self first) + 1))
        do:
        [newmaxwidth ←
            (newmaxwidth
                max: ((strike word: i+1) - (strike word: i)))].
    self maxwidth: newmaxwidth.
    ]
updateseglength
    ["compute new segment length for formset"
    strike word: 5 ←
        (5                                  "length, ascent, descent, kern, and
```

```
raster"
        + ((self wordwidth) * ((self ascent) + (self descent)))        "bits"
        + ((self last "max") - (self first      "min") + 2)             "xtabl"
    ).
  ]
width
  ["Return width of form at formindex"
  ↑ (strike word: (offsettable + (formindex - (self first)) + 1)) -
      (strike word: (offsettable + (formindex - (self first)))) ]

PARTS
ascent
  ["When form set treated as characters, describes distance from top of form
  to baseline."
  ↑(strike word: 6)
  ]
ascent: ascent
  ["When form set treated as characters, describes distance from top of form
  to baseline."
  strike word: 6 ← ascent.
  ]
descent
  ["When form set treated as characters, describes distance from bottom of
  form to baseline."
  ↑(strike word: 7)
  ]
descent: descent
  ["When form set treated as characters, describes distance from bottom of
  form to baseline."
  strike word: 7 ← descent.
  ]
first
  ["Heritage from the world of fonts"
  ↑(strike word: 2)     "minimum formindex (ascii) in the strike"]
first: first
  ["Heritage from the world of fonts"
  strike word: 2 ← first.        "minimum formindex (ascii) in the strike"
  ]
kern
  ["When form set treated as characters, describes distance this form is
  to intrude into space of preceding character."
  ↑(strike word: 8)
  ]
last
  ["Heritage from the world of fonts"
  ↑(strike word: 3)     "maximum formindex (ascii) in the strike"
  ]
last: last
  ["Heritage from the world of fonts"
  strike word: 3 ← last.        "maximum formindex (ascii) in the strike"
  ]
maxwidth
```

```
    ["All forms in this set ≤ to this value"
    ↑(strike word: 4)
    ]
maxwidth: maxwidth
    ["All forms in this set ≤ to this value"
    strike word: 4 ← maxwidth.
    ]
segmentlength
    ["Amount of space allocated for form set - 4"
    ↑(strike word: 5)
    ]
strikerightx
    ["Corner x of last form in form set"
    ↑(strike word: (offsettable + ((self last) - (self first)) + 2)).
    ]
type
    ["**BEWARE -- outside world has ideas about this value."
    ↑(strike word: 1)
    ]
type: type
    ["**BEWARE -- outside world has ideas about this value."
    strike word: 1 ← type.
    ]
wordwidth
    ["Also know as the raster of the formset.
    The width in alto words of the bits of the formset. When the display of a
    form is desired, the word and bit address of the bits of the form must
    be discovered. Adding the wordwidth to the word portion of that value,
    produces the word address of the second line of the bits of the form, and
    so on until the height of the form is spanned."

    ↑(strike word: 9)
    ]
wordwidth: wordwidth
    ["Also know as the raster of the formset.
    The width in alto words of the bits of the formset. When the display of a
    form is desired, the word and bit address of the bits of the form must
    be discovered. Adding the wordwidth to the word portion of that value,
    produces the word address of the second line of the bits of the form, and
    so on until the height of the form is spanned."

    strike word: 9 ← wordwidth.
    ]
⌟
SystemOrganization classify: ↗FormSet under: 'Graphical Objects'.⌟
FormSet classInit⌟
```

"Image"

```
Class new title: 'Image'
   subclassof: Set
   fields: ' origin rectangle path form'
   declare: 'under screen reverse over black white ';
   asFollows_!
```

This class has not yet been commented

```
INIT
at: origin
   ["create a new Image at origin with size (1@1). "
   self at: origin size: (1@1)
   ]
at: origin size: size
   ["create a new Image at origin with size (width@height). "
   rectangle ← Rectangle new origin: origin extent: size.
   self default.
   ]
classInit
   ["sets up black and white as colors and over ,under and reverse as modes
   also initializes the name screen as an image the size of the display"
   black ← 0-1.
   white ← 0.
   over ← 0.
   under ← 1.
   reverse ← 2.
   screen ← Image new origin: user screenrect origin extent: user screenrect
extent.
   ]
fromuser
   ["create a new Image whose rectangle is specified by the user. "
   rectangle ← Rectangle new fromuser.
   form ← Form new fromrectangle: rectangle.
   origin ← path ←rectangle origin.
   self default.
   ]
origin: origin extent: extent
   ["create a new Image at origin with extent (width@height). "
   rectangle ← Rectangle new origin: origin extent: extent.
   self default.
   ]
origin: origin rectangle: rectangle path: path form: form
   ["basic message to create a new instance."
   self default]
size: size at: origin
   ["create a new Image at origin with size (width@height). "
   self at: origin size: size
   ]
```

BUILDING IMAGES

```
add: p and: i | s
   ["add p (set or point) and i ( Image or Form ) and expand the
      bounding rectangle of this image."
   rectangle ←
      rectangle include:
         ((Rectangle new origin: (p origin)+ origin extent: i size)
            include: (Rectangle new origin: (p corner)+ origin extent: i size)).
   s ← Set default. s add: p ; add: i .
   self add: s
   ]
addform: f andpath: p | r
   ["add p (set or point) and f ( Form ) and expand the
      bounding rectangle of this image."
   rectangle ←
      rectangle
         include:
         (r ← ((Rectangle new origin: (p origin)+ origin extent: f extent)
         include:
         (Rectangle new origin: ((p corner) - (1⊙1))+ origin extent: f extent))).
   self add: Image new origin: p origin rectangle: r path: p form: f.
   ]
addpath: p andform: f | r
   ["add p (set or point) and f ( Form ) and expand the
      bounding rectangle of this image."
   rectangle ←
      rectangle
         include:
         (r ← ((Rectangle new origin: (p origin)+ origin extent: f extent)
         include:
         (Rectangle new origin: ((p corner) - (1⊙1))+ origin extent: f extent))).
   self add: Image new origin: p origin rectangle: r path: p form: f.
   ]
```

CHANGING IMAGES

```
comment
   ["see class Set for operations on subimages ( elements)."]
deletesubimage: i
   ["delete the ith subimage."
   self deleteindex: i.
   ]
indexofsubimageat: pt | i subimage pos
   ["return the index of the subimage which contains pt(relative to self origin)
      otherwise return false."
   for: i to: position do:
      [pos ← (self∘i)∘1.
      subimage ← (self∘i)∘2.
      (Rectangle new origin: pos extent: subimage extent) has: pt⇒ [ ↑ i]
      ]
   ↑ false
   ]
subimage: i | sub s
```

```
   ["return the ith subimage."
   sub ← (self↓i)
   s ← Image new at: (self origin) + (sub↓1) origin. s add: (0@0) and: sub↓2.
   ↑ s
   ]
subimageat: pt | i
   ["return the  subimage which contains pt (relative to self origin)
      otherwise return false."
   i ← self indexofsubimageat: pt.
   i ⊃ [ ↑ self↓ i]
   ↑ false
   ]
substitute: form1 for: form2 | i
   ["everywhere in the imagesubstitute form1 for form2"
   for: i to: self length do: [ (self↓i)↓2 ≡ form2 ⊃  [ (self↓i)↓2 ← form1]]
   ]
```

DISPLAY

```
copyto: pt effect: effect | i vector
   ["display self "
   self displayat: pt effect: effect .
   ]
cycle | i
   ["display and then erase all of the forms over their paths in the image "
   for: i to: self length  do: [ (self↓i)↓2 copyto:  ((self↓i)↓1) +  origin effect: 1
.(self↓i)↓2 copyto:  ((self↓i)↓1) + origin effect: 3]
   ]
display
   ["display all of the forms in the image on the screen "
   self displayat: 0@0 effect: 0 clippedBy: user screenrect.
   ]
displayat: pt effect: effect | i
   ["display all of the forms over their paths in the image "
   for: i to: self length  do: [ (self↓i)↓2 copyto:  ((self↓i)↓1) + (pt + origin)
effect: effect]
   ]
displayat: pt effect: effect clippedBy: cliprect | i
   ["display all of the subimages in this image "
   self length = 0⊃ [ form displayat: (pt + origin) effect: effect clippedBy:
cliprect]
   for: i to: self length  do:
            [ (self↓i) displayat: (pt + origin) effect: effect clippedBy: cliprect
            ]
   ]
displayeffect: effect | i element
   ["display all of the forms in the image "
   self displayat: 0@0 effect: effect
   ]
displaysubimage: n effect: effect | i  form pt
   ["display nth subimage in the image "

   form ← (self↓n)↓2. pt ← (self↓n)↓1.
```

```
    form copyto: pt+ origin effect: effect
    ]
erasesubimage: n  | i vector form pt
    ["erase nth subimage in the image "
    self displaysubimage: n effect: 3
    ]
```

MODULE ACCESS

```
contains: pt
    ["return true if the  bounding rectangle for the Image contains pt.."
    ↑ rectangle has: pt
    ]
extent
    ["return the extent (width,height) of the Image."
    ↑ rectangle extent
    ]
height
    ["return the height of the Image."
    ↑ rectangle extent y.
    ]
origin
    ["return the origin of the image."
    ↑ origin
    ]
origin: origin
    ["change the origin of the image."
    ]
rectangle
    ["return the rectangle that bounds the Image."
    ↑ rectangle
    ]
rectangle: r
    ["redefine rectangle that bounds the Image."
    rectangle← r
    ]
size
    ["return the extent (width,height) of the Image."
    ↑ rectangle extent
    ]
size: size
    ["change the size (width,height) of the Image."
    rectangle extent← size
    ]
width
    ["return the width of the Image."
    ↑ rectangle extent x.
    ]
```

PATTERN ACCESS

```
black
    ["black out the image"
```

```
    rectangle clear: black.
    ]
boxcomp
    ["border without disturbing the interior."
    rectangle boxcomp
    ]
gray
    ["gray out the image"
    rectangle clear: gray.
    ]
reverse
    ["reverse  the image (black to white and white to black)"
    rectangle comp.
    ]
white
    ["white out the image"
    rectangle clear: 0.
    ]
```

TRANSFORMATIONS

```
translate: delta | element
    ["translate the bounding rectangle of the Image and all its subimages."
    rectangle translate: delta.
    origin translate: delta
    ]
translateto: pt | delta
    ["move the rectangle and hence the Image to pt."
    delta ← pt - origin.
    self translate: delta
    ]
```

SYSTEM

```
printon: strm | t [
    strm append: 'an Image: '.
    array is: String⇒ [strm space append: self]
    for: t from: self do: [strm space print: t]]
⌟
SystemOrganization classify: ↱Image under: 'Graphical Objects'.⌟
Image classInit⌟
```

"Path"

```
Class new title: 'Path'
   subclassof: Set
   fields: ''
   declare: '';
   asFollows_!
```

This class has not yet been commented

```
INIT
init
   ["must be executed for each new instance."
   self default.
   ]

BUILDING PATHS
comment
   ["see Set for these ... add:, append:, and o← are the main ones"
   ]

ACCESSING PATHS
pointnearestto: p | distance i nearest d
   ["return the index of the point  in the path nearest (manhatten norm) to p."
   distance ← p dist: self↓1.
   nearest ← 1.
   for: i to: position do:
      [   d ← p dist: self↓i .
         d< distance⇒
         [ nearest← i. distance ← d.]
      ]
   ↑ nearest
   ]

MODIFYING PATHS
deleteindex: i | r [
   r ← array↓( i+1 to: position).
   position ← i-1.
   self append: r.
   array↓(position+1) ← nil.
   ]
insert: pt atindex: index | r
   [
   "insert pt at index in the path"
   index > position⇒ [self next ← pt]
   r ← [position = limit⇒ [self grow] self growby: 0].
   position ← 0.
   self append: (r ↓ 1 to: index-1);
      next ← pt;
      append: r↓(index to: r length).
   ]
```

SPECIAL PATHS

```
addarcfrom: p1 via: p2 to: p3 | pa pb i k s
   ["Kaehler method for Flegal curve"
   s← Path new init.
   s add: p1.
   pa← p2-p1. pb← p3-p2.
   k← 5 max: (pa x abs + pa y abs + pb x abs + pb y abs)/20.
   for: i to: k do:        "k is a guess how many segments are appropriate"
      [s add: (pa*i/k+p1*(k-i)) + (pb*(i-1)/k+p2*(i-1)) / (k-1)]
   s add: p3 .
   for: i to: (s length-1) do: [ self addlinefrom: s∘i to: s∘(i+1) ]
]
addlinefrom: p1 to: p2 | x1 y1 dx dy yinc x0 y0 cdl i
   ["for now just add points to the space at alto resolution between p1 and p2
   inclusive"
   dx← ( p2 x) - (p1 x).
   dy← ( p2 y) - (p1 y).

   [dx < 0 ⇒ [dx ← 0-dx. dy ← 0-dy. x0← p2 x. y0 ← p2 y]
                  x0 ← p1 x. y0 ← p1 y].
   [dy ≥ 0 ⇒ [yinc←1] yinc ← 0-1 . dy ← 0-dy].

    dx≥dy⇒
      [cdl ← ( dx/2).
      for: i from: 0 to: dx do:
         [self add: (x0⊙y0). cdl ← cdl + dy.
         x0 ← x0+1.
          cdl > dx⇒ [cdl ← cdl - dx. y0 ← y0 + yinc]
         ]
      ]

   "y is fastest mover"
   cdl ← (dy/2).
   for: i from: 0 to: dy do:
      [self add: (x0⊙y0) . cdl ← cdl+ dx.
      y0 ← y0 + yinc.
    cdl > dy ⇒ [cdl ← cdl - dy. x0← x0+1]
      ].
   ]
```

TRANSFORMATIONS

```
+ delta | i
   [ "add delta to every point in the path"
   ↑ (self copy) translate: delta
   ]
scale: factor | i
   [ "scale every point in the path by factor"
   for: i to: self length  do:
      [ self∘i ← self∘i * factor
      ]
```

```
    ]
translate: delta | i
    [ "add delta to every point in the path"
    for: i to: self length do:
        [ self@i ← self@i + delta
        ]
    ]
```

MEASURING

```
corner
    ["return the corner of the bounding rectangle that includes all the points in
the path."
    ↑ (self rectangle) corner
    ]
extent
    ["return the extent of the bounding rectangle that includes all the points in
the path."
    ↑ (self rectangle) extent]
height
    ["return the height of the bounding rectangle that includes all the points in
the path."
    ↑ self size y]
origin
    ["return the origin of the bounding rectangle that includes all the points in
the path."
    ↑ (self rectangle) origin
    ]
rectangle | r i
    ["return the bounding rectangle that includes all the points in the path."
    r ← Rectangle new origin: self@1 extent: 1@1.
    for: i to: self length do:
        [ r ← r include: self@i
        ]
    ↑ r
    ]
size
    ["return the extent of the bounding rectangle that includes all the points in
the path."
    ↑ (self rectangle) extent]
width
    ["return the width of the bounding rectangle that includes all the points in
the path."
    ↑ self size x]
```

SYSTEM

```
copy | t
    ["returns a new instance of Path that is a copy "
    t ← Path new init.
    t append: (array@(1 to: position)) copy.
    ↑ t]
printon: strm | t [
```

```
    strm append: 'a Path: '.
    array is: String⇒ [strm space append: self]
    for: t from: self do: [strm space print: t]]
↵
SystemOrganization classify: ↵Path under: 'Graphical Objects'.↵
```

"Point"

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

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

Initialization

```
x: x y: y
```

Arithmetic

```
≤ pt [↑ x≤pt x and: y≤pt y]
≥ 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)"
   ↑Point new x: (x * scale asPtX) y: (y * scale asPtY)]
+ delta
   ["Return a Point that is the sum of me and delta (which is a Point or
Number)"
   ↑Point new x: (x + delta asPtX) y: (y + delta asPtY)]
- delta
   ["Return a Point that is the difference of me and delta (which is a Point or
Number)"
   ↑Point new x: (x - delta asPtX) y: (y - delta asPtY)]
/ scale
   ["Return a Point that is the quotient of me and scale (which is a Point or
Number)"
   ↑Point new x: (x / scale asPtX) y: (y / scale asPtY)]
< pt [↑ x<pt x and: y<pt y]
= pt [↑x=pt x and: y=pt y]
> pt [↑x>pt x and: y>pt y]
abs    "absolute value of a point"
   [   ↑Point new x: (x abs) y: (y abs)      ]
dist: pt | t    "distance (Manhattan norm) between pt and self"
   [t ← (pt - self) abs.
   ↑(t x) + (t y)]
length
   [↑((x asFloat*x asFloat)+(y asFloat*y asFloat)) sqrt]
max: t
   [↑Point new x: (x max: t x) y: (y max: t y)]
min: t
   [↑Point new x: (x min: t x) y: (y min: t y)]
normal | n        "unit vector rotated 90 deg clockwise"
   [n← y asFloat neg ⊙ x asFloat.
   ↑n/n length]
translate: delta
   ["increment self by delta"
```

```
    self x ← self x + delta x.
    self y ← self y + delta y.
    ]
| grid
    [↑Point new x: x|grid y: y|grid]
```

Conversion

```
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 +-*/"
corner
    [↑ self+ (1@1)]
extent
    [↑ (1@1)]
height
    [↑ 1]
origin
    [↑ self]
printon: strm
    [strm print: x; append: '@'; print: y]
rect: p        "infix creation of rectangles"
    [↑Rectangle new origin: self corner: p]
width
    [↑ 1]
```

Access to parts

```
x     [↑x]
x ← x
y [↑y]
y ← y
⌟
SystemOrganization classify: ↶Point under: 'Graphical Objects'.⌟
```

"PressTurtle"

```
Class new title: 'PressTurtle'
   subclassof: Turtle
   fields: 'file fplace fdir filling'
   declare: '';
   asFollows_!
```

I work with Pressfile to print high resolution pictures.
All inputs can be floating point for high resolution.
Complexity is limited to about 2k lines until multiple entity lists

Initialization

```
close [file page. file close]
init: name
   [file ← (dp0 pressfile: name).
   filling← false.
   file pictureinit. self black.
   super init]
```

Pen Control

```
black [file brightness: 0]
blue [self color: 160]
color: h [file hue: h; brightness: 255; saturation: 255]
cyan [self color: 120]
green [self color: 80]
magenta [self color: 200]
place [↑fplace]
red [self color: 0]
up [dir← 270. fdir← 270.0]
white [file brightness: 255; saturation: 0]
yellow [self color: 40]
```

Drawing

```
filling expr        "Code in expr must describe a closed figure"
   [filling← true.
   file objects expr eval atScreen: fplace.
   filling← false]
go: dist | old
   [self goto: fplace +
      (([fdir\90.0=0.0⇒          "optimize horiz and vert lines"
         [fdir/90.0=0⇒[1.0⊙0.0];
                  =1⇒[0.0⊙1.0];
                  =2⇒[¯1.0⊙0.0];
                  =3⇒[0.0⊙¯1.0]]
      fdir asRadians asDirection])*dist)]
goto: p | old
   [old← fplace.
   fplace ← p x asFloat ⊙ p y asFloat.
   super goto: fplace x round ⊙ fplace y round.
   filling⇒[file objectGotoScreen: fplace pen: pen]
```

```
    pen=1⇒[file drawlinefromscreen: old to: fplace width: 0.46875*width]]
turn: angle [fdir← fdir+angle \360.0]
⌟
SystemOrganization classify: ↶PressTurtle under: 'Graphical Objects'.⌟
```

"Rectangle"

```
Class new title: 'Rectangle'
   subclassof: Object
   fields: 'origin corner'
   declare: '';
   asFollows↵
```

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

Initialization

```
fromuser | t
   ["Show the origin cursor until the user presses a mouse button,
      then get my origin"
   origin←OriginCursor showwhile: [user waitbug].
   "Show the corner cursor and complement me until the user presses
      a button again.  The loop is arranged so
      that complementing stays on for a little while."
   t←origin.
   CornerCursor showwhile:
      [while: [corner←t.  t ← user mpnext] do:
         [self comp.  t ← t max: origin.  self comp]]]
origin: origin corner: corner
origin: origin extent: extent
   [corner ← origin+extent]
```

Aspects

```
corner [↑corner]
corners | v
   [v← Vector new: 4.
   vo1← origin. vo2← corner x⊙ origin y.
   vo3← corner. vo4 ← origin x⊙corner y.
   ↑v]
corner ← corner
edge: side "Returns one side as a number."
   "Sides are numbered 0-3.  +1 goes counterclockwise.  lxor: 2 gets opposite
side."
   [side
      =0⇒["top" ↑origin y];
      =1⇒["left" ↑origin x];
      =2⇒["bottom" ↑corner y];
      =3⇒["right" ↑corner x].
   user notify: 'Invalid side']
extent
   [↑corner-origin]
extent ← extent
   [corner ← origin+extent. ↑extent]
height
   [↑corner y - origin y]
height ← h "change my bottom y to make my height h"
   [corner y ← origin y + h]
```

```
maxX [^corner x]
maxY [^corner y]
minX [^origin x]
minY [^origin y]
origin [^origin]
origin ← origin
side: side "Returns one side as a rectangle."
   "Sides are numbered 0-3.  +1 goes counterclockwise.  Xor: 2 gets opposite
side."
   [side
      =0⊃["top" ^origin rect: corner x⊙origin y];
      =1⊃["left" ^origin rect: origin x⊙corner y];
      =2⊃["bottom" ^origin x⊙corner y rect: corner];
      =3⊃["right" ^corner x⊙origin y rect: corner].
   user notify: 'Invalid side']
size
   [^corner-origin]
width
   [^corner x - origin x]
width ← w "change my right x to make my width w"
   [corner x ← origin x + w]
withEdge: side at: coord "Returns a rectangle with one side moved."
   [side
      =0⊃ [^origin x⊙coord rect: corner];
      =1⊃ [^coord⊙origin y rect: corner];
      =2⊃ [^origin rect: corner x⊙coord];
      =3⊃ [^origin rect: coord⊙corner y].
   user notify: 'Invalid side']
withSide: side at: pt "Returns a rectangle with one side moved."
   [side
      =0⊃ [^origin x⊙pt y rect: corner];
      =1⊃ [^pt x⊙origin y rect: corner];
      =2⊃ [^origin rect: corner x⊙pt y];
      =3⊃ [^origin rect: pt x⊙corner y].
   user notify: 'Invalid side']
```

Arithmetic

```
* scale [
"Return a Rectangle which is the product of me and scale (which is a Rectangle,
Point, or Number)"
   ^Rectangle new origin: origin * scale asRectOrigin corner: corner * scale
asRectCorner]
+ delta [
"Return a Rectangle which is the sum of me and delta (which is a Rectangle,
Point, or Number)"
   ^Rectangle new origin: origin + delta asRectOrigin corner: corner + delta
asRectCorner]
- delta [
"Return a Rectangle which is the difference of me and delta (which is a
Rectangle, Point, or Number)"
   ^Rectangle new origin: origin - delta asRectOrigin corner: corner - delta
asRectCorner]
```

```
/ scale [
"Return a Rectangle which is the quotient of me and scale (which is a
Rectangle, Point, or Number)"
   ↑Rectangle new origin: origin / scale asRectOrigin corner: corner / scale
asRectCorner]
= r
   [↑origin = r origin and: corner = r corner]
center
   [↑origin+corner/2]
empty
   [↑(origin < corner)≡false]
has: pt [↑origin ≤ pt and: pt < corner]
include: r "Returns the merge with an adjacent rectangle."
   [↑(origin min: r origin) rect: (corner max: r corner)]
inset: p1
   [↑origin+p1 rect: corner-p1]
inset: p1 and: p2
   [↑origin+p1 rect: corner-p2]
intersect: r [
   ↑Rectangle new origin: (origin max: r origin)
      corner: (corner min: r corner)]
intersects: r
   [↑(origin max: r origin) < (corner min: r corner)]
isWithin: rect  "am I equal to or contained within rect"
   [↑origin ≥ rect origin and: corner ≤ rect corner]
max: rect
   [↑Rectangle new
      origin: (origin min: rect origin)
      corner: (corner max: rect corner)]
minus: r | s yorg ycor          "return Vector of Rectangles comprising
            the part of me not intersecting r "
   ["Make sure the intersection is non-empty"
   [origin≤r corner and: r origin≤corner⇒ [] ↑self inVector].
   s ← (Vector new: 4) asStream.
   [r origin y>origin y⇒
      [s next ← origin rect: corner x⊙(yorg ← r origin y)]
    yorg ← origin y].
   [r corner y<corner y⇒
      [s next ← origin x⊙(ycor ← r corner y) rect: corner]
    ycor ← corner y].
   [r origin x>origin x⇒
      [s next ← origin x⊙yorg rect: r origin x⊙ycor]].
   [r corner x<corner x⇒
      [s next ← r corner x⊙yorg rect: corner x⊙ycor]].
   ↑s contents]
nearest: pt
   [↑((origin x max: pt x) min: corner x) ⊙
   ((origin y max: pt y) min: corner y)]
side: side distanceTo: pt
   [side
      =0⇒ [↑pt y-origin y];
      =1⇒ [↑pt x-origin x];
```

```
      =2⇒ [↑corner y-pt y];
      =3⇒ [↑corner x-pt x].
   user notify: 'Invalid side']
sideNearest: pt | d dmin i imin
   [dmin ← 077777.
   for: i from: (0 to: 3) do:
      [dmin>(d ← self side: i distanceTo: pt) abs⇒
         [dmin ← d.  imin ← i]].
   ↑imin]
```

Altering

```
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]
growby: pt
   [corner ← corner + pt]
growto: corner
moveby: pt
   [origin ← origin+pt. corner ← corner+pt]
moveto: pt
   [corner ← corner+pt-origin. origin←pt]
translate: pt
   [origin ← origin+pt. corner ← corner+pt]
translateto: pt
   [self translate:  pt - origin. ]
```

Conversion

```
asRectangle
   ["Return self."]
asRectCorner
   [↑corner ]
asRectOrigin
   [↑origin ]
bitsFromString: str
   ["default stores bits onto display"
   self bitsFromString: str mode: storing]
bitsFromString: str mode: mode [user croak] primitive: 52
bitsFromString: str mode: mode clippedBy: clipRect
      | destRect
   ["Load the screen bits within my area from those stored in str.
      If clipRect is not nil, then load only those bits within both
      myself and clipRect"
   self bitStringLength≠str length⇒[user notify: 'wrong bit string length']
   destRect←self intersect: user screenrect.
   [nil≡clipRect⇒[] destRect←destRect intersect: clipRect].
   BitBlt init destbase←mem∘066;
   destraster←user screenrect width/16|2;
   dest←destRect origin;
   extent←destRect extent;
```

```
    sourceraster←corner x-origin x+15/16;
    source←destRect origin-origin;
    sourcebase ← str ; copy: mode]
bitsIntoString | str [
    str ← String new: self bitStringLength.
    self  bitsIntoString: str mode: storing.
    ↑str]
bitsIntoString: str
    ["default stores bits into the string"
    self bitsIntoString: str mode: storing]
bitsIntoString: str mode: mode [user croak] primitive: 51
bitsIntoString: str mode: mode clippedBy: clipRect
        | sourceRect
    ["Store the screen bits within my area into str.  If clipRect is not nil,
        then store only those bits within both myself and clipRect,
        leaving alone the other bits in str"
    self bitStringLength≠str length⇒[user notify: 'wrong bit string length']
    sourceRect←self intersect: user screenrect.
    [clipRect≡nil⇒[] sourceRect←sourceRect intersect: clipRect].
    BitBlt init destraster←corner x-origin x+15/16;
    dest←sourceRect origin-origin;
    extent←sourceRect extent;
    sourcebase←mem∘066;
    sourceraster←user screenrect width/16|2;
    source←sourceRect origin;
    destbase ← str ; copy: mode]
bitsOntoStream: strm | rec s [
    rec ← origin rect: origin + (self width ⊙ (16 min: self height)).
    s← rec bitsIntoString  all←0.
    while: rec maxY ≤ corner y do:
        [rec bitsIntoString: s; moveby: 0⊙16.
        strm append: s].
    rec minY < corner y⇒ [
        strm append: (rec origin rect: corner) bitsIntoString]]
bitStringLength | extent [
    extent ← corner - origin.
    ↑ 2 * extent y* (extent x +15/16)]
printon: strm
    [strm print: origin; append: ' rect: '; print: corner]
```

Image

```
blowup: at by: scale | z dest
    [dest ← Rectangle new origin: at extent: self extent*scale.
        [(dest has: origin) or: (dest has: corner) ⇒
            [z ← self bitsIntoString. dest outline.
            self moveto: dest origin. self bitsFromString: z]
        dest outline].
    self blowup: at by: scale spacing: 1]
blowup: at by: scale spacing: spacing
        | extent z inc sinc slice width height dest i j spread
    [extent ← self extent.
    scale ← scale asPoint.  spacing ← spacing asPoint.
```

```
    dest ← Rectangle new origin: at extent: extent*scale.
    z ← 1⊙0. width ← extent x. height ← 0⊙extent y.
    spread ← (scale-spacing) x.
    for: i to: 2 do:                    "first do horiz, then vert"
      [inc ← z * ¯1. sinc ← z * scale.
      slice ← Rectangle new
        origin: (z * width) + [i = 1 ⇒[self origin] at]
        extent: z + height.
      dest ← at + (z * (scale * width)).
      for: j to: width do:              "slice it up"
        [dest ← dest - sinc.
        slice moveby: inc.
        slice blt: dest mode: storing]
      slice ← Rectangle new origin: at + z
            extent: height+(z*(scale-1)).
      for: j to: width do:      "clear slice source"
        [slice clear: white. slice moveby: sinc]
      slice ← Rectangle new origin: at
            extent: height + (z * ((scale*width)-1)).
      for: j to: spread - 1 do:          "spread it out"
        [slice blt: at + z mode: oring]
      z ← 0⊙1.              "flip to do vertical"
      width ← extent y. height ← (scale*extent) x⊙0.
      spread ← (scale-spacing) y]
    ]
blt: dest mode: mode [user croak] primitive: 47
blt: dest mode: mode clippedBy: clipRect
      | destRect clippedSource
    ["Copy the screen bits within my area to the rectangle whose
      origin is dest and whose extent is the same as mine.
      If clipRect is not nil, then copy only those bits within both
      the destination rectangle and clipRect"
    destRect←(Rectangle new origin: dest extent: self extent)
      intersect: user screenrect.
    [nil≡clipRect⇒[] destRect←destRect intersect: clipRect].
    "find the source for the bits after clipping"
    clippedSource←origin+destRect origin-dest.
    BitBlt init
      destbase←mem∘066;
      destraster←user screenrect width/16|2;
      dest←destRect origin;
      extent←destRect extent;
      sourcebase←mem∘066;
      sourceraster←user screenrect width/16|2;
      source←clippedSource;
      copy: mode]
bltcomp: dest mode: mode [user croak] primitive: 48
brush: dest mode: mode color: color [user croak] primitive: 49
brush: dest mode: mode color: color clippedBy: clipRect
      | destRect clippedSource
    ["Brush the screen bits within my area to the rectangle whose
      origin is dest and whose extent is the same as mine.
```

```
        If clipRect is not nil, then brush only those bits within both
        the destination rectangle and clipRect"
    destRect←(Rectangle new origin: dest extent: self extent)
        intersect: user screenrect.
    [nil≡clipRect⇒[] destRect←destRect intersect: clipRect].
    "find the source for the bits after clipping"
    clippedSource←origin+destRect origin-dest.
    BitBlt init
        color←color;
        destbase←mem∘066;
        destraster←user screenrect width/16|2;
        dest←destRect origin;
        extent←destRect extent;
        sourcebase←mem∘066;
        sourceraster←user screenrect width/16|2;
        source←clippedSource;
        paint: mode]
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]
flash [self comp; comp]
```

Border

```
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]
outline     "default border is two thick"
    [self outline: 2]
outline: thick | t
    [t ← (¯1⊙¯1)*thick.
    (self inset: t) clear: black. self clear: white]
┘
SystemOrganization classify: ↗Rectangle under: 'Graphical Objects'.┘
```

"Turtle"

```
Class new title: 'Turtle'
   subclassof: Object
   fields: 'pen ink width dir x xf y yf frame'
   declare: '';
   asFollows↵
```

Turtles can crawl around the screen drawing and printing at any angle.
Dont forget to send them the message init before any drawing commands.

Initialization

```
erase
   [frame clear: white]
frame [↑frame]
frame: frame
init
   [ink ← ¯3. pen ← width ← 1.
   x← y← xf← yf← 0.
   frame ← user screenrect.
   self home]
```

Pen Control

```
black
   [ink ← ¯3]
color: ignored  "Only implemented for PressTurtle"
ink: ink
pen: pen
pendn
   [pen ← 1]
penup
   [pen ← 0]
white
   [ink ← ¯1]
width [↑width]
width: width
xor
   [ink ← ¯2]
```

Drawing

```
filling: expr [↑expr eval]  "Only implemented for PressTurtle"
go: length [user croak] primitive: 53
goto: pt
   [pt x is: Integer⇒[user croak]
   self goto: pt x asInteger⊙pt y asInteger] primitive: 54
home
   [self up; place: frame center-frame origin]
place [↑x⊙y]
place: pt | p
   [p← pen. pen← 0. self goto: pt. pen← p]
stretchto: pt | t old
```

```
   [t ← Turtle init frame: frame. old ← x⊙y.
   t xor; place: old; goto: pt; place: old; goto: pt]
turn: angle
   [dir← dir+angle\360]
up    [dir ← 270]        "Point toward top of screen"
```

Text

```
put: char font: font          "char=ascii Integer, font=font bits (String)"
   [user croak] primitive: 56
show: str font: font | a f          "str=text (String), font=font number (0-9)"
   [f← DefaultTextStyle fontso(font+1).
   forS a from: str doS
      [self put: a font: f]]
```

Examples

```
dragon: n
   [n=0⇒[self go: 10]
   n>0⇒[self dragon: n-1; turn: 90; dragon: 1-n]
         self dragon: ¯1-n; turn: ¯90; dragon: 1+n]
"
Turtle init dragon: 8
"
filberts: order side: s | i n2
   [n2← 1 lshift: order-1.
   self penup; go: 0-n2*s; pendn.
   forS i to: 4 doS
      [self color: i-1*40.
      self fillInS [self hilbert: order side: s; go: s; hilbert: order side: s; go: s].
            self black; hilbert: order side: s; go: s; hilbert: order side: s; go: s.
      self penup; go: n2-1*s; turn: ¯90; go: n2*s; turn: 180; pendn]]
"
Turtle init erase filberts: 3 side: 10.

user displayoffwhileS
   [PressTurtle new init: 'try.press'; filberts: 4 side: 10; close].
"
hilbert: n side: s | a m
   [n=0⇒[self turn: 180]
   [n>0⇒[a←90. m←n-1] a←¯90. m←n+1].
   self turn: a; hilbert: 0-m side: s; turn: a.
   self go: s; hilbert: m side: s;
      turn: 0-a; go: s; turn: 0-a;
      hilbert: m side: s; go: s.
   self turn: a; hilbert: 0-m side: s; turn: a]
"
Turtle init hilbert: 3 side: 4
"
hilberts: n | i s
   [self penup; go: 128; pendn.
   forS i to: n doS
      [s← 256 lshift: 0-i.  self color: n-i*40; width: n-i+1.
```

```
        self penup; go: 0-s/2; turn: ¯90; go: s/2; turn: 90; pendn.
        self hilbert: i side: s; go: s; hilbert: i side: s; go: s]]
"
Turtle init erase hilberts: 5.

user displayoffwhile:
   [PressTurtle new init: 'try2.press'; hilberts: 4; close].
"
mandala: npoints diameter: d | l points i j
   [l← (3.14*d/npoints) asInteger.
   self home; penup; turn: ¯90; go: d/2; turn: 90; go: 0-l/2.
   points← Vector new: npoints.
   for: i to: npoints do:
      [pointso i← self place.
      self go: l; turn: 360/npoints].
   self pendn.
   for: i from: npoints/2 to: 1 by: ¯1 do:
      [self color: (npoints/2)-i*20\250.
      for: j to: npoints do:
         [self place: pointso j; goto: pointso(j+i-1\npoints+1)]]]
"
Turtle init mandala: 30 diameter: 400

user displayoffwhile:
   [PressTurtle new init: 'try.press'; mandala: 30 diameter: 500; close.]
"
spiral: n angle: a | i
   [for: i to: n do:
      [self color: i*2\256; go: i; turn: a]]
"
 Turtle init spiral: 200 angle: 89; home; spiral: 200 angle: ¯89.

 user displayoffwhile: [(PressTurtle new init: 'try.press')
      spiral: 403 angle: 89;
      home; spiral: 403 angle: ¯89; close.]
"
┘
SystemOrganization classify: ↱Turtle under: 'Graphical Objects'.┘
```

"Compressor"

```
Class new title: 'Compressor'
   subclassof: Object
   fields: ''
   declare: 'enc8 dec8 encoding stats decoding ';
   asFollows↵
```

Compresses or decompresses a text string using 4-bit bytes. The average compression is about 35%. (P. Deutsch)

Initialization

```
classInit | t
   [t ← ↶    "Paste in (Compressor new tables: 25) below"
      ((3 6 7 9 13 17 21 23 27 32 34 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 58 59 60 61 62 65 66 67 68 69 70 73 76 77 78 79 80 82 83 84 86 91 93 95 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 124 ) ' [←acfinost' 'e cdlnrstwx' 't :aehiorsy' 's :cehiopst' 'r :aeginost'
'o§:dlmnpruw' 'n :adegiost' 'acdglmnprst' 'i fglmnostz' '              ⍔ "[cfstu' 'l
adefilosu' 'c aehklortu' '

"''(*[cpu' 'd :adeiopst' ':
 ''adfiow' 'f :afiorsty' 'p :aeioprty' 'm .:aeimopy' '[⍔"(]cfpstu' '.
 "'').0]s' 'u eglmnprst' 'h ).:]aeiot' 'g :]aehirst' '←
 ''01lnpst' 'w .:]ahinor' ).
   self initEncode: t. self initDecode: t.
   bytesIn← bytesOut← 0.0]
```

Compressor

```
compress: text | enc en8 snk tab byte out c
   [enc ← encoding. en8 ← enc8.          "For faster access"
   snk ← (String new: text length) asStream.
   tab ← enc§1. out ← false.
   for§ byte from: text do§
      [c ←
         [byte≥(tab§1)⇒
            [byte≤(tab§2)⇒
               [tab§(byte-(tab§1)+3)]
            en8§(byte+1)]
         en8§(byte+1)].
      tab ← enc§(byte+1).
      c<10⇒ [out⇒ [snk next ← (out lshift: 4)+c. out ← false] out ← c]
      c<240⇒ [out⇒ [snk next ← (out lshift: 4)+(c lshift: ¯4).
            out ← c land: 017]
         snk next ← c]
      "12-bit code, ugh"
      [out⇒ [snk next ← (out lshift: 4)+017. snk next ← byte. out ← false]
         snk next ← (byte lshift: ¯4)+0360. out ← byte land: 017].
      ]
   [out⇒[snk next ← (out lshift: 4)+017]].
   ⍔snk contents]
```

```
initEncode: t | u n i min max str s v j
   [u ← Vector new: 256.
   n ← String new: 258. n all ← 0377. no1 ← 0.
   v ← 160.
   for: j from: to1 do: [no(j+3) ← v. v ← v+1].
   enc8 ← (no(3 to: 258)) copy. "80 most common"
   v ← 0.
   for: j from: to(2 to: 11) do: [no(jo1+3) ← v. v ← v+1]. "10 most common"
   u all ← n.
   for: i from: to(2 to: t length) do:
      [min ← io2. max ← ioi length.
      s ← (str ← String new: max-min+3) asStream.
      s next ← min. s next ← max.
      enc8o(min+1 to: max+1) copyto: s.
      v ← 0.
      for: j from: io(2 to: i length) do:
         [stro(j-min+3) ← v. v ← v+1].
      uo(io1+1) ← str].
   encoding ← u]
```

Decompressor

```
decompress: ctext | dec de8 src snk tab byte b
   [dec ← decoding. de8 ← dec8.          "For faster access"
   src ← ctext asStream.
   snk ← (String new: ctext length*2) asStream.
   tab ← deco1. byte ← false.
   while: true do:
      [[byte⇒ [b ← byte land: 017. byte ← false]
       byte ← src next⇒ [b ← byte lshift: ¯4]
       ↑snk contents].
      b<10⇒ [snk next ← b ← tabo(b+1). tab ← deco(b+1)]
      [byte⇒ [b ← byte. byte ← false]
       byte ← src next⇒ [b ← (b lshift: 4)+(byte lshift: ¯4)]
       ↑snk contents].
      b<240⇒ [snk next ← b ← de8o(b-159). tab ← deco(b+1)]
      "12-bit character, ugh"
      [byte⇒ [snk next ← (b-240 lshift: 4)+(byte land: 017). byte ← false]
       byte ← src next.
       snk next ← (b-240 lshift: 4)+(byte lshift: ¯4)].
      tab ← deco1.
      ]]
initDecode: t | u d j
   [dec8 ← to1 copyto: (String new: (to1) length).
   u ← Vector new: 256.
   d ← Vector new: 10.
   for: j to: 10 do: [doj ← to(j+1)o1]. "10 most common"
   u all ← d.
   for: j from: to(2 to: t length) do:
      [uo(jo1+1) ← (jo(2 to: 11)) copyto: (String new: 10)].
   decoding ← u]
```

Statistics

```
clear [stats← nil]
scan: text | src last v b t
   [src ← text asStream.
   (last ← src next)≡false⇒ []
   last ← [last≥128⇒[129]last+1].
   while: (b ← src next) do:
      [b ← [b≥128⇒[129]b+1].
      v ← stats∘last.
      [(t ← v∘b+1)=256⇒[v is: String⇒
         [stats∘last ← v ← v copyto: (Vector new: v length)]]].
      v∘b ← t.
      last ← b]]
scanClass: class | sel
   [class is: Vector⇒
      [user displayoffwhile:
         [for: sel from: class do:
            [user show: sel; space.
            self scanClass: Smalltalk∘sel]]]
   for: sel from: class selectors do:
      [sel=↗ClassOrganization⇒ []
      user show: sel; space.
      self scan: (class code: sel)].
   ]
sorting: v | n z i w
   [n ← (Vector new: v length) asStream. z ← (Vector new: v length) asStream.
   "Must float all of v so comparisons don't bomb"
   w ← Vector new: v length.
   for: i to: v length do:
      [v∘i=0⇒ [z next ← i] w∘i ← (v∘i) asFloat. n next ← i].
   ↑(w∘n contents) sort map reverse copy concat: z contents]
tables: n | c sum so j i v non "Produce encoding tables"
   ["To use this, gather statistics, then paste the results of a call on tables: into
the assignment in the classInit method."
   c ← Counter new.
   sum ← stats transform: v to: [c init; addvec: v; value].
   so ← self sorting: sum.
   non ← (so∘(1 to: 80)) copy sort transform: i to: i-1. "80 most common
chars"
   ↑non inVector concat:
      (so∘(1 to: n) transform: i to:
         [(i-1) inString concat:
            ( (((self sorting: stats∘i)∘(1 to: 10)) "most frequent successors"
               copyto: (String new: 10)) sort transform: j to: j-1)])]
zero | i t
   [stats ← Vector new: 129.
   for: i to: 129 do:
      [stats∘i ← t ← String new: 129.
      t all ← 0]]
↵
SystemOrganization classify: ↗Compressor under: 'Text Objects'.↵
Compressor classInit↵
```

"Dispframe"

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

I am a dialog window

Initialization

```
classInit [prompt ← '_'o1. doit ← '_!'o1]
frame ← r
   [text para: nil frame: r]
init
   [text ← Textframe new.
   self of: (String new: 16)]
rect: r
   [self init; frame ← r; clear]
```

Scheduler

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

Dialog

```
ev | t
   [while: [self cr. t ← self request: ''] do:
      [self space; print: nil 'st]
   ↑false]
kbd | n t    "false if user pauses, nil if ctrl-d, all input since prompt if _!"
   [while: user kbck do:
      [t ← user kbd.
      t=132⇒ [self append: 'done.'; show. ↑nil]; "ctl-d for done"
      =8⇒ [self last=prompt⇒[].self skip: ¯1]; "backspace"
      =30⇒ [n ← array:(position to: 1 by:¯1) find: prompt.
            n=0⇒[self append: 'lost beginning'; prompt]
            t← self last: n-1. self next← doit; show. ↑t];          "do-it (LF)"
      =145⇒ [self last=prompt⇒[] self skip: ¯1.        "ctl-w for backspace word"
```

```
            while: (position>0 and: self last tokenish) do: [self skip: ¯1]];
        =15⇒[self reset; prompt]        "ctl-x clears frame"
        self next ← t]
    self show. ↑false]
prompt [self cr; next← prompt; show]
read | t        "false if ctrl-d, all input since prompt if ↵"
    [self next← prompt; show.
    until: [user kbck⇒[t← self kbd] false] do: []
    t≡nil⇒[↑false] ↑t]
request: s        "false if ctrl-d, all input since prompt if ↵"
    [self append: s. ↑self read]
```

Image

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

Access to Parts

```
frame
    [↑text frame]
text
    [↑text]
↵
SystemOrganization classify: ↶Dispframe under: 'Text Objects'.↵
Dispframe classInit↵
```

```
                                                    "FieldNameCollector
                                                    "
Class new title: 'FieldNameCollector'
   subclassof: TokenCollector
   fields: ''
   declare: '';
   asFollows↵

This class collects identifiers as Strings (not UniqueStrings)
for scanning of class field names (Class.instvars)
by the compiler(Generator) and debugger(VariablePane)

Valid fields
identifier: s [sink next← s]

Invalid fields
leftparen [self next← '('] "just for error message"
next← value
   [user notify: 'Invalid field name: '+value asString]
rightparen [self next← ')'] "just for error message"
↵
SystemOrganization classify: ↗FieldNameCollector under: 'Text Objects'.↵
```

"Paragraph"

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

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

Initialization of parts

```
copy [↑self class new text: text runs: runs alignment: alignment]
text: text
   [alignment ← 0]
text: text alignment: alignment
text: text runs: runs alignment: alignment
```

Normal access

```
ox [↑text ox]
asParagraph [↑self]
asVector [↑text asVector]
copy: a to: b        "Return a copied subrange of this paragraph"
   [↑self class new
         text: (text copy: a to: b)
         runs: (self run: a to: b)
         alignment: alignment]
findString: str startingAt: start
   [↑text findString: str startingAt: start]
length [↑text length]
replace: a to: b by: c   ["alters self - doesnt copy"
   [runs≡nil⇒[]
   runs ← self runcat:
      (self run: 1 to: a-1),
      [c is: self class⇒ [c runs]
      self makerun: c length val: [
         runs empty⇒[0]
         runs∘((self runfind: b)∘1+1)]],
      (self run: b+1 to: text length)].
   text ← text replace: a to: b by: [c is: self class⇒[c text] c]]
style [↑DefaultStyle]
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

```
makeBoldPattern | s i c
   [s ← text asStream. i← 0.
   until: [c← s next⇒        " scan to bracket, bar or comment "
            [c=91⇒[true]; =124⇒[true]; =34⇒[true]; =25⇒[true] false]
         true]        "end"
      do: [i← i+1].
   self maskrun: 1 to: i under: 1 to: 1]
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 run: 1 to: i-1), r, (self run: j+1 to: text length)]
maskrunsunder: m to: val
   [self maskrun: 1 to: text length under: m to: val]
run: a to: b | c        "subrange of run"
   [a>b⇒[↑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↓1 ← 1+(runs↓(a↓1))- (a↓2).              "trim the end lengths"
   c↓(c length-1) ← b↓2].
   ↑c]
runcat: rs | r olen len oc c nr [
   "concatenate and compact a vector of runs"
   nr ← Stream new of: (String new: 30).
   oc ← false.
   for: r from: [rs is: Vector⇒ [rs] rs inVector] do: [
      r empty⇒ []
      r ← r asStream.
```

```
      whileꜰ (len ← r next) doꜰ [
         c ← r next.
         len = 0⇒ ["ignore empty runs (shouldn't be any)"]
         oc = c⇒ [
            (olen ← olen+len) ≤ 255⇒ []
            nr next ← 255; next ← oc.
            olen ← olen-255]
         [oc⇒ [nr next ← olen; next ← oc] "first time thru"].
         olen ← len. oc ← c]].
   oc⇒ [
      "leftovers"
      nr next ← olen; next ← oc.
      ↑nr contents]
   ↑nullString]
runcat: x to: y [↑self runcat: x, y]
runfind: index | run      t    "index into run"
   [run←1.
   whileꜰ (t ← index - (runs∘run)) > 0 doꜰ
      [index ← t. 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.
   t← s upto: 0134.
   [t length>30⇒[user notify: 'Suspicious Bravo trailer']].
   [t has: 'c'∘1⇒[self center] t has: 'j'∘1⇒[self justify]].
   len ← 0.
   untilꜰ s end doꜰ
      [(ch ← s next) isdigit ⇒ [len ← (len*10)+ch-060]
      i ← i+len. len ← 0.
      ch=0146 ⇒[t ← 0.
         whileꜰ [ch ← s next⇒[ch isdigit] false] doꜰ
            [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
      ]
   ]
bravoRuns "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: [alignment=1⇒['j\g']; =2⇒['c\g'] 'r\g'].
   len ← 0. old ← 0400.
   [runs≡nil ⇒ []
```

```
    for: i from: (1 to: runs length by: 2) do:
        [dif ← old lxor: (new ← runs∘(i+1)).
        (dif land: 0367)=0 ⇒ "No changes" [len ← len+(runs∘i)]
        [i=1⇒[] len printon: s].
        for: bit to: 3 do:
            [(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. ∩s contents]
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=newtext length⇒[i+j] i+j-1] by: ''.
        loc ← i+1]
    ∩newpara]
toBravo [∩(text concat: self bravoRuns) asParagraph]
```

Press printing

```
asTextEntity [∩(
    TextEntity new text: text runs: runs alignment: alignment)
        style: DefaultStyle]
hidePress: press first: n length: len | s [
    "stash information (name(code) and value) in a press file.
    see fromPress:name:value: for reciprocal operations"

    "text is divided among entities, and was just written in DL"
    (s ← String new: 2) word: 1 ← len.
    press skipcode: 0 data: s.

    "put all runs (if any) with first entity"
    [n> 1 or: (runs≡nil or: runs empty)⇒ []
    press skipcode: 2 data: runs].

    "if entity is split, use illegal alignment"
    press skipcode: 1 data: ([len < text length⇒ [99] alignment]) inString;
        closeEntity]
presson: press in: r | char pos s3 y chop [
    "Output paragraph inside rectangle (page coordinates)"
    y ← r corner y.     "We change corner y later"
    s3 ← ParagraphScanner new of: self to: press style: self style textStyle.
    s3 init in: r.
    pos ← s3 position.
```

```
    chop ← [alignment=1⇒ [0] alignment].
    while: (y and: (char ← s3 scan)) do: [
        char = 011 ⇒ [s3 tab]
        char = 040 or: char = 015 ⇒      [
            "carriage return or exceeded max width and backed up to blank"
            y ← s3 printfrom: pos aligned: [char=040⇒[alignment] chop] skip: 1⇒
                [r corner y ← y. s3 init in: r. pos ← s3 position]]
        char ≡ true ⇒[
            "exceeded max width with no blanks in line"
            s3 backup.
            y ← s3 printfrom: pos aligned: 0 skip: 0⇒
                [r corner y ← y. s3 init in: r. pos ← s3 position]]
        "user notify: 'unimplemented control char'"].
    "Put out trailing text if any"
    y and: (s3 width=0 or: (y ← s3 printfrom: pos aligned: chop skip: 0))⇒ [
        press append: text.
        ↑y]
    press append: text∘(1 to: pos).
    ↑self copy: pos + 1 to: text length]
```

Packing into String

compressIntoString

```
    [↑self packIntoString: (Compressor new compress: text)
            code: alignment+16]
```

packIntoString `[↑self packIntoString: text code: alignment]`

packIntoString: t code: a `| r s      [`

```
    "pack runs and t into a string"
    r ← [runs≡nil⇒[nullString] runs].
    Set new of: (s ← String new: r length + t length + [r length<192⇒[2] 3]);
        next ← a;
        nextString ← r;
        append: t.
    ↑s]
```

unpackFromString: s `[`

```
    "undo packIntoString or compressIntoString"
    s is: self class⇒[↑s copy]
    s ← s viewer.
    alignment ← s next.
    [(runs ← s nextString) empty⇒ [runs ← nil]].
    alignment ≥ 16⇒[
        alignment ← alignment\16.
        text ← Compressor new decompress: s]
    text ← s rest]
```

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

"Reader"

```
Class new title: 'Reader'
   subclassof: Object
   fields: 'source collector token nextchar typetbl '
   declare: 'typetable ';
   asFollows↓
```

Converts a string to tokens. The collector defines what to do for each kind of token: see TokenCollector and Compressor for examples. (P. Deutsch)

Initialization

```
classInit | strm type first last i "Initialize the type and mask tables"
   [typetable← String new: 256.
   strm← Stream new of: ↗(
      5 0 0377  "(initialize)"

      1 0101 0132  1 0141 0172  "upper and lower case letters"
      2 060 071  "digits"
      3 072 072  3 03 03  "colon, open colon"
      4 011 012  4 014 015  4 040 040  "TAB, LF, FF, CR, blank"
      "5 is one-char tokens"
      6 042 042  6 031 031  "comment quote and ↵"
      7 047 047  "string quote"
      8 025 025  "high-minus"
      9 032 032  "↑Z (format trailer)"
      10 036 036  "DOIT"
      11 050 051  "open and close paren"
      ).
   while: (type← strm next) do:
      [first← strm next. last← strm next.
      for: i from: (first+1 to: last+1) do:
         [typetable∘i← type]
      ]
   ]
of: s
   [typetbl← typetable.
   token← Stream default.
   source← s asStream.
   self step]
```

Main reader

```
read
   [↑self readInto: TokenCollector default]
readAtom: ncolons | type s
   [token reset.
   while: [token next← nextchar.
      (nextchar← source next)→[(type← typetbl∘(nextchar+1))≤3]
      false]
   do:
      [type=3→[ncolons← ncolons+1]].
```

```
    s← token contents.
    ncolons=0⇒[collector identifier: s];
      >1⇒[collector otheratom: s].
    s length=1⇒[collector otheratom: s] ": or ; alone"
    s↓0 length=072⇒[collector keyword: s];
      =03⇒[collector keyword: s].
    collector otheratom: s. "Colon wasn't last character"
    ]
readInto: collector | x
    [while; nextchar do;
      [x← typetblo(nextchar+1).
      "See classInit for the meanings of the type codes"
      x=4⇒ [collector separator: nextchar. nextchar← source next];
      =1⇒ [self readatom: 0];
      =5⇒ [collector onechar: nextchar. nextchar← source next];
      =6⇒ [self upto: nextchar⇒[collector notify: 'Unmatched comment quote']
        collector comment: token contents];
      =2⇒ [self readnum];
      =11⇒ [[nextchar=050⇒[collector leftparen] collector rightparen].
        nextchar← source next];
      =7⇒ [self upto: nextchar⇒[collector notify: 'Unmatched string quote']
        collector string: token contents];
      =8⇒ [self readnum];
      =9⇒ [self upto: 015⇒[collector notify: '↑Z without CR']
        collector trailer: token contents];
      =10⇒ [↑collector contents];
      =3⇒ [self readatom: 1]
      ]
    ↑collector contents]
readnum | val d e
    [val← self rdint: 025.
    nextchar=056⇒          "check for decimal point"
      [self step.
      nextchar≡false or; nextchar isdigit≡false⇒
        [collector integer: val. collector onechar: 056]          "was <integer> . "
      d← self rdint: ¯1.      "fraction part"
      [nextchar=0145⇒          "check for e<exponent> "
        [self step. e← self rdint: 025]
      e← ''].
      collector float: val fraction: d exp: e]
    collector integer: val]
```

Internal readers

```
rdint: char "Read an integer, allow char as first char"
    [token reset.
    [nextchar=char⇒[token next← char. self step]].
    while; nextchar do;
      [nextchar<060⇒[↑token contents]
      nextchar>071⇒[↑token contents]
      token next← nextchar. nextchar← source next].
    ↑token contents]
step
```

```
    [nextchar← source next]
upto: char | start "Knows about doubled ' in strings"
    [start← source position.
    token reset.
    while: (nextchar← source next) do:
        [[nextchar=char⇒
            [self step. char≠047⇒[↑false] nextchar≠047⇒[↑false]]].
        token next← nextchar].
    "Ran off end, back up."
    source skip: start - 1 - source position.
    ↑true]
┘
SystemOrganization classify: ↶Reader under: 'Text Objects'.┘
Reader classInit┘
```

"Textframe"

```
Class new title: 'Textframe'
   subclassof: Object
   fields: 'frame para style reply1 reply2 window'
   declare: '';
   asFollows_!
```

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

Initialization

```
para: para frame: frame
   [window←frame.
   reply1←reply2←0.
   style←DefaultTextStyle]
para: para frame: frame style: style
   [window←frame.
   reply1←reply2←0]
```

Scheduling

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

Image

```
asForm: pt | char ul ur f
      "put bits of character into a form -- when Form package in system"
   [char ← self charofpt: pt. ul ← reply1.
   self ptofchar: char + 1. ur ← reply1.
   f ← Form new size: ur x - ul x by: reply2 y - reply1 y.
   f translate: ul; scale: 1. ∩f ]
comp
   [window comp]
copyto: path effect: effect | i oldmode          "show clipped inside rect"
   [
   path is: Point ⇒
      [oldmode ← style mode. style mode: effect. window translateto: path. self
show. style mode: oldmode.]
   path is: Path ⇒ [for: i to: path length do: [ self copyto: path∘i effect: effect] ]
   ]
corner [∩frame corner]
displayat: path effect: effect clippedBy: cliprect| i oldmode          "show clipped
inside rect"
   [
   path is: Point ⇒
      [oldmode ← style mode. style mode: effect. frame translateto: path.
window ← cliprect. self show. style mode: oldmode.]
   path is: Path ⇒ [for: i to: path length do: [ self displayat: path∘i effect:
effect] ]
```

```
    ]
erase
    [(window inset: (¯2⊙¯2)) clear]
extent [↑frame extent]
frame [↑frame]
frame ← frame
    ["Change my frame and window."
    window ← frame.
    ]
height [↑frame height]
origin [↑frame origin]
outline
    [window border: 2 color: black]
para [↑para]
showin: rect | old          "show clipped inside rect"
    [old ← window. window ← rect. self show. window ← old]
size [↑frame extent]
width [↑frame width]
window [↑window]
```

Text

```
charnearpt: pt [user croak] primitive: 58
charofpt: pt [user croak] primitive: 58
dopressjst: pt
    ["For building justified lines in Press Format files

        [reply1 x ← trailingbits.
        reply1 y ← internalspaces.
        reply2 y ← heightofline.
        [reply2 x < 0 ⇒[linenotjustified]].
        ↑lastcharinline]

            user will have to back up in string to find last printing character
            (at least non-space, cr, or tab)"

    user croak] primitive: 64
findmaxx: char
    [char is: Integer⇒[user croak]
    self findmaxx: char asInteger] primitive: 63
lastshown
    [↑reply1]
lineheight
    [↑style lineheight]
maxx: char
    [self findmaxx: char. ↑ reply1]
measuretext: startx to: stopx string: string from: first to: last font: font
    ["Returns character index of character immediately following character
      causing exception condition.

        Reply1    =   Exception code.
                  =   1   Encountered space, cr, tab, or ascii 0.
                  =   2   Crossed stopx.
```

```
                = 3   Both 1 and 2.
                = 4   Encountered last before hitting stopx or special character.

    Reply2   =   Leftx of character causing exception condition."

  user croak] primitive: 103
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 at: pt maxextent: maxextent
  [para ← para asParagraph.
  window ← frame ← Rectangle new origin: pt extent: maxextent.
  self findmaxx: para length+1.
  window growto: reply2.
  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
  [char is: Integer⇒[user croak]
  self selectchar: char asInteger] primitive: 59
show [user croak] primitive: 57
show: para
  [para ← para asParagraph. self show]
showline: first to: last lastx: lastx spaces: spaces trailingspaces: trailingspaces
startrun: startrun firstrunlength: firstrunlength
  ["For use in conjunction with measuretext primitive.

      Characters first through last will be displayed, clipped by the frame
      and window. The lastx should be the lastx of last. If justification
      is on in para and spaces is > 0, the line will be justified.
      Passing spaces as 0 causes suppression of justificaton even if
      justification is turned on in para; space characters will be displayed
      normally even when spaces is 0.
      Trailingspaces is needed for justification. Startrun is an integer index
      into the runs, indicating which run to start on. Firstrunlength
      is the number of characters remaining in run startrun. Note that
```

while runs in a paragraph are allocated as a string, **startrun** will be considered a word index.

maxascent and **maxdescent** in **style** must be nonnil to avoid a croak."

```
user croak] primitive: 104
```

Conversion

```
printon: strm
    [strm append: 'a Textframe']
```

Printing

```
hardcopy | p
    [p ← dpO pressfile: 'frame.press'.
    self hardcopy: p.
    p page; close]
hardcopy: p | win e
    [win ← window inset: ¯2.
    for: e from: (win minus: window) do:
        [p showrect: e color: 0].
    p closeEntity.
    e← p transrect: win.
    p entityorigin: (800⊙800)+(e minX⊙0).
    para asParagraph presson: p in: e.
    p closeEntity]
┘
SystemOrganization classify: ↱Textframe under: 'Text Objects'.┘
```

"TextStyle"

```
Class new title: 'TextStyle'
   subclassof: Object
   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"'
   declare: '';
   asFollows_!
```

I am a specification of how to display a paragraph. I include a font set, a tab spacing, a space size, etc. If I do not specify ascent and descent from the baseline, then each line displayed will adjust to its tallest characters.

Initialization

```
default
   [tabandspace ← mode ← 0. self mode: 0; tab: 20; space: 5.
   fonts ← Vector new: 16. fontnames ← Vector new: 16.
   self setfont: 0 name: 'CREAM10'.      "Put default font in font 0"
   ]
mode
   [↑ mode]
mode: mode
space
   [↑ (tabandspace land: 0377)]
space: t
   [tabandspace ← (tabandspace land: 0177400) + (t land: 0377)]
tab
   [↑ ((tabandspace land: 0177400) rshift: 8)]
tab: t
   [tabandspace ← (tabandspace land: 0377) + (t lshift: 8)]
```

Fonts

```
fontfamily: n | s char
   ["return the family name taken out of fontnames"
   s ← Stream default.
   for: char from: fontnames o n do:
      [char isletter⇒ [s next ← char]
      ↑s contents]]
fontnames [↑fontnames]
fonts [↑fonts]
fontsize: n | s c size
   ["return size from fontname"
```

```
    size ← 0. s ← (fontnames o n) asStream.
    whileS (c ← s next) isletter doS [].
    whileS [size ← size*10 + (c - 060). c ← s next] doS [].
    ↑size]
setfont: n fromfile: name | f
    [(f ← File new old named: name + '.strike.')⇒
        [self setfont: n name: name fromstring: f contents]
    user notify: 'Font ' + name + '.strike. not on this disk']
setfont: n name: name | ucn
    [FontDict has: (ucn← name asUppercase)⇒
        [self setfont: n name: ucn fromstring: FontDict↑ucn]
    self setfont: n fromfile: name]
setfont: n name: name fromstring: string
        "Should update maxascent, maxdescent"
    [fontnames↑(n+1) ← name asUppercase.
    fonts↑(n+1) ← string.
    FontDict insert: fontnames↑(n+1) with: string]
setoffsetfont: n from: m by: d
    [fonts↑n ← m + [d<0⇒ [0200] 0] + (d lshift: 8)]
writeset: styleindex
    [self writeset: styleindex as: fontnames↑(styleindex+1)]
writeset: styleindex as: name
    ["write out a formset on name with strike extention"
    name ← name + '.strike.'.
    (dp0 file: name) append: fonts↑ (styleindex+1) ; close.
    ]
```

Access

```
heightofset: styleindex
    ["Return height of formset in style"
    ↑(fonts↑(styleindex+1) word: 6) +
        (fonts↑(styleindex+1) word: 7)]
lineheight
    [((maxascent ≡ nil) or: (maxdescent ≡ nil)) ⇒
    [↑((fonts↑1) word: 6) + ((fonts↑1) word: 7)]
    ↑maxascent+maxdescent]
maxascent
    [↑ maxascent ]
maxascent: maxascent
maxdescent
    [ ↑ maxdescent ]
maxdescent: maxdescent
maxwidthofset: styleindex
    ["Return maximum width of formset in style"
    ↑(fonts↑(styleindex+1) word: 4)]
nameofset: styleindex
    ["Return name of formset in style"
    ↑(fontnames↑(styleindex+1))]
strikeofset: styleindex
    ["Return strike of formset in style"
    ↑(fonts↑(styleindex+1))]
```

```
SystemOrganization classify: ↑TextStyle under: 'Text Objects'.↵
```

"TokenCollector"

```
Class new title: 'TokenCollector'
   subclassof: Object
   fields: 'sink parenstack'
   declare: '';
   asFollows_!
```

Provides standard token-collecting behavior for Reader .
See Reader-readInto: for more insight. (P. Deutsch)

Initialization

default
```
   [self to: (Vector new: 20)]
```
to: v *"Initialize"*
```
   [sink← v asStream.
   parenstack← (Vector new: 5) asStream]
```

Finalization

contents *"Close all parentheses first"*
```
   [until$ parenstack empty do$ [self rightparen].
   ↑sink contents]
```
next← obj [sink next← obj] "subclasses can override easily"
notify: errorString
```
   [user notify: errorString]
```

Constructors

comment: s
float: i fraction: f exp: e
```
   [self next← (i+'.'+f+'e'+e) asFloat]
```
identifier: s
```
   [self next← s unique]
```
integer: s
```
   [self next← s asInteger]
```
keyword: s
```
   [self next← s unique]
```
leftparen
```
   [parenstack next← sink.
   sink← (Vector new: 10) asStream]
```
onechar: c | x
```
   [x← String new: 1. x↓1← c. self next← x unique]
```
otheratom: s
```
   [self next← s unique]
```
rightparen
```
   [parenstack empty⊃[] "Error will be caught elsewhere"
   parenstack last next← sink contents.
   sink← parenstack pop]
```
separator: c
string: s
```
   [self next← s]
```
trailer: s

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

"BrowseWindow"

```
Class new title: 'BrowseWindow'
   subclassof: PanedWindow
   fields: ''
   declare: 'stdTemplates ';
   asFollows↵
```

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

Initialization

classInit

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

"CodeWindow"

```
Class new title: 'CodeWindow'
   subclassof: PanedWindow
   fields: ''
   declare: 'stdTemplates ';
   asFollows↵
```

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

Initialization

```
class: class selector: selector para: para formerly: oldpara | codePane
   [codePane ← CodePane new class: class selector: selector para: nil.
   self title: class title+ ' ' + selector with: codePane inVector at: stdTemplates.
   self newframe; show.
   codePane showing: para; formerly: oldpara; from: codePane]
classInit
   [stdTemplates ← (0@0 rect: 36@36) inVector]
file: file | codePane
   [codePane ← CodePane new.
   self title: file title asString with: codePane inVector at: stdTemplates.
   self newframe; show.
   codePane showing: file contents asParagraph; from: file]
↵
SystemOrganization classify: #CodeWindow under: 'Windows'.↵
CodeWindow classInit↵
```

"DocumentWindow"

```
Class new title: 'DocumentWindow'
   subclassof: Window
   fields: 'projector document selection title scrollBar'
   declare: '';
   asFollows↓
```

I am a window through which my projector is currently projecting my document. If not false, then selection is an entity of document which is currently selected.

Initialization

```
on: document named: title
   [self newframe. "creates scrollBar via frame:"
   projector ← Projector new projecting: document frame origin
      of: document onto: frame.
   document isSeenIn: self.
   selection ← Selection new thru: projector.
   self enter]
```

Window Protocol

```
bluebug
   [↑self hideBarsWhile: super bluebug]
close
   [selection ← selection close.
   scrollBar close.
   document isntSeenIn: self]
enter
   [self show.
   selection ← selection enter.
   self showBars]
hardcopy [document hardcopy]
kbd [selection ← selection kbd]
leave
   [self hideBars.
   selection ← selection leave]
outside [↑scrollBar startup]
redbug [selection ← selection redbug]
scrollPos
   [projector=nil⇒ [↑0.0] ↑projector scrollPos]
scrollTo: f [selection complement.
   projector scrollTo: projector aperture minX⊙(f*document height) asInteger.
   selection complement]
scrollUp: n [selection complement.
   projector scrollBy: 0⊙n.
   selection complement]
show
   [super show. "borders"
   projector=nil⇒[]
   document showThru: projector]
```

```
showOnly: screenRect
   [document showThru: (projector subProjectorTo: screenRect of: document)
clear]
yellowbug [selection ← selection yellowbug]
```

Border

```
contains: rect
   [↑rect isWithin: frame]
frame [↑frame]
frame: frame
   [frame ← self fixframe: frame.
   [projector ≡ nil⇒ [] projector screenAperture ← frame].
   ([scrollBar≡nil⇒ [scrollBar←ScrollBar new] scrollBar]) on: frame from: self]
hideBars
   [scrollBar hide]
hideBarsWhile: expr | υ
   [self hideBars. υ ← expr eval. self showBars. ↑υ]
showBars
   [scrollBar show]
title [↑title]
↵
SystemOrganization classify: ↗DocumentWindow under: 'Windows'.↵
```

"FontWindow"

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

I am a window that displays one blown up character at a time of a strike-format font

Help

```
help
["
**sysFontWindow is declared in the Smalltalk dictionary, and bound to the
font window displayed on the screen of most system releases -- intended to
provide an easy way to play around with the font editor.
**to create a window for editing default font 0 at middle-click:
    user schedule: (sysFontWindow ← 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:
   sysFontWindow setascent: 2.        **Deltas -- for entire font**
   sysFontWindow setascent: ¯3.
   sysFontWindow setdescent: 2.
   sysFontWindow setdescent: ¯2.
   sysFontWindow setchar: 046.
   sysFontWindow setwidth: 5.        **Absolute--for char in window.
   Useful for characters of zero width.**
"]
```

Initialization

```
altostyle: altostyle fontnumber: fontnumber at: origin
   [       "set up an instance"
   [fontmenu≡nil⇒[self init]].
   scale ← 9. charstr ← String new: 1. char ← 65. charstr◦1 ← char.
   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.
    ]
classInit
    [fontmenu ← Menu new string:
'strike
set width
debug
move
close']
```

Scheduler

```
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⇒[self strike];          "put strike of font in dialogue window"
                =2⇒[self setwidth];                      "grow character"
                =3⇒[self updateseglength: font raster: fontraster.
                    self updatemaxwidth.        "clean things up"
                    user notify: 'font debugging'];
                =4⇒[self frame];                         "move fontwindow"
                =5⇒[clearframe clear.
                    self updateseglength: font raster: fontraster.
                    self updatemaxwidth.        "clean things up"
                    user unschedule: self. ↑false]]
        user kbck⇒
            [char ← user kbd. self setchar: char]
        ]
    user anybug⇒[↑false]
    ]
firsttime                   "upon entry"
    [
    clearframe has: user mp⇒[self show]
    ↑false
    ]
lasttime                    "upon exit"
    []
```

Editing

```
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"
          (fonto(1 to: 18) concat: updatedfonto(1 to: updatedfont length)).
        updatedfont ←          "now add on rest of old font"
          (updatedfont concat: fonto(19 to: font length)).
        ]
      updatedfont ← (fonto(1 to: 18) concat:                          "shrink"
        fonto((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"
      self updateseglength: font raster: fontraster.
      ]
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; dstrike← true.  "lock font and get core ptr"
      bitsetter fill: storing color: color.                 "turn bit on/off in font"]
setchar: char
      [
      charstro1 ← 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"
    self updateseglength: font raster: fontraster.
    ]
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: charstr∘1.
    ]
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: background
          ].
        ].
      ].
      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.
   bitsetter sourcebase ← font.
   bitsetter sstrike← true; dstrike← true.
   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.
      "shift x-vals"
   for: i from: ((char + 1)
      to: (2 + (updatedfont word: 3) - (updatedfont word: 2) "max")) 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"
   self updateseglength: font raster: fontraster.
   self updatemaxwidth.
   ].
   ]

Image
frame
   [clearframe clear.
   frame moveto:
   (OriginCursor showwhile:
      [user waitbug⇒[user mp]]).
   self setchar: char.
   ]
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 style: altostyle.
    tempframe show.
    frame blowup: (frame origin) by: scale.
    ]
```

Strike format

```
appendxtable: thefont
    [                                   "put font's xtable on end of a grown/shrunk font"
    thefont ← thefont concat: font o ((fontxtabl * 2 - 1) to: font length).
    ↑thefont.
    ]
cufixup | "Carnegie-Mellon fixup for scale compatibility"
    [boxer extent ← (scale-1)⊙(scale-1).
    frame extent ← scale⊙0.
    clearframe extent ← scale⊙0.
    ]
makecu: name scale: cuscale "Put out font in Carnegie-Mellon format"
        | f svscale svchar bitwidth i bitmover bits
    [f ← (dp0 file: name + '.cu.').
    self updateseglength: font raster: fontraster. self updatemaxwidth.
    svscale ← scale. scale ← cuscale. svchar ← char.
    self cufixup.
    f nextword ← fontht*scale.
    f nextword ← (bitwidth ← (font word: 4)) * scale + 15 / 16.
    bits ← String new: ((fontht * scale) * ((bitwidth * scale + 15)/16)) * 2.
    bitmover ← BitBlt init.
    bitmover destbase ← bits lock.
    bitmover destraster ← bitwidth * scale + 15 / 16.
    bitmover destx ← 0.
    bitmover desty ← 0.
    bitmover sourcebase ← mem∘066.
    bitmover sourceraster ← (user screenrect extent x) + 15/16.
    bitmover sourcex ← frame origin x.
    bitmover sourcey ← frame origin y.

    for: i from: ((font word: 2) to: (font word: 3) by: 1) do:
        [self setchar: i.
        f nextword ← i. f nextword ← charwid*scale.
        bitmover width ← (frame extent x) * scale.
        bitmover height ← (frame extent y) * scale.
        bits all ← 0.
        bitmover copy: storing.
        f append: bits].
    f shorten. f close. scale ← svscale. self cufixup. bits unlock. self setchar:
svchar]
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.
    ]
strike | i    showstr "Put a strike of font into dialogue window"
    [showstr ← String new: 128. for: i to: 128 do: [showstr:i ←i].
    user clearshow: showstr]
updatemaxwidth | newmaxwidth i
    [                          "update max width"
    newmaxwidth ← 0.
    for: i from: (fontxtabl to: fontxtabl + ((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"
                ).
```

```
    ]
]
SystemOrganization classify: ↑FontWindow under: 'Windows'.]
FontWindow classInit]
```

"GalleyWindow"

```
Class new title: 'GalleyWindow'
   subclassof: DocumentWindow
   fields: ''
   declare: '';
   asFollows↵
```

I am document window whose document is a galley that can be converted to and from Bravo format.

File Conversion

```
print | f
   [user displayoffwhile:
      [f ← dpo file: title.
       document writeBravo: f.
       f close]]
```

Window Protocol

```
fixframe: f
   [f width ← document frame width. ↑f]
↵
SystemOrganization classify: ↷GalleyWindow under: 'Windows'.↵
```

"InspectWindow"

```
Class new title: 'InspectWindow'
   subclassof: PanedWindow
   fields: 'variables'
   declare: 'stdTemplates ';
   asFollows↵
```

I am a paned window with a variable pane that displays the fields of an object and a code pane to display their values and to evaluate in their context.

Initialization

```
classInit
   [stdTemplates ← (0@0 rect: 120@36), (120@0 rect: 360@36)]
of: object | instanceVarPane instanceValuePane safeVec n
   [instanceVarPane ← VariablePane new. instanceValuePane ← CodePane
new.
   self title: object class title
      with: (instanceVarPane, instanceValuePane) at: stdTemplates.
   self newframe; show.
   instanceVarPane to: instanceValuePane.
   instanceValuePane from: instanceVarPane.
   variables ← (Vector new: 16) asStream.
      [object class is: VariableLengthClass⇒
         [for: n from: object fields do:
            [self identifier: n]]
      object class fieldNamesInto: self].
   safeVec ← Vector new: 2. safeVec all ← object.
   instanceVarPane names: (↗(self) concat: variables contents) values: safeVec
wrt: false]
```

Private

```
comment: s        "called by of: via Class fieldNamesInto"
contents     "called by of: via Class fieldNamesInto"
identifier: s        "called by of: via Class fieldNamesInto"
   [variables next ← s]
separator: c        "called by of: via Class fieldNamesInto"
trailer: s     "called by of: via Class fieldNamesInto"
↵
SystemOrganization classify: ↗InspectWindow under: 'Windows'.↵
InspectWindow classInit↵
```

"NotifyWindow"

```
Class new title: 'NotifyWindow'
   subclassof: PanedWindow
   fields: 'enoughpanes'
   declare: 'bigTemplates smallFrame smallTemplates ';
   asFollows_!
```

I am a paned window with one or six panes that display the context of an error or breakpoint.

Initialization

classInit

```
   [smallTemplates ← (0⊙0 rect: 36⊙36) inVector.
   bigTemplates ← (0⊙0 rect: 12⊙18), (12⊙0 rect: 36⊙18), (0⊙18 rect: 12⊙27),
(12⊙18 rect: 36⊙27), (0⊙27 rect: 12⊙36), (12⊙27 rect: 36⊙36).
   smallFrame ← 204⊙366 rect: 404⊙402]
```

of: titleString level: level interrupt: flag | stackPane

```
   [NotifyFlag ← false. CitationReasons ← CitationReasons+1.
   stackPane ← StackPane new.
   self title: titleString with: stackPane inVector at: smallTemplates.
   smallFrame moveto:
      [level>1⇒
         [300⊙50]
      (user screenrect center-(smallFrame extent/2))].
   self frame: (self fixframe: smallFrame); show.
   stackPane context: false at: level instance: false code: false;
      interrupt: flag.
   stackPane of: (Top-level) inVector. NotifyFlag ← true]
```

of: titleString stack: stack interrupt: flag | stackPane

```
   [NotifyFlag ← false. CitationReasons ← CitationReasons+1.
   stackPane ← StackPane new.
   self title: titleString with: stackPane inVector at: smallTemplates.
   smallFrame moveto:
      [Top currentPriority>1⇒
         [300⊙50]
      (user screenrect center-(smallFrame extent/2))].
   self frame: (self fixframe: smallFrame); show.
   stackPane context: false instance: false code: false; interrupt: flag.
   stackPane of: stack inVector. NotifyFlag ← true]
```

Window protocol

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 new.
   contextVarPane ← VariablePane new. contextValuePane ← CodePane new.
```

```
    instanceVarPane ← VariablePane new. instanceValuePane ← CodePane new.
    "Create the six-paned window."
    self title: title
        with: (stackPane, codePane, contextVarPane, contextValuePane,
instanceVarPane, instanceValuePane)
        at: bigTemplates.
    self frame: frame; show.
    "Initialize the six panes."
    stackPane 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]
↵
SystemOrganization classify: #NotifyWindow under: 'Windows'.↵
NotifyWindow classInit↵
```

"PageWindow"

```
Class new title: 'PageWindow'
   subclassof: DocumentWindow
   fields: ''
   declare: '';
   asFollows↵
```

I am a document window whose document is a page.

Initialization

```
pattern: pattern named: title
   [self on: (Page new pattern: pattern) named: title]
```

Window Protocol

```
↵
SystemOrganization classify: ↗PageWindow under: 'Windows'.↵
```

"PanedWindow"

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

A paned window is a Window that has subwindows (panes) that are awakened and resized in unison.

Initialization

```
title: title with: panes at: templates | pane
   [self reset. CitationReasons ← CitationReasons+1.
   for: pane from: panes do: [pane init]]
```

Window protocol

```
close | pane
   [for: pane from: panes do: [pane close].
   CitationReasons ← CitationReasons-1]
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
   [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: 1)]]
hardcopy | p
   [p ← dpO pressfile: (self title+'.press') asFileName.
   self hardcopy: p.
   p page; close]
hardcopy: p | pane
   [self showtitle.
   titleframe hardcopy: p.
   for: pane from: panes do: [pane hardcopy: p]]
kbd | pane
   [(pane ← self pickedpane)⇒ [↑pane kbd]]
```

```
keyset | pane
   [(pane ← self pickedpane)⇒ [↑pane keyset]]
leave | pane
   [for: pane from: panes do: [pane windowleave]]
outline
   [frame outline: 1]
pickedpane | pane
   [for: pane from: panes do: [pane picked⇒ [↑pane]]
   frame flash. ↑false]
redbug | pane
   [(pane ← self pickedpane)⇒ [↑pane redbug]]
show | pane
   [super show.
   for: pane from: panes do: [pane outline]]
takeCursor
   [(panes@1) takeCursor]
title
   [↑title]
yellowbug | pane
   [(pane ← self pickedpane)⇒ [↑pane yellowbug]]
```

Pane services

```
vanish
   [self close; erase. user unschedule: self.]
```

Private

```
titlerect
   [↑frame origin - (2 @ (DefaultTextStyle lineheight + 4)) rect: (frame corner
x@frame origin y) + (2@0)]
```

SystemOrganization classify: #PanedWindow under: 'Windows'.

"SyntaxWindow"

```
Class new title: 'SyntaxWindow'
   subclassof: PanedWindow
   fields: ''
   declare: 'stdFrame stdTemplates ';
   asFollows↵
```

I am a paned window with a stack pane and a code pane to report errors during non-interactive compilations, e.g., filin, 's, understands.

Initialization

```
classInit
   [stdTemplates ← (0⊙0 rect: 12⊙36), (12⊙0 rect: 36⊙36).
   stdFrame ← 60⊙320 rect: 570⊙500]
of: errorString at: position in: stream for: class from: context | stackPane
codePane
   [stackPane ← StackPane new.
   codePane ← CodePane new class: class selector: nil para: nil.
   self title: class title with: (stackPane, codePane) at: stdTemplates.
   stdFrame moveto: (user screenrect center-(stdFrame extent/2)).
   self frame: (self fixframe: stdFrame); show.
   stackPane context: false instance: false code: codePane.
   stackPane of: context inVector.
   codePane showing: stream asArray.
   codePane from: stackPane; notify: errorString at: position in: stream]
↵
SystemOrganization classify: ↷SyntaxWindow under: 'Windows'.↵
SyntaxWindow classInit↵
```

"Window"

```
Class new title: 'Window'
   subclassof: Object
   fields: 'frame collapsed titlepara growing exitflag '
   declare: 'titlerun border titleloc titleframe windowmenu ';
   asFollows_!
```

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

Initialization

```
classInit       "Window classInit"
   [border ← 2⊙2.
   titleframe ← Textframe new para: nil frame: nil.
   titleloc ← 4⊙(¯3-titleframe lineheight).
   titlerun ← String new: 2.
   titlerun word: 1 ← 0177401.
   windowmenu ← Menu new string:
'under
frame
close
print
printbits
']
reset
   [exitflag←true. growing←false]
```

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⇒ [self reset.  ↑self enter] ↑false]
lasttime
   [self leave. ↑exitflag]
```

Framing

```
clearTitle: color
   [(titleframe window inset: ¯4⊙¯4) clear: color]
erase
   [(frame inset: ¯2⊙¯2) clear.
```

```
  self clearTitle: background]
fixedwidthfromuser: width | a b oldframe [
  user waitnobug.
  [frame≡nil⇒[] self aboutToFrame; erase].
  a ← OriginCursor showwhile: user waitbug.
  growing ← true.
  self frame: (frame ← self fixframe: (a rect: a+(width⊙32))); show.
  CornerCursor showwhile: [
    while: (a ← user mpnext) do: [ a x ← frame corner x.
      [oldframe≡nil⇒ [user cursorloc ← a max: frame corner]].
      oldframe ← frame copy.
      self frame: (frame ← self fixframe: (frame growto: a));
        moveFrom: oldframe]].
  growing ← false.
  self takeCursor]
fixframe: f [^f]
frame: f
  [frame ← self fixframe: f]
moveFrom: oldframe
  [(oldframe inset: ¯1) clear. self show]
newframe | a oldframe [
  user waitnobug.
  [frame≡nil⇒[] self aboutToFrame; erase].
  a ← OriginCursor showwhile: user waitbug.
  growing ← true.
  self frame: (frame ← self fixframe: (a rect: a+32)); show.
  CornerCursor showwhile: [
    while: (a ← user mpnext) do: [
      [oldframe≡nil⇒ [user cursorloc ← a max: frame corner]].
      oldframe ← frame copy.
      self frame: (frame ← self fixframe: (frame growto: a));
        moveFrom: oldframe]].
  growing ← false.
  self takeCursor]
outline
  ["Clear and outline me."
  frame outline]
show [
  self outline.
  growing⇒[]
  self showtitle]
showtitle
  [titleframe put:
    (Paragraph new text: self title runs: titlerun alignment: 0)
    at: frame origin+titleloc.
  titleframe outline]
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. ↑false];
      =4⇒[self hardcopy];
      =5⇒[self print]]
close []
enter [self show]
hardcopy [frame flash]
kbd [user kbd. frame flash]
keyset [frame flash]
leave []
outside [↑false]
print
   [(dpO pressfile: (self title + '.press.') asFileName)
      screenout: frame scale: PressScale]
redbug
   [frame flash]
yellowbug
   [frame flash]
↵
SystemOrganization classify: ↗Window under: 'Windows'.↵
Window classInit↵
```

"ClassPane"

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

I am a list pane that displays the names of all the classes of a category

Initialization

classInit

```
   [editmenu ← Menu new string: 'filout
print
compress
forget']
from: systemPane to: organizationPane
```

Window protocol

close

```
   [systemPane ← nil. super close]
```

yellowbug

```
   ["If there is a selection, let the user choose a command from the menu."
   selection=0⇒ [window flash]
   editmenu bug
      =1⇒ ["filout" (Smalltalk∘(list∘selection)) filout];
      =2⇒ ["print" (Smalltalk∘(list∘selection)) printout];
      =3⇒ ["compress" (Smalltalk∘(list∘selection)) compressAll];
      =4⇒ ["forget" systemPane forget: list∘selection]]
```

ListPane protocol

deselected

```
   ["I just lost my selection.  Tell organizationPane to display nothing."
   organizationPane class: nil.]
```

selected

```
   ["My selection just changed.  Tell organizationPane to display the categories
of my newly selected Class."
   organizationPane class: Smalltalk∘(list∘selection).]
```

Browser protocol

compile: parag

```
   [systemPane compile: parag]
```

dirty

```
   [↑organizationPane dirty]
```

noCode

```
   [selection=0⇒ [↑systemPane noCode] ↑'']
↵
SystemOrganization classify: ↗ClassPane under: 'Panes and Menus'.↵
ClassPane classInit↵
```

"CodePane"

```
Class new title: 'CodePane'
   subclassof: Window
   fields: 'pared class selector selectorPane scrollBar'
   declare: 'editmenu ';
   asFollows_!
```

I am a Window for editing a paragraph which may include Smalltalk source code. My selectorPane (not necessarily of class SelectorPane, and possibly even myself) compiles and doits for me.

Initialization

```
class: class selector: selector para: para
classInit
   [editmenu ← Menu new string:
      'again
copy
cut
paste
doit
compile
undo
cancel
align']
from: selectorPane
init
showing: paragraph
   [pared ← ParagraphEditor new para: paragraph asParagraph frame: nil.
   pared formerly: false; fixframe: frame.
   self windowenter.
   scrollBar ← ([scrollBar≡nil⇒ [ScrollBar new] scrollBar]) on: frame from: pared]
```

Window protocol

```
close
   [pared unselect. selectorPane ← pared ← nil. scrollBar close]
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]
enter
   [scrollBar show]
frame ← frame
   ["Change my frame and that of my pared (if any)."
   pared≡nil⇒ [] pared frame ← frame.
   scrollBar on: frame from: pared]
```

```
hardcopy: p | win e
   [win ← frame inset: ¯2.
   for: e from: (win minus: frame) do:
      [p showrect: e color: 0].
   p closeEntity.
   p entityorigin: ((e← p transrect: win) minX⊙0)+(800⊙800).
   pared contents presson: p in: e.
   p closeEntity]
kbd
   [pared typing]
keyset
   [↑pared keyset]
leave
   [scrollBar hide]
outline
   [frame outline: 1]
outside
   [↑scrollBar startup]
picked
   [↑frame has: user mp]
redbug
   [pared selecting]
show
   [frame outline. pared show]
windowenter
   [self outline. pared enter]
windowleave
   [pared=nil⇒[] pared leave]
yellowbug
   [editmenu bug
      =1⇒[pared again];
      =2⇒[pared copy];
      =3⇒[pared cut];
      =4⇒[pared paste];
      =5⇒[pared Scrap ← scrollBar hidewhile:
            (selectorPane execute: pared selectionAsStream for: self) asString
asParagraph];
      =6⇒[pared formerly⇒
            [scrollBar hidewhile: [selectorPane compile: pared contents⇒ [pared
formerly: false]]]
         frame flash];
      =7⇒[pared undo];
      =8⇒[pared formerly⇒ [self showing: pared formerly] frame flash];
      =9⇒[pared realign]]
```

Browse/Notify protocol

```
compile: parag      "as my own selectorPane"
   [↑self compile: parag in: class under: 'As yet unclassified']
compile: parag in: defaultClass under: category
   [↑Generator new
      compile: parag asStream
      in: [class=nil⇒ [defaultClass] class]
```

```
        under: category
        notifying: self]
contents
    [↑pared contents]
dirty
    [pared formerly← [↑frame] ↑false]
execute: parseStream for: codePane          "as my own selectorPane"
    [↑self execute: parseStream in: false to: nil]
execute: parseStream in: context to: receiver
    [↑Generator new evaluate: parseStream in: context to: receiver notifying:
self]
formerly: oldpara       "should not be called before 'showing:'"
    [pared formerly: oldpara]
interactive
    [↑true]
notify: errorString at: position in: stream
    [pared
        fintype;
        select: position;
        replace: ('■' + errorString + '■') asParagraph;
        select.
    ↑false]
oldContents
    [↑pared formerly]
reflects: selection  "am I trying to show the code of selectorPane's selection?"
    [↑class=nil and: selection>0]
┘
SystemOrganization classify: #CodePane under: 'Panes and Menus'.┘
CodePane classInit┘
```

"ListPane"

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

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

Initialization

```
of: list "Acquire the specified list and show me scrolled to the top"
   [firstShown ← selection ← 0. para ← nil. self fill; deselected]
revise: newlist with: sel | changing
   ["Acquire newlist. Do not change firstShown. Select sel if in list."
   [changing ← list≠newlist⇒
      [list ← newlist.
      firstShown ← firstShown min: list length.
      para ← nil. self fill]
    selection>0⇒ [changing ← list∘selection≠sel⇒ [self compselection]]
    changing ← true].
   changing⇒ [selection ← ¯1. self select: (list find: sel)]]
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]]
```

Pane protocol

```
close "Zero my selection so it wont be grayed when I close. Break cycles."
   [selection←0. scrollBar close]
eachtime
   [window has: user mp⇒
      [user kbck⇒[↑self kbd]
      user anybug⇒
         [user redbug⇒[↑self redbug]
         user yellowbug⇒[↑self yellowbug]
         user bluebug⇒[↑false]]
      user anykeys⇒[↑self keyset]]
   ↑self outside]
enter
   [scrollBar show]
firsttime
   [window has: user mp⇒[self enter]
   ↑false]
frame ← window "(Re)initialize my window"
```

```
    [para ← nil.
    scrollBar ← ([scrollBar≡nil⇒ [ScrollBar new] scrollBar]) on: window from: self]
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 ← user currentCursor.
    self scrollControl: [user keyset=6⇒[2]; =12⇒[¯2]; =2⇒[1]; =8⇒[¯1] 0].
    c show]
lasttime
    [self leave]
leave
    [scrollBar hide]
outline
    [window outline: 1]
outside [↑scrollBar startup]
picked
    [↑window has: user mp]
redbug | newSel f          "Deselect selection and select cursor item, if any"
    [self compselection. f ← self locked⇒ [f flash. self compselection]
    newSel ← (user mp y - window origin y)/self lineheight + firstShown.
    XeqCursor showwhile: [self select: [newSel = selection⇒ [0] newSel]]]
scrollPos [↑0.0]
scrollTo: ignored
windowenter "Refresh my image.  Reaffirm selection."
    [self outline; fill; select: selection.]
windowleave
    [self compselection; grayselection]
yellowbug
    [window flash]
```

Subclass defaults

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

Private

```
compselection "If I have a selection, complement its image."
    [selection≠0⇒ [self selectionRect comp]]
dummy
    [↑'~~~~~~~~~~~~~~~~']
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: isLen⇒ [(list∘i) printon: s] self dummy copyto: s].
         s cr].
      para ← s contents
      ]].
   self show]
grayselection
   [selection≠0⇒ [self selectionRect color: ltgray mode: oring]]
init
   [self para: nil frame: nil.]
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 | c
```

```
    [c ← window origin x-20.
    self scrollControl:
        [user buttons=4⇒
            [user mp x > c⇒[2] ¯2]
        0]]
selectionRect | h w
    ["I have a selection.  Return its highlighting rectangle."
    (w ← window inset: 2) height ← h ← self lineheight.
    ⌈w + (0⊙(selection-firstShown *h))]
┘
SystemOrganization classify: #ListPane under: 'Panes and Menus'.┘
```

"Menu"

```
Class new title: 'Menu'
   subclassof: Object
   fields: 'str text thisline frame'
   declare: '';
   asFollows
```

I am a list of text lines one of which can be selected with the pointing device

Initialization

```
rescan " | each. Menu allInstances notNil transforms each to each rescan."
   [self string: str]        "rescan (for new fonts, lineheight)"
string: str | i pt tpara
   [[str last≠13⇒[str←str+'
']].       "make sure str ends with CR"
   text ← Textframe new para: (tpara ← str asParagraph)
          frame: (Rectangle new origin: (pt ← 0 ⊙ 0)
                                corner: 1000 ⊙ 1000).
   pt ← text maxx: str length+1.
   text frame growto: pt + (4 ⊙ 0).
   tpara center.
   frame ← text frame inset: ¯2 ⊙ ¯2.
   thisline ← Rectangle new origin: text frame origin
          corner: text frame corner x ⊙ text lineheight]
```

User interactions

```
bug | index bits
   [bits ← self movingsetup.          "set up and save background"
   index ← self bugit.                "get the index"
   frame bitsFromString: bits.        "restore background"
   ↑ index                            "return index"
   ]
clear
   [frame clear]
fbug | index
   [   "for fixed menus"
   index ← self bugit.                "get the index"
   ↑ index                            "return index"
   ]
frame
   [↑ frame]
has: pt
   [↑ text frame has: pt]
moveto: pt
   [self clear.
   frame moveto: pt.
   text frame moveto: pt+2.
   thisline moveto: pt+2.
   ]
rebug
```

```
   [user waitbug.      "wait for button down again"
   ↑"bugcursor showwhile:" self bug]
show
   [frame clear: black. text show.]

Internal
bugit | pt bits
   [user nobug ⇒
      [↑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]

         ↑1 + (thisline origin y-text frame origin y
               / text lineheight)        "return index"
         ]
      thisline comp.                     "he left the menu"
      until: [text frame has: user mp] do:
         [user nobug⇒[↑0]]               "return 0 for abort"
      thisline comp]                     "he came back"
   ]
movingsetup | 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.
   ↑ bits
   ]
┘
SystemOrganization classify: ↗Menu under: 'Panes and Menus'.┘
```

"OrganizationPane"

```
Class new title: 'OrganizationPane'
   subclassof: ListPane
   fields: 'classPane selectorPane class'
   declare: 'editmenu ';
   asFollows_!
```

I am a list pane that displays the selector categories of a class.

Initialization

class: class

```
   [self of: (self listFor: class)]
```

classInit

```
   [editmenu ← Menu new string: 'filout
print']
```

from: classPane to: selectorPane

listFor: class

```
   [↑[class≡nil⇒ [Vector new: 0]
      ↗(ClassDefinition ClassOrganization) concat: class organization
categories]]
```

Window protocol

close

```
   [classPane ← nil. super close]
```

yellowbug

```
   ["If there is a selection, let the user choose a command from the menu."
   selection≤1⇒ [window flash]          "Can't filout or print definition by itself"
   editmenu bug
      =1⇒ ["filout the selected category"
         selection=2⇒ [class filoutOrganization]
         class filoutCategory: list∘selection];
      =2⇒ ["print the selected category"
         selection=2⇒ [window flash]          "Can't print organization"
         class printoutCategory: list∘selection]
   ]
```

ListPane protocol

deselected

```
   ["I just lost my selection.  Tell selectorPane to display nothing."
   selectorPane of: (Vector new: 0)]
```

selected

```
   [selectorPane of: [selection≤2⇒ [Vector new: 0] class organization category:
list∘selection]]
```

Browser protocol

code: selector

```
   [↑class code: selector]
```

compile: parag

```
      | sel cat
   [class≡nil or: selection=1⇒ [classPane compile: parag] "new definition"
```

```
   selection=2⇒ [class organization fromParagraph: parag. self class: class]
"new organization"
   cat ← [selection=0⇒ ['As yet unclassified'] list∘selection].
   sel ← selectorPane compile: parag in: class under: cat⇒
      [self revise: (self listFor: class) with: cat.
      selection≠0⇒ [selectorPane revise: (class organization category: cat) with:
sel]]
   ↑false]
dirty
   [↑selectorPane dirty]
execute: parag
   [↑class's parag]
forget: selector | cat
   [class derstands: selector.
   cat ← list∘selection.
   self revise: (self listFor: class) with: cat.
   selection>0⇒
      [selectorPane revise: (class organization category: cat) with: selector]]
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]']
spawn: selector with: parag formerly: oldparag
   [selectorPane compselection; select: 0.
   class edit: selector para: parag formerly: oldparag]
┘
SystemOrganization classify: ↗OrganizationPane under: 'Panes and
Menus'.┘
OrganizationPane classInit┘
```

"ScrollBar"

```
Class new title: 'ScrollBar'
   subclassof: Object
   fields: 'rect bitstr owner position'
   declare: 'DownCursor UpCursor JumpCursor ';
   asFollows_!
```

I am a bar to the left of an awake window. With the cursor in me I can make that window scroll.

Initialization

classInit

```
   [UpCursor ← Cursor new fromtext: '
1000000000000000
1100000000000000
1110000000000000
1111000000000000
1111100000000000
1111110000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000'.
   DownCursor ← Cursor new fromtext: '
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1111110000000000
1111100000000000
1111000000000000
1110000000000000
1100000000000000
1000000000000000' offset: 0⊙16.
   JumpCursor ← Cursor new fromtext: '
0111000000000000
1111100000000000
1111100000000000
0111000000000000
```

```
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO' offset: 2⊙1]
on: f from: o
   [self on: f from: o at: o scrollPos]
on: frame from: owner at: f
   [rect ← Rectangle new
      origin: frame origin-(32⊙2)
      extent: 32⊙(frame height+4).
   position ← Rectangle new
      origin: rect origin+(9⊙(4+(f*(rect height-16))))
      extent: 16⊙8]
```

Scheduling

```
close
   [owner←nil]
eachtime | p cx r        "This needs to be restructured"
   [rect has: (p← user mp)⇒
      [cx ← rect center x - 2.
      p x < cx⇒
         [r ← Rectangle new origin: rect origin corner: cx⊙rect maxY.
         DownCursor showwhile:
               [while: (r has: (p←user mp)) do:
                  [self slide: p⇒[owner scrollTo: (position minY-rect minY-4)
asFloat/(rect height-12)]
                  user redbug⇒[self reposition:[owner scrollUp: p y - rect corner
y]]]]]
      r ← Rectangle new origin: cx⊙rect minY corner: rect corner.
      UpCursor showwhile:
            [while: (r has: (p←user mp)) do:
               [self slide: p⇒[owner scrollTo: (position minY-rect minY-4)
asFloat/(rect height-12)]
               user redbug⇒[self reposition:[owner scrollUp: p y - rect origin y]]]]]
   ↑false]
firsttime
   [↑rect has: user mp]
lasttime
slide: p | bug
   [position has: p⇒
      [JumpCursor showwhile:
         [bug ← false.
         while: ((position has: user mp) and: bug=false) do:
            [user redbug⇒
```

```
            [bug ← true.
            whileS user redbug doS:
               [self reshowS:
                  [position movetoS: position origin x⊙
                     ((user mp y maxS: rect origin y+4) minS: rect corner y-12)]]]]].
      ↑bug]
   ↑false]
```

Image

```
hide    "restore background"
   [bitstr≡nil⇒ [user notifyS: 'Attempt to hide unshown scrollbar']
   rect bitsFromStringS: bitstr]
hidewhileS: expr  | v
   [self hide. v ← expr eval. self show. ↑v]
repositionS: expr
   [self reshowS:
      [expr eval.
      position movetoS: rect origin+
         (9⊙(4+(owner scrollPos*(rect height-16))))]]
reshowS: expr | r
   [r ← position insetS: ¯2.  expr eval.
   r clearS: white.  position outline]
show    "Save background and turn gray"
   [bitstr ← rect bitsIntoString.
   rect clearS: black.
   (rect insetS: 2⊙2 andS: 1⊙2) clearS: white.
   position outline]
┘
SystemOrganization classifyS: ↶ScrollBar underS: 'Panes and Menus'.┘
ScrollBar classInit┘
```

"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.

Initialization

```
classInit
   [editmenu ← Menu new string:
      'spawn
forget']
from: organizationPane to: codePane
```

Window protocol

```
close
   [organizationPane ← nil. super close]
yellowbug
   [selection=0⇒ [window flash]
   scrollBar hidewhile:
      [editmenu bug
         =1⇒ [organizationPane spawn: list∘selection with: codePane contents
               formerly: codePane oldContents];
         =2⇒ [organizationPane forget: list∘selection]]]
```

ListPane protocol

```
deselected
   [codePane showing: organizationPane noCode]
selected
   [codePane showing: (organizationPane code: list∘selection)]
```

Browser protocol

```
compile: parag
   [↑organizationPane compile: parag]
compile: parag in: class under: heading
   [↑codePane compile: parag in: class under: heading]
dirty
   [↑codePane dirty]
execute: parseStream for: codePane
   [↑codePane execute: parseStream in: false to: nil]
↵
SystemOrganization classify: ↶SelectorPane under: 'Panes and Menus'.↵
SelectorPane classInit↵
```

"StackPane"

```
Class new title: 'StackPane'
   subclassof: ListPane
   fields: 'contextVarPane instanceVarPane codePane variables
proceed'
   declare: 'stackmenu ';
   asFollows_[
```

I am a list pane that displays one or all of the stack below a context in a notify window.

Initialization

```
classInit
   [stackmenu ← Menu new string:
      'stack
spawn
proceed
restart']
context: contextVarPane at: level instance: instanceVarPane code: codePane
   [variables ← (Vector new: 16) asStream.
    proceed≡nil⇒[proceed ← (false, nil, level)]]
context: contextVarPane instance: instanceVarPane code: codePane
   [variables ← (Vector new: 16) asStream.
    proceed≡nil⇒[proceed ← (false, nil, Top currentPriority)]]
interrupt: flag
   [proceed∘1 ← flag]
```

Window protocol

```
close
   [Top enable: proceed∘3. super close. list⇒ [(list∘1) releaseFully]]
yellowbug
   [scrollBar hidewhile:
      [stackmenu bug
         =1⇒ ["show a full backtrace"
               self revise: (list∘1) stack with: [selection=0⇒ [nil] list∘selection]];
         =2⇒ ["spawn a code editor" self spawn];
         =3⇒ ["return to selected context" self continue: false];
         =4⇒ ["restart selected context" self continue: true]]]
```

ListPane protocol

```
deselected
   [contextVarPane ≡ false⇒ []
   codePane showing: ''.
   contextVarPane names: (Vector new: 0) values: ↗(nil) wrt: false.
   instanceVarPane names: (Vector new: 0) values: ↗(nil) wrt: false]
locked
   [ñcontextVarPane and: (selection>0 and: self dirty)]
selected | context instance code safeVec
   [contextVarPane ≡ false⇒ []
   context ← list∘selection. instance ← context receiver. code ← self code.
```

```
    codePane showing: [code⇒ [code] ''].
    variables reset. context variableNamesInto: self with: nil.
    [code⇒
      [contextVarPane names: (↱(thisContext) concat: variables contents)
values: (context, context tempframe) wrt: context.
       context tempframe≡nil⇒ [user notify: 'NIL TEMPFRAME']]
     contextVarPane names: ↱(thisContext) values: context inVector wrt:
context].
    variables reset. instance class fieldNamesInto: self.
    safeVec ← Vector new: 2. safeVec all ← instance.
    instanceVarPane names: (↱(self) concat: variables contents) values: safeVec
wrt: context.
    contextVarPane select: 1]
```

NotifyWindow protocol

```
compile: parseStream | ctxt selector method mcl
    [ctxt ← list∘(selection max: 1). mcl ← ctxt mclass.
     proceed∘2 ← selector ←
          codePane compile: parseStream in: mcl under: 'As yet unclassified'⇒
        [codePane reflects: selection⇒
          [method ← mcl md methodorfalse: selector⇒
            [self releaseAboveSelection.
             ctxt restartWith: method. proceed∘1 ← true.
             self of: list∘(selection to: list length) copy; select: 1]]]]
dirty
    [ⁿcodePane and: codePane dirty]
execute: parseStream for: codePane
    [ⁿproceed∘2 ←
      codePane execute: parseStream in: [selection=0⇒ [false] list∘selection] to:
nil]
```

Private

```
code "the code of my selected context, if it has code, else false"
      | mclass selector
    [mclass ← (list∘selection) mclass. selector ← self selector.
    ⁿ[mclass canunderstand: selector⇒ [mclass code: selector] false]]
comment: s          "called by selected via Class fieldNamesInto"
contents     "called by selected via Class fieldNamesInto"
continue: restarting | ctxt
    ["Close my window and resume my selected context, if any, else my first
context. If interrupted (proceed∘1) or restarting or a recompiled method, don't
return a value; otherwise, return proceed∘2."
    [selection=0⇒ [selection←1]].
    ctxt ← list∘selection.
    self releaseAboveSelection.       "release abandoned contexts"
    [restarting⇒ [ctxt restart]
     proceed∘1 and: selection=1⇒ ["resume after interrupt"]
     ctxt push: proceed∘2].
    list ← false. "Inhibit me closing." user topWindow vanish.
    list ← nil.
    Top run: ctxt at: proceed∘3.
```

```
    Top enable: proceed∘3.
    Top wakeup: proceed∘3.
    Top resetCurrent]
declaration: dummy1 name: string asArg: dummy2
    [variables next ← string]
identifier: s        "called by selected via Class fieldNamesInto"
    [variables next ← s]
notify: msg "selected context doesnt know its variables"
releaseAboveSelection
    [[selection>1⇒ [(list∘(selection-1)) sender ← nil. (list∘1) release"Fully"]].
    (list∘(selection max: 1)) verifyFrames]
selector | context
    [context ← list∘(selection max: 1). ⋂[context sender≡nil⇒ [false] context
sender thisop]]
separator: c     "called by selected via Class fieldNamesInto"
spawn | mclass selector parag oldparag
    [mclass ← (list∘(selection max: 1)) mclass.
    selector ← self selector.
    parag ← [codePane⇒ [codePane contents] mclass canunderstand: selector⇒
[mclass code: selector] ''].
    oldparag ← [codePane⇒ [codePane oldContents] false].
    self compselection; select: 0.
    mclass edit: selector para: parag formerly: oldparag]
terminate "called by parser close during initialization"
trailer: s     "called by selected via Class fieldNamesInto"
┘
SystemOrganization classify: ↱StackPane under: 'Panes and Menus'.┘
StackPane classInit┘
```

"SystemPane"

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

I am a list pane in which all the system categories are displayed.

Initialization

classInit

```
   [sysmenu ← Menu new string: 'filout
print']
```

to: classPane

update

```
   [self of: (↱(AllClasses SystemOrganization) concat: SystemOrganization
categories). mySysOrgVersion←user classNames]
```

Window protocol

enter *"be sure I am up to date"*

```
   [mySysOrgVersion≡user classNames⇒ [super enter]
   window outline. self update. super enter]
```

leave *"I am up to date"*

```
   [mySysOrgVersion ← user classNames. super leave]
```

yellowbug

```
   [selection<3⇒[window flash]
   scrollBar hidewhile:
      [sysmenu bug
         =1⇒
            [SystemOrganization filoutCategory: list∘selection];
         =2⇒
            [SystemOrganization printCategory: list∘selection]
      ]
   ]
```

ListPane protocol

deselected

```
   [classPane of: (Vector new: 0)]
```

selected

```
   [classPane of: self classes]
```

Browser protocol

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 ← nil'sparag.
    class ls: Class⇒
        [className ← class title unique.
        [cat⇒ [SystemOrganization classify: className under: cat]].
        mySysOrgVersion≡user classNames⇒
            [selection>0⇒
                [classPane of: [cat⇒ [SystemOrganization category: cat] user
classNames]]]
        self update]]
dirty
    [↑classPane dirty]
forget: className
    [user notify: 'Class '+className+' will disappear if you proceed...'.
    (Smalltalk∘className) obsolete. Smalltalk delete: className.
    SystemOrganization delete: className.
    AllClassNames ← AllClassNames delete: className.
    classPane revise: self classes with: className]
noCode
    [selection=0⇒ [↑'']; =2⇒ [↑SystemOrganization]
    ↑'Class new title: ''NameOfClass''
    subclassof: Object
    fields: ''names of fields''
    declare: ''names of class variables''' copy]
┘
SystemOrganization classify: ↷SystemPane under: 'Panes and Menus'.┘
SystemPane classInit┘
```

"VariablePane"

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

I am a list pane that displays the names of variables in a context or instance.

Initialization

```
classInit
   [varmenu ← Menu new string: 'inspect']
names: vars values: values wrt: context
   [self of: vars]
to: valuePane
   []
```

Window protocol

```
yellowbug
   [selection=0⇒ [window flash]
   scrollBar hidewhile: [varmenu bug =1⇒ [self value inspect]]]
```

ListPane protocol

```
deselected
   [valuePane showing: '']
selected
   [valuePane showing: self value asString]
```

Notify/Inspect protocol

```
compile: parag
   [window flash. ↑false]
execute: parseStream for: valuePane
   [↑valuePane execute: parseStream in: context to: values∘1]
```

Private

```
value
   [selection=1⇒ [↑values∘1] ↑(values∘2) inspectfield: selection-1]
```

Reclamation

```
_!
SystemOrganization classify: ↶VariablePane under: 'Panes and Menus'._!
VariablePane classInit_!
```

"Directory"

```
Class new title: 'Directory'
   subclassof: Object
   fields: 'dirinst bitinst filinst junkfile closed'
   declare: 'CRR dfmask CCR boffset dirname disksize CWW ';
   asFollows_!
```

I am an Alto OS-compatible file directory

Initialization

```
classInit
   [dfmask ← 02000. "bit meaning active directory entry"
   boffset ← 040. "byte offset of bit table in DiskDescriptor"
   dirname ← 'SysDir.'.
   disksize ← 4872.        "in pages"
   CCR← 044120. CRR← 044100. CWW← 044150.          "disk commands"
   ]
close [closed→[]
   filinst close.
   bitinst close.
   closed ← true]
Close [
   "force a close (and deallocation) without writing files"
   filinst ← bitinst ← closed ← nil]
on: dirinst []
open
   [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]]
reset [
   closed→ [self open]
   filinst reset.
   bitinst≡nil→ ["this can happen during self open"]; flush]
restore [self Close open]
```

Create Files

```
file: fname
   [↑[File new on: self; named: fname]]
newFile: fname | t [
   t ← [File new on: self; new; named: fname]→ [↑t]
   user notify: 'file already exists: ' + fname]
oldFile: fname | t [
   t ← [File new on: self; old; named: fname]→ [↑t]
   user notify: 'no such file: ' + fname]
```

```
pressfile: fname
  [↑PressFile new of: (self file: fname)]

Store/Retrieve Files
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]
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.
  while: (w ← filinst nextword) do:
    [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]
findSN: sn | entry
  [for: entry from: self do:
    [(entry word: 3)=sn⇒[↑(entry↓(14 to: entry↓13+13)) copy]]
  ↑'Not found']
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]
rename: name to: newname | entry nentry
    ["assumes newname already checked to be not there"
    entry ← self delentry: name⇒
        [nentry ← self insert: newname.          "leaves filinst at entry∘1"
        nentry∘(3 to: 12) ← entry∘(3 to: 12).
        filinst append: nentry]
    ↑false]
```

Stream of Files

```
asStream [self reset]
next | w entrylen
    [while: (w ← filinst nextword) do:
        [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]
```

Directory Mapping

```
filesMatching: pattern | f s
    [s ← (Vector new: 10) asStream.
    self for: f matching: pattern do: [s next ← f].
    ↑s contents]
filin: s [
    s is: Vector⇒ [
        for: s from: s do: [self filin: s]]
    (self oldFile: [
        s is: String⇒ [s]
        s asString + '.st.']) filin]
for: var matching: pattern do: expr | pat x entry
        "where the pattern is   *.st   for example, or a list of patterns"
    [pattern is: Vector⇒
        [for: pat from: pattern do:
            [self for: x matching: pat do:
                [var value ← x.  expr eval]]].
    [pattern last≠056⇒ [pattern←pattern+'.']].
    (pattern has: '*'∘1) or: (pattern has: '#'∘1)⇒          "Use String match:"
        [for: entry from: self do:
            [pattern match: entry∘(14 to: entry∘13 + 13)⇒
                [var value ← (entry∘(14 to: entry∘13 + 13)) copy.
                expr eval]]]
    var value ← pattern.  expr eval]
list | entry
```

```
    [for: entry from: self do:
        [user cr; show: entry.(14 to: entry.13 + 13)]]
list: pattern | f     list     " dp0 list: '*.st' "
    [for: f from: (list ← ((self filesMatching: pattern) sort)) do:
        [user cr; show: f]
    ↑ list]
multiplefilin: pattern | f
    [self for: f matching: pattern do:
        [user show: f; cr.  (self file: f) filin]]
multiplenewChars: pattern | f
    [self for: f matching: pattern do:
        [user show: f; cr.  (self file: f) newChars]]
```

BitTable Access

```
alloc: dadr          "allocate a new disk page from bitTable"
        | index start stop ch i m
    [self open.
    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) = 0377⇒ []
            m ← 0200.
            while: m > 0 do: [
                [(ch nomask: m)⇒
                    [ch ← ch lor: m.
                    "check if page is really free"
                    junkfile zaplabel dskprim: (junkfile virtualToAlto: index)
                        command: CCR page: 0 pages: 1] false]⇒[m←¯1]
                index ← index + 1.
                m ← m lshift: ¯1].
            bitinst skip: ¯1; next← ch.          "update bittable"
            m = ¯1⇒ [↑junkfile curadr]]
        stop ← start.        "wrap around to where we started"
        start ← 0].
    user quitThen:
'//   YOUR DISK IS FULL - Please make some space available.
//   Then resume Smalltalk and interrupt or continue as desired...'.
    ↑self alloc: dadr]
dealloc: dadr | index ch m
    [self open.
    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 skip: ¯1; next← ch - m]
        bitinst error: 'page already free (Directory dealloc:)'].
        junkfile dskprim: dadr command: CRR page: 0 pages: 1⇒[
            dadr ← junkfile nextp.
```

```
        junkfile zaplabel docommand: CWW];
      error: 'could not read page'].
  bitinst flush]
flushBits [bitinst flush]
free | npages tpages ch i [
  self open.
  npages ← 0.
  bitinst settopage: 1 char: boffset.
  for: i from: 1 to: disksize by: 8 do: [
    (ch ← bitinst next)
      =0⇒ [];
      =0377⇒ [npages ← npages+8]
    until: ch = 0 do: [
      npages ← npages + (ch land: 1).
      ch ← ch lshift: ¯1]]
  ↑disksize - npages]
getNewSn | sn [
  bitinst settopage: 1 char: 010.
  sn ← bitinst next: 4.
  sn word: 2 ← (sn word: 2) + 1.
  [(sn word: 2) = 0⇒ [sn word: 1 ← (sn word: 1) + 1]].
  bitinst skip: ¯4; append: sn; flush.
  ↑sn]
```

Grow Smalltalk

```
growSmalltalkBy: n | zfpt i f
  [zfpt ← CoreLocs new base: (Vmem specialLocs↓7) length: 128.
  i← 1. until: [zfpt↓(i+64)=0] do: [i← i+1].          "find last entry in file page
table"
  f ← junkfile copy.
  f zaplabel dskprim: (f virtualToAlto: (zfpt↓(i+63))+(zfpt↓i)-(zfpt↓(i-1))-1)
    command: CRR page: 0 pages: 1⇒          "just sort of go there..."
      [↑f extendby: n];
  error: 'Growing Smalltalk']
```

Disk ID

```
username | f u [
  "return O.S. user name and disk name"
  f ← self file: 'sys.boot'.
  f readonly settopage: 2 char: 0.
  u ← f next: f next.
  [u length even⇒ [f next]].
  ↑u, (f next: f next)]
```

```
SystemOrganization classify: ↶Directory under: 'Files'.
Directory classInit
```

"File"

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

Provides Stream-like access to the disk.
See Stream for most read/write operations

Initialization

```
classInit
   ["status"
      old← 1. oldornew← 2. new← 3. created← 4.

   "disk commands (Check/Read/Write  header-label-data)"
      CRR← 044100. CRW← 044110.
      CCR← 044120. CCW← 044130. CWW← 044150.

   "label fields (as in label∘nextp)"
      nextp← 1. backp← 2. lnused← 3. numch← 4.
      pagen← 5. version← 6. sn1← 7. sn2← 8.

   "rwmode"
      read← 1. write← 2. shorten← 4.]
named: filename [
   filename ← self namecheck: filename⇒ [↑self find]
   ↑false]
new [status ← new]
old [status ← old]
oldornew [status ← oldornew]
on: dirinst []
printon: strm
   [strm append: 'File ' + filename asString]
random [rvec ← Set new vector: 0]
readonly [rwmode ← read]
readwrite [rwmode ← read + write]
reopen [
   [array=nil⇒ [self of: (String new: self pagelength)]].
   (self dskprim: leader command: CCR page: 0 pages: 1) and:
   filename - (array∘(14 to: array∘13 + 13)) = 0⇒ [
      "the correct file"
      ↑self old init]
   "leader page failed or name failed"
   [rvec=nil⇒ [] self random].
   ↑self oldornew find]
writeonly [rwmode ← write]
```

Global file operatons

```
close [self close: false]
close: s [
   ["shorten file on demand or if flag is on"
   s or: (rwmode allmask: shorten)⇒ [self shortento: labelopagen char: position]
   self flush].
   "reopen will reallocate"
   array ← nil]
contents | n [
   n ← self settoend position.
   ↑self reset next: n]
delete
   [rwmode allmask: write⇒
      [↑dirinst delete: filename]
   ↑false]
inMode: m do: expr | oldmode [
   "mainly useful for changing between reading/writing"
   oldmode ← rwmode.
   rwmode ← m.
   expr eval.
   rwmode ← oldmode]
makerandom | p old [
   "generate a set of disk addresses"
   old ← rvec.
   rvec← nil.
   p ← self settoend page.
   "check if old address are still valid (same last page)"
   [old≡nil⇒ [false]
   old length = p and: oldop = curadr]⇒ [rvec ← old]

   rvec ← Set new of: (Vector new: p + 10 "extra") to: p.
   self inMode: read do: [
      for: p from: p to: 1 by: ¯1 do: [
         self settopage: p char: 0.
         rvecop ← curadr]].
   "end up at page 1"]
newChars | i
   "converts this file from old to new (5.0) character set IN PLACE.
   Do not use more than once, as a strange character mapping would result"
   [self reset.
   while: labelonumch=self pagelength do:
      [for: i to: array length do: [arrayoi← Xlateo(arrayoi+1)].
      self settopage: labelopagen+1 char: 0].
   for: i to: array length do: [arrayoi← Xlateo(arrayoi+1)].
   self reset]
overlay [user overlay: self fileid]
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.
```

```
   filename ← newname.
   self settopage: 0 char: 12.
   self next ← filename length.
   self append: filename]
sclose [self close: true]
```

Test and alter position

```
char [↑position]
end [
   position < (label∘numch)⇒ [↑false]
   ↑label∘nextp = 0]
page [↑label∘pagen]
pagelength [↑512]
pageposition [↑self position/self pagelength]
position [↑ self page - 1 * self pagelength + position]
position← p [self settopage: 1 char: p]
reset
   [[self page=1⇒ [self flush]].
   self settopage: 1 char: 0]
settoend [
   self flush;
      inMode: read do: [self settopage: 30000 char: 0]]
settopage: page char: char | pch pn pagelength [
   pagelength ← self pagelength.
   page ← (char/pagelength + page) asSmall.
   pch ← (char abs \ pagelength) asSmall.
   [char < 0 and: pch ≠ 0⇒[
      pch ← pagelength - pch. page ← page-1]].
   page < 0⇒ [self error: 'negative page']

   "what to do with current page"
   [label∘pagen < page and: label∘nextp = 0⇒ [self extendto: page]
   label∘pagen = page⇒ []
   self flush].

   [rvec ≡ nil⇒ []
   label∘pagen = page⇒ ["already there"]
   "try random access"
   pn ← page min: rvec length.
   pn = 0 or: rvec∘pn ≡ nil⇒ []
   self dskprim: rvec∘pn
      command: CCR page: pn pages: page+1-pn⇒ [];
    error: 'random access'].

   "do sequential reads to get there, possibly extending file"
   [(pn ← label∘pagen) < page⇒ [
      "chained read forward"
      [label∘nextp ≠ 0⇒ [
         self dskprim: label∘nextp
            command: CCR page: pn+1
            pages: page-pn⇒ [];
          error: 'forward chaining']].
```

```
      labelopagen = page⇒ ["finished"]
      rwmode allmask: write⇒ [self extendto: page]
      "change the request"
      page ← labelopagen. pch ← labelonumch];
    › page⇒
      [pn-1 = page⇒
        ["one page backwards"
        self dskprim: labelobackp
          command: CCR page: page pages: 1⇒ [];
         error: 'reading backwards']
      "forward from leader"
      self dskprim: leader
        command: CCR page: 0 pages: page+1⇒ [];
       error: 'leader reading']].

  labelopagen ≠ page⇒ [self error: 'didnt get to page']
  limit ← labelonumch.
  position ← [rwmode allmask: write⇒ [pch] limit min: pch]]
skip: n [
  [(n ← position + n) ‹ 0⇒ []; › limit]⇒ [self settopage: labelopagen char: n]
  position ← n]
```

Grow/Shrink

```
extendby: n [↑self settoend extendto: labelopagen+n]
extendto: dest | p n old
      "Maintains valid eof during grow"
  [rwmode nomask: write⇒ [↑false]
  n ← labelopagen.
  whileꜱ n<dest doꜱ
    [p←dirinst alloc: curadr.
    old← labelobackp.
    labelobackp← curadr.
    curadr← p.
    labelonextp← labelonumch← 0.
    labelopagen ← n+1.
    self docommand: CWW.          "stamp the new page"
    labelonextp ← curadr.
    curadr← labelobackp.
    labelobackp← old.
    labelonumch← self pagelength.
    labelopagen← n.
    self docommand: CWW.          "link it on"
    labelobackp← curadr.
    curadr← labelonextp.
    labelopagen← n← n+1.
    rvec ≡ nil⇒[]
    rvec next ← curadr]
  labelonextp← labelonumch← position← 0.
  dirinst flushBits]
pastend [
  self end⇒ [↑false]
  self settopage: labelopagen+1 char: 0.
```

```
    ↑self next]
pastend ← x [
    rwmode nomask: write⇒ [self error: 'no writing on read only files']
    [limit < self pagelength⇒ [limit ← self pagelength]
    self settopage: labelopagen+1 char: 0].
    ↑self next ← x]
shorten [self shortento: labelopagen char: position]
shortento: page char: char [
    rwmode nomask: write⇒ [↑false]
    self flush; settopage: page char: char.
    dirinst dealloc: labelonextp.
    labelonextp ← 0.
    labelonumch ← labelonumch min: position.
    self docommand: CWW.
    rvec ≡ nil⇒ []
    rvec position ← page]
```

Current Page

```
curadr [↑curadr]
fileid | v [
    v ← Vector new: 5.
    vo1 ← labelosn1.
    vo2 ← labelosn2.
    vo3 ← labeloversion.
    vo4 ← 0.
    vo5 ← curadr.
    ↑v]
flush [
    rwmode nomask: write⇒ [↑false]
    labelonumch ≤ position⇒ [
        position ≠ self pagelength⇒ [
            labelonumch ← (limit ← position).
            self docommand: CWW]
        labelonextp = 0⇒ [self extendto: labelopagen + 1]
        self docommand: CWW]
    self docommand: CCW]
nextp [↑labelonextp]
```

Filin/out

```
asParagraphPrinter        "Defines the default output format for filout, etc."
    [↑BravoPrinter init of: self]
filin | s [
    user cr.
    self end⇒ [user notify: 'No data']
    [self filinCompatible⇒ []
    user notify: filename+' is an old filout. Proceed if newChars has been run
(once).'].
    self readonly.
    until: self end do: [
        FilinSource ← self.
        s ← self upto: 036.
```

```
      user print: nil'ss; space].
   FilinSource ← nil]
filinCompatible      "trivial test for now;  assumes file non-empty"
   [↑self peek=047]
filout
   [user cr.
   user displayoffwhile: [self filout: Changes contents sort]]
filout: source
   [self asParagraphPrinter stamp; printchanges: source; close]
filoutclass: class
   [self asParagraphPrinter stamp; printclass: class; close]
printout: source
   [(PressPrinter init of: self) stamp; printchanges: source; close]
```

Address conversions

```
altoToKeys: dadr | i k [
   k ← Stream default.
   for: i to: 14 do: [
      dadr allmask: (0100000 lshift: 1-i)⇒ [
         k next←'546E7DUVOK-P/\'oi]]
   ↑k contents]
altoToVirtual: dadr [
   ↑(dadr lshift: ¯12) + (3 * (dadr land: 07774))]
virtualToAlto: vadr [
   ↑(vadr \ 12 lshift: 12) + (vadr/12 * 4)]
```

File edit

```
compile: parag       "rewrites file with contents of edit window"
   [user displayoffwhile: [self reopen; append: parag; sclose]]
edit
   [user restartup: (CodeWindow new file: self)]
execute: parag for: codePane          "doit in file edit"
   [↑codePane execute: parag for: codePane]
title [↑filename]
```

Private/Internal operations

```
copy | f l
   [l ← label. label ← l copy.
   f ← super copy. label ← l.
   ↑f]
docommand: com [
   self dskprim: curadr
      command: com
      page: label∘pagen
      pages: 1⇒ [];
    error: 'docommand']
dskprim: curadr
   command: command
   page: startpage
   pages: npages
   [↑false] primitive: 80
```

```
error: s | t [
   [status=0⇒ []
   status≡nil⇒ []
   "see Alto hardware manual for details on status word format"
   user cr show: 'current sector'; space print: (status bits: (0 to: 3)).
   [(status land: 3)
      =1⇒ [user cr show: 'hardware error or sector overflow'];
      =2⇒ [user cr show: 'check error'];
      =3⇒ [user cr show: 'disk command specified illegal sector']].
   for: t to: 6 do: [
      status allmask: (0200 lshift: 1-t)⇒ [
         user cr show: ↗(
            'seek failed, possible illegal track'
            'seek in progress'
            'disk unit not ready'
            'hardware late'
            'hardware not transferring'
            'checksum')∘t]].
   user cr show: 'label'.
   for: t to: 8 do: [user space show: (label∘t) base8]].
   ↑user notify: 'in file '+ filename +', ' + s]
find | entry [
   [status ≡ nil⇒ [self oldornew]].
   [rwmode ≡ nil⇒ [self readwrite]].
   [dirinst ≡ nil⇒ [dirinst← dp0]].
   [array ≡ nil⇒ [self of: (String new: self pagelength)]].
   (label ← Vector new: 8) all← 0.
   label∘numch ← self pagelength.
   entry ← dirinst find: filename⇒
      [status = new⇒ [↑false]
      label∘sn1 ← entry word: 2.
      label∘sn2 ← entry word: 3.
      label∘version ← entry word: 4.
      self dskprim: (leader ← self virtualToAlto: (entry word: 6))
         command: CCR
         page: 0
         pages: 1⇒[self init]
      self error: 'leader page']

   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.
   entry word: 6 ← self altoToVirtual: leader.
   dirinst next ← entry.
   self init]
init | oldmode s [
   position ← 0.    "assumes positioned at page 0"
```

```
    s ← user timewords.

    oldmode ← rwmode.
    self inMode: write do: [
        [status = created⇒["creation date" self append: s] self skip: 4].
        [oldmode allmask: write⇒["write date" self append: s] self skip: 4].
        self append: s.      "read date"
        [status = created⇒
            [self next ← filename length;        "file name (Bcpl style)"
                append: filename]].
    "see <altodocs>altofilesys.d, (p.3 leader page) for further stuff"

        self reset.
        rvec ≡ nil⇒[]
        "create random access vector"
        self makerandom]]
namecheck: name
    [↑self namecheck: name fixing: false]
namecheck: name fixing: fixing | i x copy
    [name length = 0⇒
        [fixing⇒ [↑'$.']
        self error: 'empty name']
    name length > 191⇒
        [fixing⇒ [↑self namecheck: (name∘(1 to: 191)) copy fixing: true]
        self error: 'name too long']
    copy ← name.
    for: i to: name length do:
        ["check characters"
        x ← name∘i.
        x isletter⇒ []
        x isdigit⇒ []
        0≠('+-$!?.' find: x)⇒ []
        fixing⇒ [[copy≡name⇒ [copy ← name copy]]. copy∘i ← '-'∘1]
        self error: 'illegal character ' + (name∘(i to: i))].
    x ≠ ('.'∘1)⇒ [↑copy + '.']
    ↑copy]
zaplabel [
    labelonextp ← labelopagen ← 0.
    labelosn1 ← labelosn2 ← labeloversion ← ¯1]
```

Misc

```
editBravo   "full page width column"
    [self editBravo: (6.5*72) asInteger]
editBravo2   "half page width column"
    [self editBravo: (3*72) asInteger]
editBravo: width | g w   "width is in points (72 per inch)"
    [g ← user displayoffwhile: [Galley new readBravo: self width: width].
    w ← GalleyWindow init on: g named: filename.
    user restartup: w]
```

Aspects

```
dirinst [↑dirinst]
name [↑filename]
↵
SystemOrganization classify: ↗File under: 'Files'.↵
File classInit↵
```

"Events"

"EventQueue"

```
Class new title: 'EventQueue'
   subclassof: Queue
   fields: 'primitivequeue readwriteelapsed time'
   declare: 'write read elapsed ';
   asFollows_!
```

The EventQueue (Events) is a subclass of class Queue. It contains a regular queue which is filled from a block of memory (currently 128 words long), updated during the 60HZ interrupt. This block of memory is called the primitivequeue, and is unpacked into the regular queue when events are available upon receiving the message peek or next. The fundamental difference between peek and next is that next dequeues the current event and peek does not. Furthermore peek will return false when the queue is empty, and next, when the queue is empty, will create a time elapsed event and return that. An event will be of class UserEvent as created by the primitiveDequeue and next messages. The machine code thinks of events as a 4-word structure as follows:

Word 1:
Left Byte:
1 = Key down. 0 = Key up.
Right Byte:
8-bit Ascii value.
Mouse Buttons: Left/Top = 0
Middle = 1
Right/Bottom = 2
Keyset: Leftmost = 3
= 4
= 5
= 6
Rightmost = 7
Values 8 - 255, keyboard decodings as expected by rawkbd.

Word 2: Time (sixtieths of a second since last event).

Word 3: Cursor x coordinate (UserView htab accounted for).

Word 4: Cursor y coordinate (UserView vtab accounted for)..

Initialization

```
init                "sets up system wide event queue -- only one from the
present"
   ["***BEWARE, USED ONLY AT TIME OF SYSTEM GENERATION***"
   primitivequeue ← CoreLocs new base: (mem∘0114) length: (mem∘0115).
   readwriteelapsed ← CoreLocs new base: 0111 length: 3.
   read ← 0. write ← 1. elapsed ← 2.      "offsets of read, write and elapsed
pointers"
   time ← 0.                          "start time at 0"
   super of: (Vector new: 4).          "initialize Smalltalk queue"
   ]
```

Public Access

```
elapsedtime [↑readwriteelapsed∘elapsed] "return elapsed time"
next | event elapsedtime "return event from queue unless both queues empty,
when return null event"
   [event ← self dequeue ⇒ [↑event]                    "Queue not empty?"
   event ← self primitiveDequeue ⇒ [↑event]            "primitivequeue not empty?"
   elapsedtime ← readwriteelapsed∘elapsed.
   ↑UserEvent new                                      "when empty return null event"
      x:          user x          "event x"
      y:          user y          "event y"
      type:       0               "2=up, 1=down, 0=null, only time passed"
      stroke:     0               "0 stroke for null event"
      elapsed:    elapsedtime     "1-32767 sixtieths of a sec -- since last event"
      time:       (time + elapsedtime) asSmall        "time since timer reset"
   ]
peek | event "unless both queues empty, return event from queue, but dont
dequeue"
   [event ← super peek ⇒[ ↑event]
   event ← self primitiveDequeue ⇒ [↑self next ← event]
   ↑false    ]
reset "for the present, just reset time to 0"
   [time ← 0]
time [↑time]          "return time"
time ← time           "reset time"
```

Private

```
primitiveDequeue | rp tands nrp event elapsedtime
      "unless empty, return event from primitivequeue"
      [(readwriteelapsed∘read) = (readwriteelapsed∘write) ⇒                [↑false].
"primitivequeue empty?"
      "Build event from primitive queue and return it."
         "first word has type and stroke packed"
      tands ← primitivequeue∘(rp ← readwriteelapsed∘read).
      elapsedtime ← primitivequeue∘(rp + 1).
      event ← UserEvent new
         x:          primitivequeue∘(rp + 2)          "event x"
         y:          primitivequeue∘(rp + 3)          "event y"
         type:       [tands < 0 ⇒ [2] 1]              "2=up, 1=down"
         stroke:     [tands > 0 ⇒ [tands] 0 - tands]  "1-336"
         elapsed:    elapsedtime                      "1-32767 sixtieths of a second"
         time:       (time ← (time + elapsedtime) asSmall).

      nrp ← rp + 4.                                   "set bumped read pointer"
      [nrp ≥ (primitivequeue length) ⇒ [nrp ← 1]].    "Wrap-around?"
      readwriteelapsed∘read ← nrp.                    "bump read pointer"
      ↑event                                          "Return event"
   ]
┘
SystemOrganization classify: ↶EventQueue under: 'Events'.┘
```

"MessageTally"

```
Class new title: 'MessageTally'
   subclassof: Object
   fields: 'class method tally rcvrs'
   declare: 'timer ';
   asFollows↵
```

The following statement analyzes the evaluation of 'user restore'. It checks every 10 sixtieths of a second to see what method is being executed. It prints the analysis on file 'restore.spy'.

```
   spy every: 10; ons [user restore]; report: 'restore.spy'; close.
```

Read further to learn of more flexible ways to use message tallies.

A message tally is a tally of how many times, according to some authority, a certain method or any of that method's callees has been invoked. Message tallies for the callees are listed in the vector, rcvrs; thus, each message tally is a node in a tree. The root of that tree is called the 'root tally'; its method the 'root method'; and the context that was running that method the 'root context'. Contexts that do not have the root context on their stack are tallied as if the root context were at the bottom of their stack.

The authority informs the root tally of invocations in either of two ways: 'explicitly' or by 'spying'.

To explicitly create the tallies from a root context, rc, a root tally is created with:

```
   mt ← MessageTally new from: rc
```

and is informed of the invocation of each context c with:

```
   mt tally: c
```

To spy on (periodically sample) the execution of a statement sequence, ss, every t sixtieths of a second (t>1), a root tally is created with:

```
   mt ← MessageTally new every: t
```

and is informed of invocations with:

```
   valOfSS ← mt ons [ss].
```

The context that executes the latter statement is the root context. Its method should not be called recursively by ss. Only one spying operation can be in progress at a time, and all spies share the most recently specified time interval. Thus, there may as well be only one spying message tally in existence, and the global variable 'spy' is predefined as such.

Spying typically adds 1/4 second per probe to execution. Ctrl-shift-esc can be used to abort a spying operation.

Tallies can be printed by:

```
   mt report: 'filename.spy'
```

which prints two tables of invocations sorted by tally, with tallies expressed as percentages and with methods described in terms of their defining class and selector. In the first table, 'Leaves', tallies do not include time spent in submethods (this is like the spy in Swat). In the second table, 'Tree', the percentages do include time spent in submethods, and those submethods are

displayed indented. In both tables, entries below a 'cutoff' of 2 per cent are suppressed.

Different cutoffs can be specified for each table. To cut off Leaves at 7.5 per cent and Tree at 3 per cent, use:
mt report: 'filename.spy' cutoff: 7.5,3
A cutoff of 100 or greater suppresses printing of that table completely. To output to a specified stream, use the message 'fullprinton:cutoff:'.

To release the storage occupied by the tally tree, use the message 'close'.

Public Tallying

```
abort
    [timer is: Timer⇒ [timer disable]]
classInit
    [Smalltalk define: ↗spy as: (MessageTally new every: 10)]
close "release storage"
    [class ← method ← tally ← rcvrs ← nil]
every: sixtieths        "Create a spy that samples with the specified period"
    [self abort. timer ← Timer new for: sixtieths actions [self tally: Top▫1. timer
reset]]
from: context  "Create a tallier from the specified root"
    [self class: context receiver class method: context method]
ons remote | val  "Spy on the specified evaluation"
    [self from: remote. timer reset. val ← remote eval. timer disable. ↑val]
tally: context | root  "Explicitly tally the specified context and its stack"
    [context method≡method⇒ [↑self bump]
    (root ← context sender)≡nil⇒[↑self bump tallyPath: context]
    ↑(self tally: root) tallyPath: context]
```

Public Reporting

```
fullprinton: s cutoff: pct | set mt i t
    [user displayoffwhile:
        [s print: tally; append: ' tallies'; cr. tally=0⇒ []
        s cr; cr. [pct is: Vector⇒ [] pct ← pct, pct].
        [pct▫1<100⇒
            [s append: '**Leaves**'; cr. t ← ((pct▫1)*(tally-1)/100) asInteger.
            set ← HashSet new init: 128. self leaves: set.
            self cumprinton: s from: set total: tally over: t. s next←12; cr.
            set ← nil]].
        [pct▫2<100⇒
            [s append: '**Tree**'; cr. t ← ((pct▫2)*(tally-1)/100) asInteger.
            self treeprinton: s tab: 0 total: tally over: t. s next←12; cr]].
        s skip: ¯2]]
printon: s
    [class≡nil⇒ [super printon: s] self printon: s total: 100]
report: filename
    [self report: filename cutoff: 2]
report: filename cutoff: pct | f  "pct=(leaves,roots,tree) or one number for all"
    [f ← dpo file: filename. f append: filename; space.
    self fullprinton: f cutoff: pct. f shorten; close]
```

Private Tallying

bump
```
[tally ← tally+1]
```
tallyPath: context | m path mt c
```
[m ← context method. path←false.
for: mt from: rcvrs do: [mt method≡m⇒ [path←mt]].
[path≡false⇒
   [path ← MessageTally new class: context receiver class method: m. rcvrs ←
rcvrs, path]].
↑path bump]
```

Private Reporting

< mt
```
[↑tally > mt tally]
```
= mt
```
[↑mt method≡method]
```
> mt
```
[↑tally < mt tally]
```
breakdown | n b mt
```
[b ← rcvrs. b≡nil or: b length=0⇒ [↑↑()]
n ← tally. for: mt from: b do: [n ← n - mt tally].
[n>0⇒ [b ← b, [MessageTally new class: class method: method; primitives:
n]]].
↑b]
```
bump: n
```
[tally ← tally+n]
```
cumprinton: s from: set total: total over: threshold | mt
```
[for: mt from: set contents sort do:
   [mt tally>threshold⇒ [mt printon: s total: total. s cr] ↑self]]
```
hash
```
[↑method asOop]
```
into: set | mt i
```
[[i ← set find: self⇒ [mt ← set objects↓i]
set insert: (mt ← MessageTally new class: class method: method)].
mt bump: tally]
```
leaves: ldict | b mt
```
[b ← self breakdown. b length=0⇒ [self into: ldict] for: mt from: b do: [mt
leaves: ldict]]
```
primitives: tally
```
[rcvrs ← nil]
```
printon: s total: total | i v
```
[v ← (0.0+tally/total*1000.0+0.5) asInteger asString. i ← v length.
s append: '  '↓(i to: 2); append: v↓(1 to: i-1); append: '.'; next←v↓i; space.
rcvrs≡nil⇒ [s append: 'primitives']
class describe: method on: s]
```
tally
```
[↑tally]
```
treeprinton: s tab: tab total: total over: threshold | i mt
```
[tally≤threshold⇒ []
[tab>0⇒ [for: i to: tab-1 do: [s append: '  |']. self printon: s total: total. s cr]].
```

```
    for: mt from: self breakdown sort do:
        [mt treeprinton: s tab: tab+1 total: total over: threshold]]
```

Private Common

```
class: class method: method
    [tally ← 0. rcvrs ← Vector new: 0]
method
    [↑method]
┘
SystemOrganization classify: ↶MessageTally under: 'Events'.┘
MessageTally classInit┘
```

"PriorityInterrupt"

```
Class new title: 'PriorityInterrupt'
   subclassof: Object
   fields: 'scheduler priority'
   declare: '';
   asFollows↵
```

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

Initialization
from: scheduler at: priority

Level numbers
+ arg
[↑priority + arg]

Scheduling
deepsleep
[scheduler deepsleep: priority]
disable
[scheduler disable: priority]
enable
[scheduler enable: 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]

↵
SystemOrganization classify: ↗PriorityInterrupt under: 'Events'.↵

```
                                                    "PriorityScheduler"
Class new title: 'PriorityScheduler'
   subclassof: Object
   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: 'TimeInt CtlCDisp UserInt GRODSK CtlShftEscInt 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.

Initialization

```
o priority
   [↑suspendedContexts○priority]
currentPriority [↑currentPriority]
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.]
replaceUser: stack
   [UserInt run: stack]
```

Install and Terminate

```
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
  ["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]
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]
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]
```

Enable and Disable

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]
```

Wakeup and Sleep

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]
```

errorReset

```
    ["There has been an error. Initialize the state of the process that was
running.
     If it was not the user process (priority 1), request it to cease running and
     prevent its further running (i.e. disable it)"
    currentPriority=1⇒[self init: currentPriority]
    awakePriorities ← awakePriorities land: bitoff∘currentPriority.
    enabledPriorities ← enabledPriorities land: bitoff∘currentPriority.
    self init: currentPriority]
```

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]
resetCurrent
   ["Initialize the state of the process that is running. If it is not the
     user process (priority 1), request it to cease running"
    currentPriority=1⇒[self init: currentPriority]
    awakePriorities ← awakePriorities land: bitoff∘currentPriority.
    self init: currentPriority]
restart: priority
   ["Initialize the state of a suspended process and request it to run"
    interruptedPriorities ← interruptedPriorities lor: biton∘priority.
    self init: priority]
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]
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]
```

Top level

```
init1
   [UserInt ← Top
      install: [user restart]
      at: 1.
    UserInt enable wakeup]
init11    " Top terminate: 11; init11. "
   [GRODSK ← Top
      install:
         [user displayoffwhile:
            [user show: '
Smalltalk needs more space.
Just a moment...'.
            dp0 growSmalltalkBy: 100.
            user show: '  Done.'; cr].
         GRODSK deepsleep]
      at: 11.
   GRODSK enable]
init3
   [TimeInt ← Top
      install: ["user show: 'Ten minutes have passed with no activity.'; cr."
         TimeInt sleep]
      at: 3.
```

```
    TimeInt enable]
init8 | nw
  [CtlCInt ← Top
    install: 
      [nw ← user notifier: 'Control c Interrupt' level: 1 interrupt: true.
      [nw⇒
        [user schedule: nw.
         nw takeCursor.
        nw ← nil.
         UserInt restart]].
      CtlCInt sleep]
    at: 8.
   CtlCInt enable]
init9
  [CtlShftEscInt ← Top
    install: [spy abort. user restoredisplay. UserInt restart. CtlShftEscInt sleep]
    at: 9.
   CtlShftEscInt enable]
initsched
  [Top ← self fromSource: (PriorityInterrupt new).
   self init1.
   self init3.
   self init11.
   self init8.
   self init9.
   Top top]
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
```

Critical sections

```
critical: expr| t v
  ["Execute <expr> without allowing it to be interrupted"
   t ← self disable.
   v ← expr eval.
   enabledPriorities ← t.
   self reselect. ↑v]
```

Private

```
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
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 max: 1.
    (newPriority = currentPriority)
      and: (currentPriority = priority)⇒
        [enabledPriorities ← tempenabled.
         sourceIndirect run: (initialContexts∘priority) cleancopy]
    suspendedContexts∘priority ← (initialContexts∘priority) cleancopy.
    newPriority = currentPriority⇒[enabledPriorities ← tempenabled.]
    newContext ← suspendedContexts∘newPriority.
    suspendedContexts∘newPriority ← nil.
    oldPriority ← currentPriority.
    currentPriority ← newPriority.
    enabledPriorities ← tempenabled.
    oldPriority = priority⇒
      [sourceIndirect run: newContext]
    sourceIndirect
      swap: newContext
      with: (suspendedContexts ref: oldPriority)]
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]
printon: s | i b2 j
  [super printon: s.
  for: i to: 5 do:
    [s cr.
    b2← (usedPriorities, enabledPriorities, awakePriorities,
          interruptedPriorities, (1 lshift: currentPriority-1))∘i base: 2.
    for: j to: 16-b2 length do: [s next← '0'∘1].
    s append: b2; space;
      append: ↗('used' 'enabled' 'awake' 'interrupted' 'current')∘i]
  ]
```

Reclamation

```
SystemOrganization classify: ↗PriorityScheduler under: 'Events'.
```

```
                                                        "Timer"
Class new title: 'Timer'
   subclassof: Object
   fields: '    activeTime     "how long this Timer will be the active one"
          nextTimer     "the Timer which will fire after this one"
          lastTimer     "the Timer which will fire before this one"
          delay         "how long between setting and firing"
          action        "what happens when this timer fires"'
   declare: 'currentTimer timerActions ';
   asFollows_!
```

A Timer is an object which causes an action after an interval of time. The time interval is measured in units of a sixtieth of a second from when the instance was initialized with the message <for: {time interval} action: [{code for action}]>. When the interval is over the Timer fires by placing the action on a queue to be evaluated before processing at the user level continues. There is no need to mantain a name for a Timer while it is active, but a named timer may be disabled or reused. The Timers waiting to fire form a doubly linked list whose first link is referred to by the class variable currentTimer. Each Timer knows how long it should run after the preceding Timer has fired

Initialization

```
classInit    "Initialize the processes used by the Timers"
   [timerActions ← Queue new of: (Vector new: 4).
    self init16.
    self init12]
for: delay action: action
   ["Initialize a new Timer"]
init12 | nextAction      "Initialize the process which evals Timer actions"
   [Top install:
      [while: true do:
         [while: (nextAction ← timerActions next) do:
            [nextAction eval].
          Top sleep: 12]] at: 12.
    Top enable: 12]
init16    "Initialize the process wakened by a Timer timing out"
   [Top install:
      [while: true do:
         [currentTimer fire.
          Top sleep: 16]] at: 16.
    Top enable: 16]
reset | nextTimeout foundit
   ["Set up this Timer to add <action> to the Queue of remote Contexts to be
evaled after an interval of <delay> sixtieths of a second. Find the proper place in
the doubly linked list and calculate the amount of time to run after the
preceeding timer fires"
    Top critical:
      [[activeTime=nil]:[self disable].
       activeTime ← delay. nextTimer ← currentTimer. lastTimer ← nil.
       foundit ← false.
```

```
    until: foundit do:
      [nextTimer=nil⊃[foundit ← true].
       (nextTimeout ← nextTimer activetime) > activeTime⊃[foundit ← true].
       activeTime ← activeTime - nextTimeout.
       lastTimer ← nextTimer.
       nextTimer ← lastTimer nexttimer].
    [nextTimer=nil⊃[] nextTimer insertlast: self].
    lastTimer=nil⊃[self startup] lastTimer insertnext: self]]
```

List Behavior

deletelast

```
  ["Delete the Timer before this one. When deleting a Timer, the activeTime of
the Timer after it must be increased by its activeTime"
   activeTime ← activeTime + lastTimer activetime.
   (lastTimer ← lastTimer lasttimer)=nil⊃[self startup]]
```

deletenext

```
  ["Delete the Timer after this one"
   nextTimer ← nextTimer nexttimer]
```

insertlast: lastTimer

```
  ["Insert a new Timer before this one. When inserting a Timer in front of
another, the activeTime of the later one must be reduced so it is the amount of
time after the new Timers firing"
   activeTime ← self activetime - lastTimer activetime]
```

insertnext: nextTimer

lasttimer

```
  [↑lastTimer]
```

nexttimer

```
  [↑nextTimer]
```

Timing Behavior

activetime

```
  ["If this is the current Timer return the time until it fires, otherwise return
activeTime"
   ↑activeTime] primitive: 96
```

disable *"Remove this timer from the list"*

```
  [Top critical:
    [[lastTimer=nil⊃
      [nextTimer=nil⊃
        [self shutoff. Top deepsleep: 16]]
     lastTimer deletenext].
     nextTimer=nil⊃[activeTime ← nil]
     nextTimer deletelast.
     activeTime ← nil]]
```

fire *"Time is up, add the action to the Queue to be evaled"*

```
  [timerActions next← action.
   Top wakeup: 12.
   activeTime ← nil.
   nextTimer=nil⊃[self shutoff]
   nextTimer startup]
```

primstartup

"this message informs the virtual machine that this is the next Timer to

```
fire"
    [] primitive: 95
shutoff
    ["this message informs the virtual machine and class Timer that there are
no more Timers to fire"
     currentTimer ← nil] primitive: 97
startup
    ["make this the next Timer to fire"
     lastTimer ← nil.
     currentTimer ← self.
     Top deepsleep: 16.
     self primstartup]
↵
SystemOrganization classify: ↱Timer under: 'Events'.↵
Timer classInit↵
```

"UserEvent"

```
Class new title: 'UserEvent'
   subclassof: Point
   fields: 'type stroke elapsed time'
   declare: '';
   asFollows↵
```

This class is used by the Events queue (updated in the 60HZ interrupt routine) to package up and return an event every time the queue is popped. The class provides easy access to various parts of the event. Users may create their own events by pushing onto the Events queue, which is why the Initialization here is classified as private.

Initialization

x: x y: y type: type stroke: stroke elapsed: elapsed time: time
"make an event, usually called from EventQueue"

Public Access

elapsed *"return an event stroke"*
["1 - 32767 sixtieths of second since previous non-time-elapsed event recorded"
↑elapsed]
isKbdDown
["if stroke a down stroke and not keyset or mouse button, return it, otherwise return false"
type ≠ 1 ⇒ [↑false] ↑stroke > 16]
stroke *"return an event stroke"*
["0-2 = top,middle,bottom mouse buttons, 3-7 = keyset left to right, 8-255 = keyboard"
↑stroke]
time *"return an event stroke"*
["1 - 32767 sixtieths of second since Events time reset"
↑time]
type *"return event type"*
["2 = upstroke event, 1 = downstroke event, 0 = time-elapsed event"
↑type]
↵

```
SystemOrganization classify: ↗UserEvent under: 'Events'.↵
```

"BitBlt"

```
Class new title: 'BitBlt'
   subclassof: Object
   fields: 'function color destbase destraster destx desty
         width height sourcebase sourceraster sourcex sourcey
         sstrike dstrike '
   declare: 'pageOneCursor ';
   asFollows↵
```

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

Access to Parts

color [^color]
color ← color
destbase [^destbase]
destbase ← destbase
destraster [^destraster]
destraster ← destraster
destx [^destx]
destx ← destx
desty [^desty]
desty ← desty
dest ← pt [destx← pt x. desty ← pt y]
dstrike [^dstrike]
dstrike ← dstrike
extent ← pt [width ← pt x. height ← pt y]
function [^function]
function ← function
height [^height]
height ← height
sourcebase [^sourcebase]
sourcebase ← sourcebase
sourceraster [^sourceraster]
sourceraster ← sourceraster
sourcex [^sourcex]
sourcex ← sourcex
sourcey [^sourcey]
sourcey ← sourcey
source ← pt [sourcex← pt x. sourcey ← pt y]
sstrike [^sstrike]
sstrike ← sstrike
width [^width]
width ← width

Setup

classInit

```
   [pageOneCursor ← 0431. "location of hardware cursor"
   ]
forCursor
   [function← color← 0.
   destbase← sourcebase← 0431.
   width← height← 16. destraster← sourceraster← 1.
   destx← desty← sourcex← sourcey← 0. sstrike ← dstrike ← false
   ]
init
   [function ← color ← destbase ← destraster ← destx ← desty ← width ←
   height ← sourcebase ← sourceraster ← sourcex ← sourcey ← 0. sstrike ←
dstrike ← false
   ]
```

Operations

```
callBLT
   [user croak] primitive: 33
checksandcall | destlocked sourcelocked
   ["checks if either base a string and/or strike and locks accordingly"
      function ← function land: 017.          "set up function to add XM stuff"
   [destbase class ≡ String ⇒
      [destlocked ← destbase.
         destbase ← ([dstrike ⇒ [((destlocked lock) + 9)]
                                          destlocked lock      "strike header"])
      ]
   [destbase ≠ pageOneCursor ⇒
   [function ← function + 020. "dest not string, then must be in display"]
      "unless doing cursor work in page one location"]
   ].
   [sourcebase class ≡ String ⇒
      [sourcelocked ← sourcebase.
         sourcebase ← ([sstrike ⇒ [((sourcelocked lock) + 9)]
                                          sourcelocked lock      "strike header"])
      ]
   [sourcebase ≠ pageOneCursor ⇒
   [function ← function + 040. "source not string, then must be in display"]
      "unless doing cursor work in page one location"]
   ].
   self callBLT.
   [destlocked ≡ nil ⇒ [] destbase ← destlocked unlock].
   [sourcelocked ≡ nil ⇒ [] sourcebase ← sourcelocked unlock].
   ]
copy: mode
   [function ← mode land: 3. self checksandcall]
copycomp: mode
   [function ← 4 + (mode land: 3). self checksandcall]
fill: mode color: color
   [function ← 12 + (mode land: 3). self checksandcall]
paint: mode
   [function ← 8 + (mode land: 3). self checksandcall]
↵
SystemOrganization classify: #BitBlt under: 'Primitive Access'.↵
```

BitBlt classInit

```
                                                        "CoreLocs"
Class new title: 'CoreLocs'
   subclassof: Array
   fields: 'base length'
   declare: '';
   asFollows↵

I am an array mapping a section of Alto memory

Initialization
base: base length: length

Reading and Writing
o x [x isLarge⇒[↑self o (x asSmall)] user croak] primitive: 42
o x ← val [x isLarge⇒[↑self o (x asSmall) ← val] user croak] primitive: 43
base [↑base]
length [↑length]
↵
SystemOrganization classify: ↶CoreLocs under: 'Primitive Access'.↵
```

```
                                                        "VirtualMemory"
Class new title: 'VirtualMemory'
   subclassof: Object
   fields: 'map "Pclass Map"
      premap "block before map"
      FirstContextOop "oop of FirstContext"
      SpecialOopsOop "oop of SpecialOops"
      zip  "Zone Index of Pages"
      prezip  "block before zip"
      zadd  "file info"
      vmemfile "file for vmem" '
   declare: 'ZVPN ZJMP zend zfused zfree zjmpt pmint num00 pmend
ZN zflen pmatm zlong PCL CPT BSC ISC ZHB RCI PIN ';
   asFollows_!

A model of the OOZE virtual memory

Pclass Map
CPT: oop | poma
   [poma ← 2*(oop field: PCL) +1.
   ↑mapopoma field: CPT]
freelist: object  | a
   ["one-order index into freelists."
   object class is: Class ⇒[↑1]
   a ← self ISC: object asOop.
   a < 024 ⇒[↑1+a] ↑1 + a-013]
freelistOffset: len "offset of freelist for this length in a variable length class"
   [↑Class instsize + [len < 9 ⇒[len] (len-1) log2 + 6]]
highpm0: oop gets: val | poma
   [premapopmatm ← premapopmatm min: (oop field: PIN ← 0).
   poma ← 2*(oop field: PCL) +1.
   mapopoma ← val]
highPM: oop | pcl a
   [pcl ← oop field: PCL.
   a ← premapopmatm field: PCL.
   pcl < a ⇒[↑false]
   ↑mapo(2*pcl +1)]
ISC: oop | poma
   [poma ← 2*(oop field: PCL) +1.
   ↑mapopoma field: ISC]
lowPM: oop | pcl a
   [pcl ← oop field: PCL.
   a ← premapopmend field: PCL.
   pcl ≥ a ⇒[↑false]
   ↑mapo(2*pcl +1)]
newHighPM← pm0 | pcl a
   [a ← premapopmatm.
   premapopmatm ← (a ← a-0200).
   pcl ← a field: PCL.
```

```
    mapo(2*pcl +1) ← pm0.
    ↑a]
newLowPM← pm0 | poma a
    [mapo1 = pm0 ⇒["reserved pclasses for Class"
        mapo3 = 0 ⇒[mapo3 ← pm0. ↑0200]
        mapo5 = 0 ⇒[mapo5 ← pm0. ↑0400] ]
    "normal case"
    a ← premapopmend.
    premapopmend ← a+0200.
    poma ← (a field: PCL) *2 +1.
    mapopoma ≠ 0 ⇒["skip over already filled"
        ↑self newLowPM← pm0]
    mapopoma ← pm0.
    ↑a]
newZN: oop
    [self ZN: oop gets: self newZone]
pclassesOf: cls | i clsoop n z p "find pclasses of this class (in order of ZHB)"
    ["ignore small integers and nil"
    p ← (Vector new: 1) asStream.
    z ← (Vector new: 1) asStream.
    clsoop ← cls asOop.
    for: i from: (1 to: 2*(0174000 field: PCL) by: 2) do:
        [mapoi = 0 ⇒[]
        (mapoi field: RCl) = clsoop ⇒[p next← i/2. z next← (mapo(i+1) field:
ZHB)]].
    p ← p contents. z ← z contents.
    n ← p copy.
    for: i to: z length do:
        [no(zoi +1) ← poi].
    ↑n]
pclassesOf: cls length: len | i clsoop n z p isc pm0 "find pclasses of this class
and length (in order of ZHB)"
    ["ignore small integers and nil"
    p ← (Vector new: 1) asStream.
    z ← (Vector new: 1) asStream.
    isc ← [len < 9 ⇒[len] (len-1) log2 + 17].
    clsoop ← cls asOop.
    for: i from: (1 to: 2*(0174000 field: PCL) by: 2) do:
        [(pm0 ← mapoi) = 0 ⇒[]
        (pm0 field: RCl) ≠ clsoop ⇒[]
        (pm0 field: ISC) = isc ⇒[p next← i/2. z next← (mapo(i+1) field: ZHB)]].
    p ← p contents. z ← z contents.
    n ← p copy.
    for: i to: z length do:
        [no(zoi +1) ← poi].
    ↑n]
pmatm [↑premapopmatm]
ZHB: oop | poma
    [poma ← 2*(oop field: PCL) +1.
    ↑mapo(1+poma) field: ZHB]
ZHB: oop gets: val | poma i
    [poma ← 2*(oop field: PCL) +1.
```

```
    i ← mapo(1+poma).
    mapo(1+poma) ← (i field: ZHB ← val)]
ZN: oop | poma
    [poma ← 2*(oop field: PCL) +1.
    ↑mapo(1+poma) field: ZN]
ZN: oop gets: val | poma i
    [poma ← 2*(oop field: PCL) +1.
    i ← mapo(1+poma).
    mapo(1+poma) ← (i field: ZN ← val)]
```

Writing and Reading

```
obwiz: oop | poma pm0 pm1 isc bsc i len zz pin zhb
    ["find (zn, zp, zw, dlen)"
    zz ← Vector new: 4.
    poma ← (oop field: PCL)*2 +1.
    (pm0 ← mapopoma) = 0 ⇒ [user notify: 'bad oop']
    pm1 ← mapo(1+poma).
    isc ← pm0 field: ISC.
    bsc ← pm0 field: BSC.
    zhb ← pm1 field: ZHB.
    [isc < 024 ⇒[len ← ((isc max: 1)+ 1-bsc) / (2-bsc)]
        len ← 8. i ← isc+bsc.
        until: 024 = i do: [i ← i-1. len ← len*2]
        len ← len+1 "length word" ].
    zz04 ← len ← len+1. "dlen with refct"
    "w ← (dlen*128*zhb) +dlen*pin"
    pin ← oop field: PIN.
    zz02 "zp" ← (zhb/2 *len) +(len/256 *pin).
    zz03 "zw" ← (zhb\2 *128 *len) +(len\256 *pin).
    zhb\2 * len ≥ 512 ⇒[user notify: 'address overflow']
    zz02 "zp" ← zz02 + (zz03 field: 136 "highbyte").
    zz03 "zw" ← zz03 field: 128 "lowbyte".
    zz01 ← pm1 field: ZN.
    ↑zz]
readin: oop | zinfo
    [zinfo ← self obwiz: oop.
    ↑self read: zinfo04 at: zinfo]
systemNoHistory [] primitive: 82
writeout: oop with: image | zinfo
    [zinfo ← self obwiz: oop.
    [zinfo04 ≥ (image length /2) ⇒[]
        user notify: 'bad image length'].
    self write: image at: zinfo
    ]
```

Init

```
AComment
    ["VirtualMemory models the Pclass Map of Ooze.
    Vmem is one with CoreLocs on real system.
        (VirtualMemory new) thisvmem.
    Pmap is new one we are building.
```

```
        (VirtualMemory new) fakevmem.
    map is 1-order addressing.
    In init, set pmatm by searching PM for
    highest 0. (No LMAT)
    map RCI field is old class oop. only convert to
        new with mapPM at end.
    Atoms may not cover more than half of the
        address space. (highpm0:gets:)
    (VirtualMemory new) giveBirth.
    Zone Index of Pages. works for both vector zip
    and corelocs of real zip.
    zip, zadd are one-order.
    zfree is 1-order in fake, core addr in this.
    zlong = 512.
    zend is not used (0 fake, core addr in this)
    "]
fakevmem "build another vmem on dp1"
    ["this instance is a model of a new vmem"
    self init.
    map ← Vector new: 1024.
    map all ← 0.
    premap ← (0176000, 0174000, 0, 0).
        "num00, pmint, pmatm, pmend"
    prezip ← (3, 01244, 0). "zfree, zlong, zend"
    zip ← Vector new: prezip◦zlong.
    zip all ← 0.
    zadd ← (1, 1). "zfused, zflen"
    dp1 restore.
    vmemfile ← dp1 file: 'small.boot'.
    vmemfile settopage: 1800 char: 0.
    vmemfile makerandom.
    Smalltalk define: ↗Pmap as: self]
init "common to both real and fake vmem"
    ["field descriptors and indicies in tables"
    num00 ← 1. pmint ← 2. pmatm ← 3. pmend ← 4.
    RCI ← 151. CPT ← 22. BSC ← 21. ISC ← 80.
    ZHB ← 136. ZN ← 128.
    PCL ← 151. PIN ← 112.
    ZJMP ← 76. ZVPN ← 16*12.
    zfree ← 1. zlong ← 2. zend ← 3.
    zfused ← 1. zflen ← 2.
    zjmpt ← (603, 570, 471, 410, 353, 300, 253,
        210, 169, 132, 101, 72, 49, 30, 13)]
premap [↑premap] "array of limits in Pclass Map"
thisvmem | c m "create Vmem, a VirtualMemory viewing this very system"
    [c ← self specialLocs.
    map ← CoreLocs new base: c◦2 -1 length: 1025.
    premap ← CoreLocs new base: c◦2 -5 length: 5.
    Smalltalk define: ↗Vmem as: self.
    self init.
    m ← 1023.
    until: (map◦m = 0) do: [m ← m-2].
```

```
premap←pmatm ← (0 field: PCL ← m/2 +1).
prezip ← CoreLocs new base: co3 -4 length: 4.
zip ← CoreLocs new base: co3 -1 length: prezip←zlong+1.
zadd ← CoreLocs new base: co12 -3 length: 3.
(vmemfile ← File new) on: dp0; readonly; named: 'small.boot'.
FirstContext ← Context new sender: nil
   receiver: user mclass: nil
   method: (m ← UserView md method: ↗restart)
   tempframe: (Vector new: (mo3 max: mo4)) pc: mo6
   stackptr: mo5 -1.
SpecialOops←1 ← Object md method: ↗error.
FirstContextOop ← FirstContext asOop.
SpecialOopsOop ← SpecialOops asOop]
```

Vmem Writing

afterBirth

```
   [MethodKeeperKeeper ← nil]
```

giveBirth | pm0 vm *"initialize map for early pclasses"*

```
   [(VirtualMemory new) thisvmem.  "define Pmap, Vmem"
   (VirtualMemory new) fakevmem.
   self preBirth.
   "small Integer, 16 pclasses, oops 0174000-0177777"
      pm0 ← Vmem highPM: 0176000. "size, class info for Integer"
      until: (Pmap newHighPM← pm0) = 0174000 do: [].
   "first UniqueString"
      (vm ← Vmapper new) object: (0173600 asObject); touchoop.
   "Class oops 0-0177"
      vm object: Class; touchoop.
   "more Class oops 0200-0577"
      Pmap newLowPM← 0; newLowPM← 0.
   "VariableLengthClass oops 0600-0777"
      vm object: Vector; touchoop.
   "skip over some oops"
      until: (Pmap newLowPM← 0) = 01600 do: [].
   "Object oops 02000-02177, false=02000"
      (vm object: false) map ≠ 02000 ⇒[user notify: 'bad false oop']
   "set pointer back to oops skipped over"
      Pmap premap ← pmend ← 01000
   ]
```

giveBirth2 | vm

```
   [vm ← Vmapper new.
   vm object: (Smalltalk ref: ↗FirstContext).
   FirstContextOop ← (vm readin word: 2). "new oop of FirstContext"
   vm reset; object: (Smalltalk ref: ↗SpecialOops).
   SpecialOopsOop ← (vm readin word: 2). "new oop of SpecialOops"]
```

giveBirth3 | vm *"create a new Small.boot on DP1"*

```
   ["Write out all objects rooted at Smalltalk.
   Do not write the stack (rooted in R76, the current context).
         Pmap file close. Pmap ← nil.
         (VirtualMemory new) giveBirth3. user quit.
   *** Above is the commmand to start a seven hour long Vmem write ***
   *** Beware:  Is DP1 on?  Do you want to use old UniqueStrings? *** "
```

```
    Flushed← false.
    (Vmapper new) init; UseOldUniqueStrings: true.
    user show: 'init '; displayoffwhile: [self giveBirth].
    vm ← Vmapper new.
    user show: 'run '; displayoffwhile: [vm object: Smalltalk; map].
    user show: 'USTable '; displayoffwhile: [vm arefsRectify].
    user show: 'freelist '; displayoffwhile: [vm writefreelists].
    user show: 'mrefs '; displayoffwhile: [vm mrefsRectify].
    user show: 'PM '; displayoffwhile: [Pmap mapPM; giveBirth2].
    "These work as many times as needed on dp0 or dp1"
    user show: 'surgery '; displayoffwhile: [Pmap surgery: (dp1 file:
'Smalltalk.run')].
    user show: 'ram '; displayoffwhile: [Pmap ramwrite: (dp1 file: 'byterp.mb')].
    Pmap file flush]
mapPM | c i v rci ctrans  "convert to new RCI"
    [ctrans ← (Vmapper new) classtrans.
    for: c from: (1 to: 1023 by: 2) do:
        [(i ← mapoc) = 0 ⇒[]
        rci ← i field: RCI.
        (v ← ctrans lookup: rci) ⇒
            [mapoc ← (i field: RCI ← v01)]
        user notify: 'class not mapped']
    ]
preBirth | v
    [MethodKeeperKeeper ← MessageDict new.
    MethodKeeperKeeper init: (v ← MethodKeeper contents) length *2;
holdMethods: v]
ramwrite: source | s a i sour "write 8 pages of ram image into small.boot"
    [s ← source title.
    [source dirinst ≡ dp0 ⇒[]
        sour ← dp0 file: s. sour append: source.
        sour close "copy onto dp0"].
    s ← s o(1 to: s length-4) concat: '.br.'.
    a ← Stream default. a append: 'packmu '; append: source title.
    a space; append: s; append: '; resume '; append: Vmem file title.
    user quitThen: a contents.  "go out to OS, packram, and come back"
    s ← dp0 file: s.
    s skip: 021+0400 *2. "skip .br stuff and constants"
    vmemfile settopage: 0400 char: 0; readwrite.
    a ← String new: 512. "now write in ram image"
    for: i to: 8 do: [s into: a. vmemfile append: a].
    s close. vmemfile flush.
    self ≡ Vmem ⇒[vmemfile readonly]]
```

Surgery

```
dynaLocs: guide | l locs keys v i "Surgery - add core location and value pairs to
guide dictionary to test after flush"
    [l ← self specialLocs.
    locs ← (premap base +pmend, (lo11 -1), (lo12 -2), (lo4 +1)).
    keys ← (↷premaptest, ↷purgetest, ↷zaddtest, ↷loxtest).
    for: i from: 1 to: keys length do:
        [v ← locsoi, (memo(locsoi)).
```

```
      guide insert: keysoi with: v].
   (guide lookup: ↗loxtest)o2 ← ¯1]
runLocs: f | guide g i v locs keys offsets "Surgery - create dictionary of
locations of key system tables in Smalltalk.run"
   [i ← f title. v ← io(1 to: i length-5) concat: '.syms.'.
   (g ← File new) on: f dirinst; readonly; named: v.
   locs ← self symsFind: ('PMBASE', 'ZIP', 'ZADD', 'INITIALIZE') on: g.
   keys ← (↗maprun, ↗ziprun, ↗zaddrun, ↗initrun).
   offsets ← (¯1, ¯1, ¯3, ¯3). "words offset"
   guide ← Dictionary new init: 16.
   for: i from: 1 to: locs length do:
      [v ← (locsoi lshift: ¯8), ((locsoi land: 0377) -19 *2). "page, byte"
      "magic fudge factor of 19 words for .run file"
      vo2 ← vo2 +(offsetsoi *2). "bytes offset"
      guide insert: keysoi with: v].
   v ← (guide lookup: ↗maprun) copy.
   vo2 ← vo2 +(¯4 *2). "offset ¯5 total"
   guide insert: ↗premaprun with: v.
   v ← (guide lookup: ↗ziprun) copy.
   vo2 ← vo2 +(¯3 *2). "offset ¯4 total"
   guide insert: ↗preziprun with: v.
   ↑guide]
specialLocs [] primitive: 84
staticTest: guide on: f | l locs offsets values names i v "Surgery - test known
static constants in thisvmem and in smalltalk.run"
   [offsets ← (pmint, 2, 0, 0).
   values ← (0174000, 01244, 0100, 031415).
   names ← (↗premaprun, ↗preziprun, ↗zaddrun, ↗initrun).
   for: i to: names length do:
      [v ← guide lookup: namesoi.
      f settopage: vo1 char: vo2 +(offsetsoi *2).
      f nextword ≠ (valuesoi)⇒
      [user notify: 'bad .run loc']].
   self ≡ Vmem ⇒
      [l ← self specialLocs.
      offsetso4 ← 0. valueso4 ← 014764.
      locs ← (premap base, prezip base, zadd base, (lo11) "bacpur").
      for: i from: 1 to: locs length do:
         [valuesoi ≠ (memo(locsoi +(offsetsoi)))⇒
         [user notify: 'bad system loc']]]
   ]
surgery: f | guide Etemp a i v keys "write virtual memory tables into
Smalltalk.run using Smalltalk.syms"
   ["Vmem ramwrite: (dp0 file: 'byterp.mb').
      Vmem surgery: (dp0 file: 'Smalltalk.run'). "
   [self ≡ Vmem ⇒[self thisvmem.
      Flushed ← true. Etemp ← Events. a ← (0,0,0,0,0)]].
   guide ← self runLocs: f.
   self staticTest: guide on: f.
   [self ≡ Vmem ⇒[user clearshow: 'Wait for the safe light to flash, then hit
any key'.
      until: user kbck do: []. user kbd.
```

```
      self dynaLocs: guide.
      user clearshow: 'Dont forget user systemStartup_] in the new system to
enable interrupts and get the display set up. '.
]].
   self tablewrt: guide on: f. "write in Smalltalk.run"
   f close.
   [self ≡ Vmem ⊃
      [keys ← (↗premaptest, ↗purgetest, ↗zaddtest, ↗loxtest).
      for: i to: keys length do:
         [v ← guide lookup: keys:i.
         v:2 ≠ (mem:(v:1))⊃[user notify: 'please surgery again']].
      Events ← nil. self systemVmemOut.
      Events ← Etemp. user overlay: a]]
   ]
symsFind: strs on: f1 | f2 S N L offset val str j i "find initial values of SRELs in
smalltalk.syms"
   ["S is start of string space (words)
   N is start of symbol table "
   (f2 ← File new) on: f1 dirinst; readonly; named: f1 title.
   f1 settopage: 1 char: 4.
   S ← f1 nextword.
   N ← f1 nextword.
   f1 settopage: 1 char: N*2.
   L ← f1 nextword.
   for: i to: L do:
      [offset ← f1 nextword. "next offset"
      f1 skip: 4.
      val ← f1 nextword.
      f2 settopage: 1 char: S+offset *2.
      str ← String new: f2 next.
      f2 into: str.
      for: j to: strs length do:
         [(strs:j) class ≡ String ⊃
            [strs:j = str ⊃[strs:j ← val]]].
      ].
   for: j to: strs length do:
      [(strs:j) class ≡ String ⊃
         [user notify: 'symbol not found']].
   ↑strs]
systemVmemOut [] primitive: 83
tablewrt: guide on: f | i v "copy tables from self to .run file"
   [v ← guide lookup: ↗premaprun.
   f settopage: v:1 char: v:2 +(2*pmatm).
   f nextword← premap:pmatm.
   f nextword← premap:pmend.
   for: i to: 1024 do: [f nextword← map:i].
   v ← guide lookup: ↗preziprun.
   f settopage: v:1 char: v:2 +(2*1 "zfree").
   i ← [self ≡ Vmem ⊃ [¯1-zip base "relative indexing"]¯1 "make 0-order"] .
   f nextword← prezip :1 +i. "zfree"
   v ← guide lookup: ↗zaddrun.
   f settopage: v:1 char: v:2 +(2*1 "zfused").
```

```
    f nextword← zadd o1. " zfused "
    v ← guide lookup: ↱ziprun.
    f settopage: vo1 char: vo2 +(2*1).
    for: i to: prezipozlong do: [f nextword← zipoi]
    v ← guide lookup: ↱initrun.
    f settopage: vo1 char: vo2 +(2*1 "SpecialOops").
    f nextword← SpecialOopsOop.
    f nextword← FirstContextOop ]
```

Zone Pages

```
cover: zz | pp zrp lpir zjmp vpn olpir
    [pp ← zzo2 + 1. "zp 1-order"
    lpir ← 1. olpir ← 0. zrp ← zzo1 *2 +1.
    until: lpir ≥ pp do: [
        0 = (zjmp ← zipozrp field: ZJMP) ⇒
            [self run: [lpir ≥ 32 ⇒[32] lpir +1] after: zrp.
            self cover: zz]
        zrp ← (zjmptozjmp + zrp -1)\ (prezipozlong) +1.
        olpir ← lpir.
        lpir ← lpir + [lpir ≥ 32 ⇒[32] lpir +1] ].
    vpn ← (zipozrp field: ZVPN) +pp -(olpir +1).
    0 = vpn ⇒[user notify: 'pclass has no zone']
    vpn ← vpn + 0407.
    vmemfile settopage: vpn char: zzo3 *2]
file [↑vmemfile]
getpages: i | vpn
    [[zaddozflen < (zaddozfused + i) ⇒
        [zaddozflen ← zaddozflen + (i max: 20).
        vmemfile settopage: zaddozflen + 0407 char: 0]].
    vpn ← zaddozfused.
    zaddozfused ← vpn+i.
    ↑vpn]
newZone | zrp
    [zip is: Vector ⇒[
    zrp ← prezipozfree.
    until: zipozrp = 0 do: [zrp > (prezipozlong) ⇒
        [user notify: 'zip overflow']
        zrp ← zrp+2].
    prezipozfree ← 2+zrp.
    zipozrp ← self getpages: 1.
    ↑zrp/2]]
read: len at: zz | i str
    [i ← 256 - (zzo3) "zw" min: len.
    self cover: zz.
    str ← String new: 2*i.
    vmemfile into: str.
    i=len ⇒[↑str]
    zzo2 ← zzo2 +1. "zp"
    zzo3 ← 0. "zw"
    ↑str concat: (self read: len-i at: zz)]
run: n after: zrp | i j
    [for: i to: 15 do:
```

```
      [j ← zjmptoi +zrp -1 \ (prezipozlong) +1.
      zipoj = 0 ⇒
         [zipozrp ← zipozrp field: ZJMP ← i.
         zipoj ← self getpages: n. ↑zrp]
      ].
   user notify: 'new zip is full']
write: str at: zz | i len
   [len ← str length /2.
   i ← 256 - (zzo3) "zw" min: len.
   i = len ⇒[
      self cover: zz.
      vmemfile append: str]
   self write: str from: 1 at: zz]
write: str from: pos at: zz | i len
   [len ← str length +1 -pos. "bytes yet to write"
   i ← len min: (256 - (zzo3) "zw")*2. "bytes in page"
   self cover: zz.
   vmemfile append: stro(pos to: i+pos-1).
   i=len ⇒[]
   zzo2 ← zzo2 +1. "zp"
   zzo3 ← 0. "zw"
   self write: str from: i+pos at: zz]
┘
SystemOrganization classify: ↱VirtualMemory under: 'Primitive Access'.┘
```

"Vmapper"

```
Class new title: 'Vmapper'
   subclassof: Object
   fields: 'object oop noop image'
   declare: 'classtrans PCL mapqueue stopcount USTableOop mrefs
arefs count UseOldUniqueStrings ';
   asFollows↓
```

This class has not yet been commented

```
Mapping
map | i
   [(i ← mrefs lookup: oop) ⇒[io1 ← io1 -1. ↑io2]
   [user kbck ⇒ [user kbd; ev] self bump].
   object class = Integer ⇒[↑self mapInteger];
   = String ⇒[↑self mapString];
   = UniqueString ⇒[↑self mapUniqueString];
   = Vector ⇒[↑self mapVector];
   = Float ⇒[↑self mapFloat];
   = MessageDict ⇒[↑self mapMdict]
   object is: HashSet ⇒[↑self mapHashSet]
   object class = Class ⇒[↑self mapClass];
   = VariableLengthClass ⇒[↑self mapClass];
   = Vmapper ⇒[↑↑1]; "avoid recursion in writing"
   = VirtualMemory ⇒[↑self mapVmem]; "avoid recursion in writing"
   "put any fixed octave class here"
   = Object ⇒[↑self mapObject]
   ↑self mapNormal]
mapClass | nlen oadd i refs vm j
   [nlen ← 1 + object class instsize. "refct + normal fields"
   i ← [object is: Class ⇒[oadd ← 0. 1] oadd ← 1. 20].
   noop ← self newoop.
   [(vm ← classtrans lookup: oop) ⇒[vmo1 ← noop]
      vm ← Vector new: i+1. vmo1 ← noop.
      classtrans insert: oop with: vm].
   nlen ← nlen + i + oadd.
   image ← String new: 2*nlen.
   [(refs ← object refct) = 2 ⇒["exactly one reference"]
      mrefs insert: oop with: (refs-2, noop)].
   image word: 1 ← refs -2.
   [oadd=1 ⇒[image word: 2 ← nlen-2]].
   for: j from: (2+oadd to: nlen-i) do:
      [vm ← Vmapper new object: (object instfield: j-1-oadd).
      image word: j ← vm map].
   self writeout.
   ↑noop]
mapFloat | refs i
   [image ← String new: 8.
   noop ← self newoop.
   [(refs ← object refct) = 2 ⇒["exactly one reference"]
      mrefs insert: oop with: (refs-2, noop)].
```

```
    image word: 1 ← refs-2.
    for: i to: 3 do: [image word: (i+1) ← object instfield: i].
    self writeout.
    ↑noop]
mapHashObs | nlen oadd i refs vm trans ob
    [nlen ← object length. "in words"
    oadd ← [nlen≤8⇒[0]1].
    nlen ← oadd +1 +nlen.
    image ← String new: 2*nlen.
    [(refs ← object refct) = 2 "exactly one ref" ⇒
        [noop ← self newoop]
        (i ← mrefs lookup: oop) ⇒[io1 ← io1 -1. noop ← io2]
        noop ← self newoop.
        mrefs insert: oop with: (refs-2, noop)].
    image word: 1 ← refs -2.
    [oadd=1⇒[image word: 2 ← object length]].
    ob ← (trans ← self permHashSet) objects. "translation dictionary"
    for: i from: (2+oadd to: nlen) do:
        [vm ← ob(i-1-oadd). "integer noops"
        vm ≡ nil ⇒[] image word: i ← vm].
    self writeout.
    ↑noop, trans values]
mapHashSet | nlen i refs vm perm "all subclass on HashSet except Mdict"
    [mrefs asOop = oop ⇒[↑¯1] "do not map"
    arefs asOop = oop ⇒[↑¯1] "do not map"
    nlen ← 1 + (object class instsize). "refct + fields"
    image ← String new: 2*nlen.
    noop ← self newoop.
    [(refs ← object refct) = 2 ⇒["exactly one reference"]
        mrefs insert: oop with: (refs-2, noop)].
    image word: 1 ← refs -2.
    vm ← Vmapper new object: (object instfield: 1).
    i ← vm mapHashObs. "(oop, perm)"
    image word: 2 ← io1.
    perm ← io2. "permutation of other parts"
    for: i from: (3 to: nlen) do:
        [vm ← Vmapper new object: (object instfield: i-1).
        image word: i ← (vm mapHashVals: perm)].
    self writeout.
    ↑noop]
mapHashVals: perm | nlen oadd i refs vm "values vec of dictionary"
    [oop = ¯1 ⇒[↑¯1]
    nlen ← object length. "in words"
    oadd ← [nlen≤8⇒[0]1].
    nlen ← oadd +1 +nlen.
    image ← String new: 2*nlen.
    [(refs ← object refct) = 2 "exactly one ref" ⇒
        [noop ← self newoop]
        (i ← mrefs lookup: oop) ⇒[io1 ← io1 -1. noop ← io2]
        noop ← self newoop.
        mrefs insert: oop with: (refs-2, noop)].
    image word: 1 ← refs -2.
```

```
    [oadd=1⊃[image word: 2 ← object length]].
    object ← object∘perm.
    for: i from: (2+oadd to: nlen) do:
        [(vm ← Vmapper new) object: object∘ (i-1-oadd).
        image word: i ← vm map].
    self writeout.
    ↑noop]
mapInteger | refs
    [¯02000 ≤ object and: object < 01777 ⊃
        [↑noop "small integer"]
    image ← String new: 4.
    noop ← self newoop.
    [(refs ← object refct) = 2 ⊃["exactly one reference"]
        mrefs insert: oop with: (refs-2, noop)].
    image word: 1 ← refs-2.
    image word: 2 ← object.
    self writeout.
    ↑noop]
mapMdict | i refs vm perm  " maps MessageDict instance "
    [image ← String new: 12.
    noop ← self newoop.
    [(refs ← object refct) = 2 ⊃["exactly one reference"]
        mrefs insert: oop with: (refs-2, noop)].
    image word: 1 ← refs -2.
    (vm ← Vmapper new) object: (object instfield: 1).
    i ← vm mapHashObs. "(oop, perm)"
    image word: 2 ← io1.
    perm ← io2. "permutation of other parts"
    for: i from: (3 to: 6) do:
        [(vm ← Vmapper new) object: (object instfield: i-1).
        i = 3 ⊃[image word: 3 ← (vm mapMethodVec: perm)]
        image word: i ← (vm mapHashVals: perm)].
    self writeout.
    ↑noop]
mapMethod | nlen oadd refs i a vm b
    [oop = ¯1 ⊃[↑¯1]
    nlen ← object length. "in bytes!!"
    oadd ← [nlen≤8⊃[0]1].
    nlen ← 2*oadd +2 +nlen.
    image ← String new: nlen+1 |2.
    [(refs ← object refct) = 2 "exactly one ref" ⊃
        [noop ← self newoop]
        (i ← mrefs lookup: oop) ⊃[io1 ← io1 -1. noop ← io2]
        noop ← self newoop.
        mrefs insert: oop with: (refs-2, noop)].
    image word: 1 ← refs -2.
    [oadd=1⊃[image word: 2 ← object length]].
    [nlen ≤ 8 or: object∘6 = 6 ⊃.
        [image∘(2*oadd +3 to: nlen) ← object]
        a ← 2*oadd +3. b ← object∘6.
        image∘(a to: a+5) ← object∘(1 to: 6).
        for: i from: (4 to: b /2) do:
```

```
        [(vm ← Vmapper new) object: (object word: i) asObject.
        image word: (i+1+oadd) ← vm map].
      image∘(a+b to: nlen) ← object∘(b+1 to: object length)].
  self writeout.
  ↑noop]
mapMethodVec: perm | nlen oadd i refs vm
  [oop = ¯1 ⇒[↑¯1]
  nlen ← object length. "in words"
  oadd ← [nlen≤8⇒[0]1].
  nlen ← oadd +1 +nlen.
  image ← String new: 2*nlen.
  [(refs ← object refct) = 2 "exactly one ref" ⇒
    [noop ← self newoop]
    (i ← mrefs lookup: oop) ⇒[io1 ← io1 -1. noop ← io2]
    noop ← self newoop.
    mrefs insert: oop with: (refs-2, noop)].
  image word: 1 ← refs -2.
  [oadd=1⇒[image word: 2 ← object length]].
  object ← object∘perm.
  for: i from: (2+oadd to: nlen) do:
    [(vm ← Vmapper new) object: object∘ (i-1-oadd).
    image word: i ← vm mapMethod].
  self writeout.
  ↑noop]
mapNormal | nlen i refs vm oadd "all fixed, pointer type classes"
  [nlen ← object class instsize.
  oadd ← [nlen≤19⇒[0]1].
  nlen ← oadd +1 +nlen.
  image ← String new: 2*nlen.
  noop ← self newoop.
  [(refs ← object refct) = 2 ⇒["exactly one reference"]
    mrefs insert: oop with: (refs-2, noop)].
  image word: 1 ← refs-2.
  [oadd=1⇒[image word: 2 ← object length]].
  for: i from: (2+oadd to: nlen) do:
    [vm ← Vmapper new object: (object instfield: (i-1-oadd)).
    image word: i ← vm map].
  self writeout.
  ↑noop]
mapObject | refs "nil, false, true"
  [object≡nil ⇒
    [mrefs insert: ¯1 with: (500, ¯1). "for speed of
    catching nil. watch out on cleanup" ↑¯1]
  image ← String new: 4.
  noop ← self newoop.
  noop\128>1 ⇒
    [user notify: 'Unidentified flying Object']
  [(refs ← object refct) = 2 ⇒["exactly one reference"]
    mrefs insert: oop with: (refs-2, noop)].
  image word: 1 ← refs -2.
  self writeout.
  ↑noop]
```

```
mapString | nlen oadd refs
   [nlen ← object length. "in bytes!!"
   oadd ← [nlen≤8⇒[0]1].
   nlen ← 2*oadd +2 +nlen.
   image ← String new: nlen+1 |2.
   noop ← self newoop.
   [(refs ← object refct) = 2 ⇒["exactly one reference"]
      mrefs insert: oop with: (refs-2, noop)].
   image word: 1 ← refs -2.
   [oadd=1⇒[image word: 2 ← object length]].
   image∘((2*oadd +3) to: nlen) ← object.
   self writeout.
   ↑noop]
mapUniqueString | nlen oadd refs
   [[UseOldUniqueStrings ⇒[self UniStroop. noop ← oop]
      (refs ← arefs lookup: oop) ⇒[↑refs]
      noop ← self newoop.
      arefs insert: oop with: noop].
   nlen ← object length. "in bytes!!"
   oadd ← [nlen≤8⇒[0]1].
   nlen ← 2*oadd +2 +nlen.
   image ← String new: nlen+1 |2.
   image word: 1 ← 1.
   [oadd=1⇒[image word: 2 ← object length]].
   image∘((2*oadd +3) to: nlen) ← object.
   self writeout.
   ↑noop]
mapVector | nlen oadd i refs vm
   [[UseOldUniqueStrings ⇒[] USTableOop = oop ⇒[↑¯1]
      "do not map since new atom Oops"].
   nlen ← object length. "in words"
   oadd ← [nlen≤8⇒[0]1].
   nlen ← oadd +1 +nlen.
   image ← String new: 2*nlen.
   noop ← self newoop.
   [(refs ← object refct) = 2 ⇒["exactly one reference"]
      mrefs insert: oop with: (refs-2, noop)].
   image word: 1 ← refs-2.
   [oadd=1⇒[image word: 2 ← object length]].
   for: i from: (2+oadd to: nlen) do:
      [vm ← Vmapper new object: (object∘ (i-1-oadd)).
      image word: i ← vm map].
   self writeout.
   ↑noop]
mapVmem | nlen i refs vm  "VirtualMemory, Vmem and Pmap"
   [nlen ← 1 + (object class instsize). "refct + fields"
   image ← String new: 2*nlen.
   noop ← self newoop.
   [(refs ← object refct) = 2 ⇒["exactly one reference"]
      mrefs insert: oop with: (refs-2, noop)].
   image word: 1 ← refs -2.
   for: i from: (2 to: nlen) do:
```

```
    [image word: i ← ¯1]. "Vmem needed for purgealittle"
  self writeout.
  ↑noop]
permHashSet | ob i bb nili vm   "gets new atoms, returns dict of objects and permutation"
  [bb ← Dictionary new init: object length.
  i ← 1. while: (objectoi ≡ nil and: i ≠ object length) do: [i ← i+1].
  [((objectoi) class) ≡ Integer ⇒[false]; ≡ String ⇒[false] true] ⇒
    [for: i to: object length do:
      [objectoi ≡ nil ⇒[nili ← i]
      vm ← Vmapper new object: objectoi.
      bb insert: vm map with: i].
    ob ← bb values.
    for: i to: ob length do:
      [oboi ≡ nil ⇒[oboi ← nili] ].
    ↑bb]
  ob ← bb objects.
  for: i to: ob length do:
    [vm ← Vmapper new object: objectoi.
    oboi ← vm map].
  ob ← bb values.
  for: i to: ob length do:
    [oboi ← i].
  ↑bb]
```

Writing Out

```
method: sel | v str a j i   "object is a class. find this method and construct vmapper"
  [(v ← classtrans lookup: oop) ≡ false ⇒[user notify: 'object not a class']
  str ← Pmap readin: vo1. "class image"
  i ← Pmap readin: (str word: 1+4 +(oop/0200)). "mdict image"
  str ← Pmap readin: (i word: 1+1). "objects vector image"
  a ← String new: 2. a word: 1 ← sel asOop. "selector"
  j ← str find: ao2.
  ao1 ≠ (stro(j-1)) ⇒[user notify: 'can not find']
  i ← Pmap readin: (i word: 1+2). "methods vector image"
  v ← Vmapper new object: (object md method: sel).
  v noop: (i word: (j/2)). v readin.
  ↑v]
newoop | v a i indx o "see if classtrans can help find new oop"
  [(v ← classtrans lookup: (i ← object class) asOop) ⇒
    [indx ← 1 + (Vmem freelist: object).
     [i ≡ Class ⇒[oop ≤ 027 ⇒[↑oop] "touchoop set it up"
          voindx ≤ 027 ⇒[voindx ← 027+1] ];
       ≡ VariableLengthClass ⇒[o ← voindx. "in case pclass changes"
          voindx ← o max: o|128 +(oop\128) +1.
          ↑ o|128 +(oop\128)] ].
    a ← self nextfree: voindx.
    voindx ← ao2. ↑ao1]
  a ← [i is: Class ⇒[1]20].
  v ← Vector new: a+1.
  classtrans insert: (object class) asOop with: v.
```

```
    ↑self newoop]
newpcl: n | a pm0
   ["object class = Foo ⇒[an exception]"
   pm0 ← Vmem lowPM: oop.
   a ← [pm0 ⇒[Pmap newLowPM← pm0]
        Pmap newHighPM← (Vmem highPM: oop)].
   n=¯1 ⇒[Pmap newZN: a. Pmap ZHB: a gets: 0. ↑a]
   Pmap ZN: a gets: (Pmap ZN: n).
   Pmap ZHB: a gets: 1 + (Pmap ZHB: n).
   ↑a]
nextfree: n | a
   [n≡nil ⇒[n ← self newpcl: ¯1.
      ↑(n, (n+1))]
   n\0200 = 0177 ⇒[a ← self newpcl: n.
      ↑(n, a)]
   ↑(n, (n+1))]
object: object
   [oop ← object asOop]
readin | v "read object off vmemfile and return image"
   [[noop ≡ nil ⇒[(v ← mrefs lookup: oop) ⇒[noop ← v∘2]
      user notify: 'can not find new oop']].
   ↑image ← Pmap readin: noop]
writeout
   ["write image of object on file"
   Pmap writeout: noop with: image]
```

Init and Exceptions

Acomment

```
   ["Vmapper writes objects out.
   watch out for abnormally high refcts.
   we trust map to catch all special cases (especially
   fixed octave classes (VariableLengthClass)).
      Vmapper edit: ↗mapObject.
   Vmem is VirtualMemory of this system.
   Pmap is VirtualMemory of new system.
   refct in mrefs is 1-order (0 = all done)
   mrefs - a Dictionary
      old oop  -> (remaining refct, new oop)
   classtrans - a Dictionary
      old class oop  -> (new oop, freelist oop)
            -> (new oop, 20 freelist oops)
   sequence of foreign object:
      map (put in mrefs)
      newoop (classtrans lookup old class)
      not there⇒ create pclass, get zone, zip entry
         enter freelist, no new class in classtrans
      put nop in mrefs, write object out
      later: map class, fill in classtrans new class
      later: pass over PM, convert RCI

   To remap atoms:              To only copy atoms:
   giveBirth3 - arefsRectify       - don't call
```

```
    mapUniqueString -uniStrOop        -newoop, arefs
    mapVector - filter USTable        -don't
    writefreelists - don't freelistRectify          -call it
      (Speed only below)
    permHashSet - test objects        -force fail on test
    Now above done by setting UseOldUniqueStrings in giveBirth3.
    "]
arefsRectify | i ustable vm "create USTable and write out"
    [UseOldUniqueStrings ⇒[↑nil "dont create if using old atoms"]
    ustable ← Vector new: USTable length.
    for: i to: ustable length do:
      [ustable∘i ← Vector new: 2].
    i ← ustable. ustable ← USTable. USTable ← i.
    for: i from: arefs objects do:
      [i ≡ nil ⇒[]
      ↗a intern: i asObject].
    i ← ustable. ustable ← USTable. USTable ← i.
    vm ← Vmapper new.
    vm object: ustable.
    self reset; object: (Smalltalk ref: ↗USTable); readin.
    image word: 2 ← vm map. "ref new table"
    self writeout; reset]
bump "count objects mapped, print"
    [count ← count + 1.
    count \ 100 = 0 ⇒[user clearshow: count asString; space;
      show: oop base8; cr.
      count = stopcount ⇒[user ev; show: stopcount asOop base8]]
    ]
freelistRectify | i v "fix bad atom freelist"
    [[UseOldUniqueStrings ⇒[] ↑nil]. "dont rectify if new atom names created"
    for: i from: (Vmem pmatm to: 0174000 - 0200 by: 0200) do:
      [(Vmapper new object: i asObject) UniStroop].
    v ← classtrans lookup: 0602.
    for: i from: (2 to: 21) do: [v∘i ≡ nil ⇒[]
      v∘i ← v∘i + 1]
    ]
init "set up class variables"
    [PCL ← 151.
    classtrans ← Dictionary new init: 32.
    mrefs ← Dictionary new init: 64.
    arefs ← Dictionary new init: 64.
    count ← 1.
    stopcount ← 5 "no stopping".
    USTableOop ← USTable asOop.
    "override this in VirtualMemory giveBirth3"
    UseOldUniqueStrings ← true]
mrefsRectify | v zz
    [for: v from: mrefs values do:
      [v ≡ nil ⇒[]
      v∘1 = 0 ⇒["refct ok"]
      v∘2 = ¯1 ⇒["nil not on disk"]
      zz ← Pmap obwiz: v∘2. "noop"
```

```
      image ← Pmap read: 2 at: zz.  "2 words"
      image word: 1 ← (image word: 1) - (vo1).
      Pmap writeout: vo2 with: image.
      self reset]
   ]
noop: noop "give it noop for readin"
reset [noop ← image ← nil]
touchoop | v a i indx "claim pclass, zone, classtrans"
   [(v ← classtrans lookup: (object class) asOop) ⊃
      [indx ← 1 +(Vmem freelist: object).
      a ← self nextfree: voindx.
      voindx ← ao1. "not take it" ∩ao1]
    i ← [object class is: Class ⊃[1] 20].
    v ← Vector new: i+1.
    classtrans insert: (object class) asOop with: v.
    ∩self touchoop]
UniStroop | pm0 v indx
   ["Unique Strings must get same oop and pclass"
   v ← classtrans lookup: 0602.
   indx ← 1+ (Vmem freelist: object).
   pm0 ← Pmap highPM: oop.
   pm0 and: 0 ≠ pm0 ⊃["test for new max freelist"
      (oop field: PCL) = (voindx field: PCL) ⊃
      [voindx ← voindx max: oop]]
   Pmap highpm0: oop gets: (Vmem highPM: oop).
   Pmap ZHB: oop gets: (Vmem ZHB: oop).
   voindx ≡ nil ⊃[Pmap newZN: oop.
      voindx ← oop]
   Pmap ZN: oop gets: (Pmap ZN: voindx).
   voindx ← oop min: voindx
   ]
UseOldUniqueStrings: a [UseOldUniqueStrings ← a] "true if do not want atoms
remapped"
writefreelists | v oadd i nlen str
   [self freelistRectify.
   nlen ← 1 + Class instsize. oadd ← 1.
   str ← String new: 4. str word: 1 ← 1. "refct=1, link = nil"
   for: v from: classtrans values do:
      [v ≡ nil ⊃[]
      image ← Pmap readin: vo1.  "noop"
      [v length = 2 ⊃
         ["a Class" vo2 ≡ nil ⊃[]
         image word: (1+nlen) ← vo2.
         Pmap writeout: vo2 with: str]
      "a Variable Length Class"
      for: i to: 20 do:
         [vo(1+i) ≡ nil ⊃ []
         image word: (i+nlen+oadd) ← vo(1+i).
         Pmap writeout: vo(1+i) with: str] ].
      Pmap writeout: vo1 with: image].
   self reset]
```

```
As yet unclassified
Addons
   [(dpO file: 'VmapperAdd.st') filout: ↗(
   )]
arefs "dictionary of UniqueString oops and noops"
   [ΠarefS]
classtrans [Πclasstrans]
image [Πimage]
mrefs "dictionary of multiple refct oops"
   [ΠmrefS]
noop [Πnoop]
object [Πobject]
oop [Πoop]
printon: strm
   [oop ≡ nil ⇒[super printon: strm]
   strm append: 'Vmapper '; append: oop base8]
show | i "show image of object in base8"
   [for: i to: image length/2 do:
      [user show: (image base8: i); space.
      i\3 = 0 ⇒[user cr]]
   ]
SystemOrganization classify: ↗Vmapper under: 'Primitive Access'.
```

"ParagraphScanner"

```
Class new title: 'ParagraphScanner'
  subclassof: Object
  fields: 'para "<Paragraph>"
    style "<TextStyle>"
    press "<PressFile> for output"
    runstrm "<Stream> of paragraph runs"
    textstrm "<Stream> of paragraph text"
    font "<WidthTable> current font"
    ascent "<Integer> max ascent"
    descent "<Integer> negative max descent"
    width "<Integer> total width"
    spaces "<Integer> number of spaces"
    rect "<Rectangle> for printing"
    tabpos "<Stream> (text position, new X position) of tabs"
    '
  declare: '';
  asFollows↓
```

Scans through a paragraph computing the dimensions of a partial line of text.

Initialization

in: rect
init
```
  [ascent ← descent ← width ← spaces ← 0. tabpos reset]
```
of: para to: press style: style
```
  [textstrm ← '' asStream.
  runstrm ← para runs asStream.
  tabpos ← (Vector new: 10) asStream]
```

Access

position [↑textstrm position]
width [↑width]

Scanning

backup
```
  [textstrm skip: ¯1]
```
scan *"Scan up to a zero-width character, back up to last blank if width exceeded"*
```
    | maxw sp char t
      "Save state" spos slim srunpos sasc sdesc swidth ssp sfont stpos
  [textstrm end and: self newrun≡false⊃ [↑false]
  maxw ← rect width.
  while:
    [ascent ← ascent max: font ascent.
    descent ← descent min: font descent.
    sp ← font space.
    while: [
```

```
        t ← font scan: textstrm until: width exceeds: maxw.
        [(char ← to1) ≡ true⇒ [] width ← to2].
        char = 040] do:
        ["Save state"
          spos ← textstrm position. slim ← textstrm limit.
          srunpos ← runstrm position. stpos ← tabpos position.
          sasc ← ascent. sdesc ← descent.
          swidth ← width. ssp ← spaces. sfont ← font.
        spaces ← spaces+1.
        width ← width+sp].
      char⇒
        [char≡true and: nil≠spos⇒ "Back up to just past last blank"
          [textstrm of: para text from: spos+1 to: slim.
          runstrm position ← srunpos. tabpos position ← stpos.
          ascent ← sasc. descent ← sdesc.
          width ← swidth. spaces ← ssp. font ← sfont.
          ↑040]
        ↑char]
      self newrun]
    do: [].
  ↑false]
tab
  [spaces ← 0.
  tabpos next ← textstrm position;
    next ← (width ← width + font tab)]
```

Printing

```
printfrom: charpos aligned: align skip: n "Returns false if goes below bottom"
      | ybot a b ix px xs sp rs len tpos ts ntab [
  "this code basically writes the EL (entity list) for a line"
  (ybot ← rect corner y - ascent + descent)
    < rect origin y ⇒ [↑false]          "Won't fit"
  a ← charpos + 1.
  b ← textstrm position - n.
  [a > b ⇒ ["No text"]
  xs ← rect width - width.
  ts ← tabpos viewer.
  tpos ← ts next.

  "do setx before showchars"
  ix ← [align=2⇒ [xs/2]; =4⇒[xs] 0].
  px ← false.
  press sety: rect corner y - ascent;
    setspacex: (sp ← font space       "Kludge?").

  rs ← (para run: a to: b) asStream.
  while: (len ← rs next) do: [
    press selectfont: (press fontindex: rs next style: style) - 1.
    b ← a+len.
    ntab ← 0.
    while: (tpos and: tpos<b) do: [       "Put out tabs"
      [tpos = a⇒ ["no text between this tab and last"]
```

```
        "put out accumulated tabs or initial x"
        [ntab> 0⇒ [
          press skipchars: ntab; setx: px.
          ntab ← 0]
        px⇒ []
        press setx: (px ← ix)].
        press showchars: tpos-a].
        ntab ← ntab+1.
        px ← ts next.
        a ← tpos+1.
        tpos ← ts next].
      [ntab> 0⇒ [
        press skipchars: ntab;
          setx: px]
      px⇒ []
      press setx: (px ← ix)].

      [align=1 and: tpos≡false ⇒          "Reset space width"
        [press setspacex: xs/spaces+sp.
        align ← 0]].
      rs end⇒ [
        "for more compactness, maybe"
        press showchars: b-a skip: n.
        ↑ybot]
      press showchars: b-a.
      a ← b]].
    [n > 0⇒ [
      "skip over ending blank or carriage return"
      press skipchars: n]].
    ↑ybot]
```

Private scanning

```
newrun | len pos [
  len ← runstrm next⇒
    [pos ← textstrm position.
    textstrm of: para text from: pos+1 to: pos+len.
    font ← press codefont: (runstrm next) style: style]
  ↑false]
```

SystemOrganization classify: ↷ParagraphScanner under: 'Press File Support'.

"PressFile"

```
Class new title: 'PressFile'
   subclassof: Object
   fields: 'DL "<File> stores data list"
      EL "<Set> accumulates entity list"
      parts "<Set> accumulates part directory"
      DLstart "<Integer> position of current entity in DL"
      ELstart "<Integer> word position of current entity in EL"
      Pstart "<Integer> record position of current page in DL"
      eorigin "<Point>"
      scale "<Integer> micas per Alto screen dot"
      boundbox "<Rectangle> bounding box for current page"
      fontcodes "<Vector> of run codes corresponding to current fonts"
      fontdefs "<Vector of WidthTables> corresponding to fontcodes"
      estate "<Vector> of some entity state" '
   declare: 'prevstyle SMentity recordsize ';
   asFollows_!
```

There are two levels of code in this class: the low-level Press commands and the high level user commands. At the moment, only text, lines and bitmaps are supported (see Paragraph presson:in: and class ParagraphScanner for the former). ignores bounding box stuff. limited reading.

see <GR-DOCS>PressFormat.Press and PressFormat-figure.Press for more details

Initialization

```
edit [↑self edit: Galley new]
edit: d [
   "this is a kludge so readpart can talk to the document"
   (estate ← d)
      fromPress: self name: 0 value: nil.
   "for garbage collection"
   parts ← false.
   estate ← nil.
   ↑d]
of: DL [
   EL ← Set new string: 200.
   parts ← Set new string: 40.
   fontcodes ← Vector new: 0.
   fontdefs ← Vector new: 0.
   estate ← Vector new: 3 "font, spacex, spacey, ...".
   prevstyle← nil.
   self scale: PressScale;
      startPage]
reset [self of: DL reopen readwrite]
scale: scale
```

Aspects

```
name [↑DL name]
scale [↑scale]
```

Entity/Page/File Commands

```
box: rect hue: hue sat: sat bright: bright containing: expr | w r
   [self entity: (self transrect: (w← rect inset: ¯2)) containing:
      [for: r from: (w minus: rect) do:
         [self showrect: r color: 0].
      [ColorPrint⇒
         [self hue: hue; saturation: sat;
            showrect: rect color: bright; brightness: 0]].
      expr eval]]
clip: boundingbox
close | p i font [
   self closePage.
   parts≡false or: parts empty⇒ []

   "put font names and descriptions into font directory (part)"
   self part: [
      for: i to: fontdefs length do: [
         font ← fontdefs o i.
         DL nextword← 16; nextword← i-1;
            next ← font min; next ← font max;
            append: (font name asBCPL: 20);
            next ← font face; next ← font min;
            nextword ← font pointsize; nextword← 0]]
      code: 1.

   "write part directory. Pstart is current page position"
   DL append: parts.
   self padpage.
   p ← DL pageposition.

   "document directory"
   DL nextword← 27183; "press password"
      nextword← p + 1 "number of records";
      nextword← parts position / 8 "number of parts";
      nextword← Pstart; "part dir and length"
      nextword← p - Pstart;
      nextword← ¯1; "backpointer"
      append: user timewords; "2 time words"
      nextword← 1; "first and last copies"
      nextword← 1;
      nextword← ¯1; "first and last pages"
      nextword ← ¯1.
   for: p from: 12 to: 0177 do: [DL nextword← ¯1].
   p ← user now.
   DL append: (self name asBCPL: 52);
      append: (dp0 username o1 asBCPL: 32);
      append: ((p o 1) asString+' '+ (p o 2) asString asBCPL: 40).
   self padpage.
   DL sclose.
```

```
   parts reset]
entity: box containing: expr | v [
   self startEntity.
   boundbox ← box.
   v ← expr eval.
   self closeEntity.
   ↑v]
entityorigin: eorigin
page [self closePage]
pictureinit [self pictureinit: user screenrect scale: PressScale]
pictureinit: rect scale: scale
   [boundbox ← boundbox include: (self transrect: rect).
   self somefont]
screenout: rect scale: scale
   ["puts a bit map image onto the PressFile.  The standard
   scaling is 32 micas per Alto dot.  22 looks better, Dover only
   works with 32"
   user displayoffwhile: [
      self somefont; bitmap: rect bits: false; close]]
toPrinter [self toPrinter: ↗(3 0121 'Clover')]
toPrinter: dest | a [
   E≡nil⇒ [
      "use O.S. if Smalltalk ethercode not alive"
      a ← 'Empress.Run ' + self name.
      [dest length> 2⇒ [a ← a + ' ' + (dest∘3) + '/H']].
      user quitThen: a continue: true]
   "for now, always init and kill ether"
   E systemInit.
   a ← EFTPSender new net: dest∘1 host: dest∘2.
   a enable;
      send: DL readonly reopen;
      close.
   E etherKill]
```

Fonts

```
codefont: code style: style
   [↑fontdefs∘(self fontindex: code style: style)]
fontindex: code style: style | ix font n
   ["return index if in font dictionary"
   code ← code land: 0363.          "Remove underline and strikeout"
      [style=prevstyle⇒
         [(ix ← fontcodes find: code) > 0 ⇒ [↑ix]]
      fontcodes all← nil. "invalid across style change"
      prevstyle← style].
   n ← code / 16 + 1.
   font ← (WidthTable new
      named: (style fontfamily: n)
      pointsize: (style fontsize: n)
      face: (code / 2 land: 1) + (code * 2 land: 2))
    lookup.
   (ix← fontdefs find: font)>0⇒
      [fontcodes∘ix← code. ↑ix]
```

```
   "add entry to font dictionary"
   fontdefs length=16⇒[user notify: 'too many fonts'. ↑1]
   fontcodes ← fontcodes, code.
   fontdefs ← fontdefs, font.
   ↑fontcodes length]
selectfont: f [
   estate∘1 = f⇒ []
   EL next ← 0160 + (estate∘1 ← f)]
somefont        "fool self into writing non-empty fontdir"
   [self fontindex: 5*16 style: DefaultTextStyle]
```

Transformations

```
transpt: p
   [↑ Point new x: (p x * scale) asInteger y: (25400 - (p y * scale)) asInteger]
transrect: rect
   [↑ Rectangle new
      origin: (self transpt: rect minX ⊙ rect maxY)
      corner: (self transpt: rect maxX ⊙ rect minY)]
```

EL commands

```
brightness: b [EL next← 0370; next← b]
hue: b [EL next← 0371; next← b]
onlyoncopy: n [EL next ← 0355; next ← n]
resetspace [EL next ← 0366]
saturation: s [EL next← 0372; next← s]
setp: p [
   "self setx: p x; sety: p y"
   EL next ← 0356; nextword ← p x;
      next ← 0357; nextword ← p y]
setspacex: x [
   estate∘2 = x⇒ []
   (estate∘2 ← x)
      between: 0 and: 2047⇒ ["short form" EL nextword ← 060000 + x]
   EL next ← 0364; nextword ← x]
setspacey: y [
   estate∘3 = y⇒ []
   (estate∘3 ← y)
      between: 0 and: 2047⇒ ["short form" EL nextword ← 064000 + y]
   EL next ← 0365; nextword ← y]
setx: x [EL next ← 0356; nextword ← x]
sety: y [EL next ← 0357; nextword ← y]
showchar: char ["immediate" EL next ← 0363; next ← char]
showchars: n [
   n=0⇒ [];
      between: 1 and: 32⇒ ["short form" EL next ← n-1]
   EL next ← 0360; next ← n]
showchars: n skip: t [
   t=1 and: (n between: 1 and: 32)⇒ [EL next ← 0100 + n-1]
   self showchars: n; skipchars: t]
showdots: nwords [
   EL next ← 0374; nextNumber: 2 ← nwords]
```

```
showdotsopaque: nwords [
   EL next ← 0375; nextNumber: 2 ← nwords]
showrect: rect color: c | r [
   r ← self transrect: rect.
   self setp: r origin;
      brightness: c;
      showrectwidth: r width height: r height]
showrectwidth: w height: h [EL next ← 0376; nextword ← w; nextword ← h]
skipchars: n [
   n=0⇒ [];
      between: 1 and: 32⇒ ["short form" EL next  ← 040 + n-1]
   EL next ← 0361; next ← n]
skipcontrol: n [
   "immediate"
   EL next ← 0353; next ← n.
   "now put n bytes in EL"]
skipcontrol: n type: t [
   "n bytes have been put in DL"
   EL next ← 0362; nextword ← n; next ← t]
space [EL next ← 0367]
```

Bitmaps/Dots

```
bitmap: rect bits: bits | r w w16 h [
   r ← self transrect: rect.
   w ← rect width.
   w16 ← w + 15 | 16 "width to next word boundary".
   h ← rect height.
   self setp: r origin;
      dots: [
         self setcoding: 1 "bitmap" dots: w16 lines: h;
            setmode: 3 "to right and to bottom";
            setsizewidth: scale * w16 height: scale * h;
            setwindowwidth: w height: h;
            dotsfollow.
         bits⇒["bits supplied" DL append: bits]
         "else from screen"
         rect bitsOntoStream: DL]]
dots: exp | dlpos [
   dlpos ← self padword.
   exp eval.
   self showdots: DL wordposition - dlpos]
dotsfollow [DL nextword ← 3]
setcoding: c dots: d lines: l [
   DL next ← 1; next ← c;
      nextword ← d; nextword ← l]
setmode: m [DL next ← 2; next ← m]
setsizewidth: w height: h [
   DL nextword ← 2; nextword ← w; nextword ← h]
setwindowwidth: w height: h [
   DL nextword ← 1;
      nextword ← 0 "skip dots"; nextword ← w;
      nextword ← 0 "skip lines"; nextword ← h]
```

Lines/Objects

```
drawcurve: v [
   v length ≠ 12⇒ [user notify: 'illegal drawcurve']
   DL nextword ← 2.
   for: v from: v do: [DL nextword ← v]]
drawdiscat: pt radius: radius | dx dy i
   [radius ≤ 16 ⇒ []
    dx ← ↗(5 4 3 1 ¯1 ¯3 ¯4 ¯5 ¯5 ¯4 ¯3 ¯1 1 3 4 5).
    dy ← ↗(1 3 4 5 5 4 3 1 ¯1 ¯3 ¯4 ¯5 ¯5 ¯4 ¯3 ¯1).
    self showobject: [
      self moveto: pt + ((dx∘16*radius/5) ⊙ (dy∘16*radius/5)).
       for: i to: 16 do:
         [self drawto: pt + ((dx∘i*radius/5) ⊙ (dy∘i*radius/5))]]]
drawlinefrom: p1 to: p2 width: width | d length t1 t2
   [[(d ← p2-p1) = (0⊙0) ⇒ []
      d x← d x asFloat. d y← d y asFloat. width ← width asFloat.
      length ← ((d x*d x)+(d y*d y)) sqrt.
      d x← (d x*width/length) asInteger.
      d y← (d y*width/length) asInteger.
      t1 ← d y ⊙ (0 - d x).
      t2 ← 0 - d y ⊙ d x.
      self showobject: [
         self moveto: p1 + t1.
         self drawto: p2 + t1.
         self drawto:  p2 + t2.
         self drawto:  p1 + t2.
         self drawto:  p1 + t1]].
    self drawdiscat: p2 radius: width]
drawlinefromscreen: p1 to: p2 width: width
   [↑self drawlinefrom: (self transpt: p1) to: (self transpt: p2) width:
(width*scale)]
drawto: p [DL nextword ← 1; nextPoint ← p]
moveto: p [DL nextword ← 0; nextPoint ← p]
object: expr atScreen: p
   [self showobject: [self objectGotoScreen: p pen: 0. expr eval]]
objectGotoScreen: p pen: pen [
   DL nextword← pen; nextPoint ← (self transpt: p)]
showobject: exp | p [
   p ← self padword.
   "expression containing moveto, drawto, drawcurve"
   exp eval.
   EL next ← 0373; nextword ← DL wordposition - p]
```

Private

```
append: x
   [↑DL append: x]
classInit [
   Smalltalk declare: ↗PressScale as: 32.
   recordsize ← 512.
   SMentity ← 5]
```

```
closeEntity [self closeEntity: SMentity]
closeEntity: etype [
   EL wordposition = ELstart⇒ []
   "Put a trailer into the EL"
   EL positioneven: 0377;          "word-pad EL with <Nop>"
      next ← etype;
      next← 0; "fontset"
      "dlstart relative to DL location in file"
      nextNumber: 2 ← DLstart - (Pstart*recordsize);
      nextNumber: 2 ← DL position - DLstart;
      nextPoint← eorigin; "entity origin"
      nextPoint← boundbox origin;
      nextPoint← boundbox extent.
   EL nextword ← EL wordposition - ELstart + 1.
   self startEntity]
closePage [
   self closeEntity.
   EL empty⇒ []
   DL positioneven: 0;
      nextword ← 0;
      append: EL.
   self part: 0 start: Pstart]
data ["slightly dangerous" ↑DL]
padpage | i p [
   "words of padding to end of page"
   for: i to: (p ← recordsize - DL char\recordsize) do: [DL next ← 0].
   ↑p/2]
padword [
   "make object (lines or dots) start on word boundary"
   [DL char even⇒ []
   DL positioneven: 0.
   self skipchars: 1].
   ↑DL wordposition]
part: exp code: c | fp [
   self closePage.
   fp ← DL pageposition.
   exp eval.
   self part: c start: fp]
part: type start: start | padding [
   padding ← self padpage.
   parts nextword ← type;
      nextword ← start;
      nextword ← DL pageposition - start;
      nextword ← padding.
   self startPage]
skipcode: code data: s | t [
   "called by hidePress:"
   (t ← s length+1) < 256⇒ [
      "immediate, in EL"
      self skipcontrol: t.
      EL next ← code; append: s]
   "in DL"
```

```
   DL next ← code; append: s.
   self skipcontrol: t type: SMentity]
startEntity [
   DLstart ← DL position.
   ELstart ← EL wordposition.
   boundbox ← 0 asRectangle.
   eorigin ← 800⊙800.
   estate all ← ¯1.
   estate∘1 ← 0]
startPage [
   EL reset.
   Pstart ← DL pageposition.
   self startEntity]
```

Reading

```
asStream [self open]
interpret: entity to: obj | command t i [
   while: (command ← entity next) do: [
   "some stuff arranged by probable frequency"
   command
   < 0100⇒ [
      "show-characters-short (0-037)
      skip-characters-short (040-077)"
      DL skip: (command land: 037) +1];
   = 0356⇒ ["set-x" entity nextword];
   = 0357⇒ ["set-y" entity nextword];
   < 0140⇒ [
      "show-characters-and-skip (0100-0137)"
      DL skip: (command land: 037) +2];
   < 0160⇒ [
      "set-space-x-short (0140-0147)
      set-space-y-short  (0150-0157)"
      "(command land: 7)*256 +" entity next];
   < 0200⇒ ["font" "command land: 017"];
   = 0362⇒ [
      "skip-control-bytes"
      t ← entity nextword.
      entity next≠SMentity⇒ ["ignore" DL skip: t]
      i ← DL next.
      obj ← self sendcontrol: obj name: i value: (DL next: t-1)];
   = 0360⇒ ["show-characters" DL skip: entity next];
   = 0377⇒ ["nop"];

   < 0353⇒ [
      "available (0200-0237)
      spare (0240-0352)"];
   = 0353⇒ [
      "skip-control-bytes-immediate"
      t ← entity next.
      i ← entity next.
      obj ← self sendcontrol: obj  name: i value: (entity next: t-1)];
   = 0354⇒ ["alternative" entity skipwords: 5];
```

```
    = 0355⇒ ["only-on-copy" entity next];
    = 0361⇒ ["skip characters" DL skip: entity next];

    = 0363⇒ ["show-character-immediate" entity next];
    < 0366⇒ [
      "set-space-x (0364)
      set-space-y (0365)" entity nextword];
    < 0370⇒ [
      "reset-space (0366)
      space (0367)" ];
    < 0373⇒ [
      "set-brightness (0370)
      set-hue (0371)
      set-saturation (0372)" entity next];

    = 0373⇒ ["show-object" DL skipwords: entity nextword];
    < 0376⇒ [
      "show-dots (0374)
      show-dots-opaque (0375)"
      DL skipwords: (entity nextNumber: 2)];
    = 0376⇒ ["show-rectangle" entity skipwords: 2]
    ].
    ↑obj]
  next [↑self nextEntity: TextEntity default]
  nextEntity: obj | e trailer [
    "get next Smalltalk entity (may span press entities/pages)"
    while: true do: [
      trailer ← EL next⇒ [
        e ← EL next viewer.
        trailer◦1 ≠ SMentity⇒ ["ignore this entity"]
        DLstart ← (trailer◦(3 to: 6)) asStream nextNumber: 2.
        DL settopage: Pstart + 1 char: DLstart.
        (obj ← self interpret: e to: obj) complete⇒ [↑obj]]

      "no more entities on current part (page)"
      self readPart⇒ []
      "no more pages"
      ↑false]]
  open | t [
    "read the parts (and font directory?)"
    DL readonly settoend; skip: ¯512.
    DL nextword = 27183 and: DL nextword = DL page⇒ [
      t ← DL nextword.
      DL settopage: DL nextword+1 char: 0.
      parts ← (DL next: t*8) viewer.
      self readPart]
    user notify: 'not a press file']
  readPart | t [
    "read parts until we find a printed page or end"
    while: (t ← parts nextword) do: [
      Pstart ← parts nextword.
      t ≠ 0⇒ [
```

```
            "not a printed page"
            parts skip: 4.
            t> 0⇒ ["font or other part"]
            "a non-standard part. let document (estate) interpret"
            DL settopage: Pstart+1 char: 0.
            estate fromPress: self name: t value: DL]

        "go to end of last record of entity list, ignoring padding"
        t ← parts nextword "length".
        DL settopage: Pstart + t char: 2 * (255 - parts nextword).
        EL ← Set new vector: 50.
        "scan backwards for beginning of entity list, reading entities"
        while: (t ← DL nextword) > 0 do: [
            t < 12⇒ [user notify: 'illegal entity']
            DL skipwords: 0-t.
            "read entity and trailer (last 12 words of entity)"
            EL next ← DL next: t-12 *2.
            EL next ← DL next: 24.
            DL skipwords: ¯1 - t].
        "now reverse:  trailer, entity (1st), ... (last)"
        ↑EL ← (EL asArray◦(EL length to: 1 by: ¯1)) asStream]
    ↑false]
sendcontrol: obj name: code value: s [
    obj fromPress: self name: code value: s⇒ [↑obj]

    "a change of the id-entity of obj can occur here,
    e.g. normally TextEntitys are on the file, but a BitRect or
    some other entity might have been written.

    look at code and send the fromPress: message again"

    code = 10⇒ [
        (obj ← BitRect new) fromPress: self name: code value: s⇒ [↑obj]
        user notify: 'BitRect should recognize code 10']

    user notify: 'unrecognized code']
]
SystemOrganization classify: ↶PressFile under: 'Press File Support'.]
PressFile classInit]
```

"WidthTable"

```
Class new title: 'WidthTable'
   subclassof: Object
   fields: 'name "<String> name of font family"
      pointsize "<Integer> size in points"
      face "<Integer> Press face code"
      min "<Integer> min character code in font"
      max "<integer> max character code in font"
      "Ascent, descent, and width are in micas"
      ascent "<Integer> max ascent of characters in font"
      descent "<Integer> NEGATIVE max descent of characters in font"
      widths "<Vector of Integers> widths of characters"
      '
   declare: 'WidthDict ';
   asFollows↓
```

Holds font parameters and width table for a Press font. It knows how to load itself from FONTS.WIDTHS.

Initialization

```
classInit
   [WidthDict ← Dictionary init]
lookup | key font i [
   key ← name + pointsize asString + (↱('' 'I' 'B' 'BI')∘(face+1)).
   font ← WidthDict lookup: key⇒ [↑font]
   self fontfrom: (dp0 file: 'fonts.widths') readonly.
   for: i from: ↱(011 015 040) do:
      [i between: min and: max ⇒ [widths∘(i-min+1) ← 0]].
   WidthDict insert: key with: self.
   ↑self]
named: name pointsize: pointsize face: face
```

Access

```
ascent [↑ascent]
descent [↑descent]
face [↑face]
max [↑max]
min [↑min]
name [↑name]
pointsize [↑pointsize]
scan: strm until: width exceeds: maxw | char w [
   while: (char ← strm next) do:
      [char < min ⇒
         [char=040 or: (char=015 or: char=011) ⇒ [↑char, width]
         user notify: 'char too low']
      char > max ⇒ [user notify: 'char too high']
      (w ← widths∘(char+1-min)) = 0 ⇒ [↑char, width]
      (width ← width + w) > maxw ⇒ [↑true, width]
      ].
```

```
   ↑false, width]
space [↑150]
tab [↑500]

Reading FONTS.WIDTHS
findfield: n on: file | IXH [
   while: [
      IXH ← file nextword.
      (IXH bits: (0 to: 3)) "type"
         = 0 ⊃ [user notify: 'field not found'];
         ≠ n]
    do:
      [file skipwords: (IXH land: 07777 "length") - 1]]
fontfrom: file | i code fam fmin fmax start len found w scale
   ["find code for font family"
   file reset. fam ← ''.
   until: (fam = name) do:
      [self findfield: 1 on: file.
      code ← file nextword.
      fam ← file next: (len ← file next).
      file skip: 19 - len].
   "now search for proper face"
   found ← false.
   "Convert from points to micas"
   scale ← (pointsize asFloat * 2540 / 72) asInteger.
   until: found do:
      [self findfield: 4 on: file.
      found ← [file next = code].
      [file next ≠ face ⊃ [found ← false]].
      fmin ← file next.
      fmax ← file next.
      i ← file nextword. [i ≠ scale and: i ≠ 0 ⊃ [found ← false]].
      file skip: 4. start ← file nextword. file skip: 4].
   scale ← [i ≠ 0 ⊃ [1 "don't need to scale"] pointsize asFloat * 254 / 7200].
   min ← fmin. max ← fmax.
   "get bb and x-tables"
   file wordposition← start+1.
   descent ← 0 - (scale * file nextword) asInteger.
   file nextword.
   ascent ← (scale * file nextword) asInteger.
   file nextword.
   widths ← Vector new: (max - min + 1).
   for: i to: widths length do:
      [w ← file nextword.
      widths↓i ← [w > 0 ⊃ [(scale * w) asInteger] 0]].
   ]
┘
SystemOrganization classify: ↶WidthTable under: 'Press File Support'.┘
WidthTable classInit┘
```

"Generator"

```
Class new title: 'Generator'
   subclassof: Object
   fields: 'literals nTemps maxTemp local environment parser
supered root requestor sourceStream'
   declare: '';
   sharing: ByteCodes;
   asFollows_!
```

I generate code parsed by parser. The symbol tables I use are local and environment. The run-time needs of the code are recorded in literals, nTemps, and maxTemp. If a message was passed to super, then supered is true. I remember my root context to abort in case of error.

Services

```
compile: sourceStream in: class under: category notifying: requestor | selector
   [user displayoffwhile:
      [selector ← self compileIn: class⇒
         [class organization classify: selector under: category]].
   ↑selector]
evaluate: sourceStream in: context to: receiver notifying: requestor
      | method nvars value tframe
   [method ← user displayoffwhile: [self evaluateIn: context to: receiver].
   root≡true≡false⇒ [↑method] "compilation failed, return false or corrected
value"
   nvars ← nTemps.
   context⇒ "frame copy here because interpret loses control"
      [tframe ← context tempframe∘(1 to: nvars) copyto: (Vector new:
method∘3).
      value ← context interpret: method with: tframe.
      tframe∘(1 to: nvars) copyto: context tempframe.
      ↑value]
   ↑Context new have: receiver interpret: method]
```

Errors

```
abortWith: errorString | mySender
   [[WhatFlag⇒ [user notify: errorString]].
   mySender ← thisContext swapSender: root sender.
   root sender ← nil. root ← nil. parser terminate.
   mySender release. mySender ← nil.
   user restoredisplay.
   ↑requestor notify: errorString at: sourceStream position in: sourceStream]
"Parser notify"
notify: errorString
   [parser notify: errorString]
"ParsedObjectReference remote"
terminate
   [root ← nil]
"Parser terminate"
```

Code generation

compileIn: class

```
    | block method nargs selector primitive
  [parser ← Parser new. root ← thisContext. parser from: sourceStream to: self.
  self initSymbols: class.
  block ← ParsedBlock default.
  selector ← parser pattern: block. nargs ← nTemps.
  parser temporaries: block. primitive ← parser body: block.
  parser mustBeDone. parser ← nil.
  block mustReturn: true "defaults to ↑self".
    [method ← [primitive=0 and: nargs=0⇒ [block quickCode] false]⇒ []
    method ← self generate: block in: class.
    method↓2 ← primitive; ↓4 ← nargs].
  class install: selector method: method literals: literals
    code: sourceStream asArray backpointers: nil.
  [HuhFlag⇒ [Huh←nil. Huh ← (self decompile: method onto: Stream default)
contents. HuhFlag←false]].
  ↑selector]
"compile"
```

decompile: method onto: s

```
  [method length<6⇒
    [s append: 'Quick code: '; append: method asBytes. ↑s]
  s print: method↓4; append: ' args; ';
    print: method↓5; append: ' temps; ';
    print: (method↓3) - (method↓5); append: ' stack; ';
    print: (method↓6) -6 /2; append: ' literals; '.
  [(method↓2) > 0⇒ [s append: ' primitive: '; print: method↓2; append: ';']].
  s print: method length; append: ' bytes total.'; cr.
  method↓2 = 40⇒ [↑s]
  ↑self decompileBytes: method onto: s]
```

decompileBytes: method onto: s

```
    | dict x i c m t
  [dict ← Dictionary new init: 64.
  dict insertall: ((128 to: 131) copy, 125 concat: (144 to: 175) copy)
    with: ↷(
      '←↑' '←' '↑' '↑' 'end'
      'jmp1' 'jmp2' 'jmp3 ' 'jmp4' 'jmp5' 'jmp6' 'jmp7' 'jmp8'
      'bfp1' 'bfp2' 'bfp3 ' 'bfp4' 'bfp5' 'bfp6' 'bfp7' 'bfp8'
      'jmp' 'jmp' 'jmp' 'jmp' 'jmp' 'jmp' 'jmp' 'jmp'
      'bfp' 'bfp' 'bfp' 'bfp' 'bfp' 'bfp' 'bfp' 'bfp').
  for: x from: local contents do:
    [i←local↓x. t ← i land: 255.
    [i>255 and: t<16⇒ [i←((i lshift: ¯8)-1 lshift: 4) + t]].
    dict insert: i with: x].
  for: x from: stdSelectors contents do:
    [dict insert: stdSelectors↓x with: [x is: Integer⇒ [x inString] x]].
  for: i to: 5 do: [dict insert: toLoadConst+i-1 with: ↷('¯1' '0' '1' '2' '10')↓i].
  for: t from: (m ← (method↓(method↓6 +1 to: method length)) asStream) do:
    [[t≥toLoadFieldLong and: t≤toSendLitLong⇒ [t←((t-0207) lshift: 8)+ m
next]].
     [c ← dict lookup: t⇒ [s append: c] s append: '#'. s append: t base8].
     s space.
```

```
        t < toLongJmp⇒ [] t ≥ 0260⇒ []
        s print: t\8 -4 *256 + m next; space].
    s cr.
    ↑s]
evaluateIn: context to: receiver
        | block method class nvars
    [ "If context is false, receiver will evaluate in top level"
    block ← ParsedBlock default.
    parser ← Parser new. root ← thisContext.
    parser from: sourceStream to: self.
    [context⇒
        [self initSymbols: (class ← context mclass).
        context variableNamesInto: self with: ParsedBlock default.
        nvars ← nTemps.
        root ← thisContext. "because variableNamesInto nil'ed it"]
     self initSymbols: (class ← receiver class)].
    parser temporaries: block; statements: block; mustBeDone. parser ← nil.
    block mustReturn: false "returns last value".
    method ← self generate: block in: class.
    [HuhFlag⇒ [Huh←nil. Huh ← (self decompile: method onto: Stream default)
contents. HuhFlag←false]].
    root ← true. "to signify success"
    nTemps ← nvars.
    ↑method]
"evaluate"
generate: block in: class
        | header method code lit stack
    [[(lit ← literals find: nil)>0⇒ [literals ← (literals∘(1 to: lit-1)) copy]].
    [supered⇒ [literals ← literals, (Smalltalk ref: class title unique)]].
    header ← 6 + (2* literals length).
    code ← (method ← String new: header + block sizeForValue) asStream.
    code
        next ← 0; next ← 0; next ← 0;
        next ← 0; next ← maxTemp; next ← header.
    for: lit from: literals do:
        [code next ← lit PTR lshift: ¯8; next ← lit PTR land: 0377].
    stack ← ParseStack init.
    block emitForValue: code on: stack.
    [stack position≠1⇒ [user notify: 'Compiler stack discrepancy']].
    [code position≠method length⇒ [user notify: 'Compiler code size
discrepancy']].
    method∘3 ← maxTemp + stack length.
    ↑method] "compile/evaluate"
```

Parse tree

```
assignment: var expr: expr
    [↑ParsedAssignment new var: var expr: expr]
"Parser expression"
block
    [↑ParsedBlock default]
"Parser primary/Parser alternatives"
evalKeyword: arg
```

```
    [↑arg]
"Parser keywordMessage"
ifExpr: ifExpr thenExpr: thenExpr elseExpr: elseExpr
    [↑ParsedConditional new ifExpr: ifExpr thenExpr: thenExpr elseExpr: elseExpr]
"ifthen.../Parser alternatives"
keywordMessage: rcvr selector: sel args: args
    [sel='and:'⇒
        [↑ParsedConjunct new left: rcvr right: args local];
      = 'or:'⇒
        [↑ParsedDisjunct new left: rcvr right: args local]
    ↑self rcvr: rcvr selector: sel args: (args remote: self)]
"Parser keywordMessage"
noEvalKeyword: arg
    [↑arg asRemoteCode: self]
"Parser keywordMessage"
nullStatement: block
    [block next ← toLoadNil. ↑block]
"ifthen/Parser statements"
rcvr: rcvr selector: sel args: args
    [[rcvr≡toSuper⇒ [supered←true]].
    ↑ParsedMessage new rcvr: rcvr op: (self encodeSel: sel) args: args]
"loop/keywordMessage/Parser binaryMessage/Parser unaryMessage"
receivingVar: expr | rcvr var "who in expr is cascade recipient"
    [rcvr ← expr emittedReceiver⇒
        [var ← rcvr emittedVariable⇒ [↑var]
        var ← self newTemp. "if a non-variable, compute it just once"
        expr emittedReceiver ← ParsedAssignment new var: var expr: rcvr.
        ↑var]
    parser notify: 'MAY ONLY FOLLOW A MESSAGE']
"Parser cascade"
variable: name
        | var global ref unq
    [var ← local lookup: name⇒ [↑var]
    [unq ← name hasBeenUniqued⇒
        [for: global from: environment do:
            [ref ← global lookupRef: unq⇒
                [↑codeLoadLitInd + (self litIndex: ref)]]]].
    requestor interactive⇒
        [parser notify: '␍Smalltalk declare: #' + name + ' as: nil␍TO DECLARE
GLOBAL']
    user show: ' (' + name + ' is Undeclared) '.
    unq ← name unique.
    Undeclared declare: unq.
    ↑codeLoadLitInd + (self litIndex: (Undeclared ref: unq))]
"Parser expression/Parser primary"
```

Macros

```
for: var from: startMinus1 to: stop do: ritual on: block | temp
    [temp ← self newTempForMacro.
    "temp←stop. var←startMinus1. while: temp≥(var ← 1+var) do: ritual"
    block next ← ParsedAssignment new var: temp expr: stop;
        next ← ParsedAssignment new var: var expr: startMinus1;
```

```
      next ← ParsedLoop new
         whileExpr:
            (ParsedMessage new rcvr: temp op: toGeq args:
               (ParsedAssignment new var: var
                  expr: (ParsedMessage new rcvr: toLoad1 op: toPlus args: var)))
         doExpr: ritual]
"for...todoargs"
forfromdo: block args: args | var sequence ritual strm
   [var ← (args∘1) local. sequence ← args∘2. ritual ← (args∘3) local.
   strm ← self newTempForMacro.
   "strm ← sequence asStream. whiles (var ← strm next) dos ritual"
   block next ← ParsedAssignment new var: strm
         expr: (ParsedMessage new rcvr: sequence op: toAsStream args: false);
      next ← ParsedLoop new
         whileExpr:
            (ParsedAssignment new var: var
               expr: (ParsedMessage new rcvr: strm op: toNext args: false))
         doExpr: ritual]
"macro (perform)"
forfromtobydo: block args: args
   ["fors var from: (start to: stop by: step) dos ritual"
   args∘2 ← self rcvr: args∘2 selector: 'to:by:' args: (args∘(3 to: 4)) copy.
   self forfromdo: block args: (args∘↗(1 2 5)) copy]
"macro (perform)"
forfromtodo: block args: args
   [self for: (args∘1) local
      from: (ParsedMessage new rcvr: args∘2 op: toMinus args: toLoad1)
      to: args∘3
      do: (args∘4) local
      on: block]
"macro (perform)"
fortodo: block args: args
   [self for: (args∘1) local
      from: toLoad0
      to: args∘2
      do: (args∘3) local
      on: block]
"macro (perform)"
ifthen: block args: args
   [block next ← self ifExpr: (args∘1) local thenExpr: (args∘2) local elseExpr: (self
nullStatement: ParsedBlock default)]
"macro (perform)"
ifthenelse: block args: args
   [block next ← self ifExpr: (args∘1) local thenExpr: (args∘2) local elseExpr:
(args∘3) local]
"macro (perform)"
macro: block selector: sel args: args
      | special
   [special ← inLineMsgs lookup: sel↘
      [self perform: special with: block with: args]
   Context canunderstand: sel unique↘
      [block next ← self rcvr: toLoadThisCtxt selector: sel args: (args remote:
```

```
self)]
    ↑false]
"Parser keywordMessage"
untildo: block args: args
    [block next ← ParsedLoop new whileExpr: (ParsedNegation new rcvr: (args∘1)
local op: toEq args: toLoadFalse) doExpr: (args∘2) local]
"macro (perform)"
whiledo: block args: args
    [block next ← ParsedLoop new whileExpr: (args∘1) local doExpr: (args∘2) local]
"macro (perform)"
```

Table maintenance

```
balance
    [↑nTemps]
"Parser cascade"
comment: s
"Class fieldNamesInto"
contents
"Class fieldNamesInto"
declaration: block name: name asArg: asArg
        | permVar tempVar
    [tempVar ← self newTemp.
    permVar ← local lookup: name ⇒
        [asArg and: permVar isField⇒
            [block next ← ParsedAssignment new var: permVar expr: tempVar]
        parser notify: 'NAME ALREADY IN USE']
    local insert: name with: tempVar]
"Parser declaration/temporaries"
encodeSel: sel
        | code
    [code ← stdSelectors lookup: sel⇒ [↑code]
    ↑codeSendLit+ (self litIndex: [sel class≡Integer⇒ [UST1∘(sel+1)] sel unique])]
"rcvr/ParsedFieldReference remote/ParsedRemote remote"
identifier: s
    [local insert: s with: (nTemps ← nTemps + 1)]
"Class fieldNamesInto"
initSymbols: class | s
    [environment ← class environment, Smalltalk.
    local ← Dictionary new copyfrom: stdPrimaries.
    nTemps ← codeLoadField-1.
    for: s from: class instvars do:
        [local insert: s with: (nTemps ← nTemps + 1)].
    nTemps ← maxTemp ← 0. literals ← Vector new: 123. supered ← false]
"compile/evaluate"
juggle | oldTemps
    [oldTemps ← maxTemp. maxTemp ← nTemps. ↑oldTemps]
"Parser macro"
literal: x  | i
    [[x class≡Integer⇒ [0≠(i←↷(¯1 0 1 2 10) find: x)⇒ [↑toLoadConst+i-1]]].
    ↑codeLoadLit + (self litIndex: x)]
"Parser primary/ParsedFieldReference remote"
litIndex: oop  | i t
```

```
    [for: i to: 123 do:
        [(t ← literals oi)≡nil⇒ [literals oi←oop. ↑i-1]
        t class≡oop class⇒ [t sameAs: oop⇒ [↑i-1]]].
    parser notify: 'MORE THAN 123 LITERALS REFERENCED']
"encodeSel/literal"
newTemp
    [(nTemps ← nTemps+1) > maxTemp and: (maxTemp ← nTemps) > 256⇒
        [parser notify: 'MORE THAN 256 TEMPS REQUIRED']
    ↑codeLoadTemp + nTemps-1]
"receivingVar/declaration"
newTempForMacro "juggle arranged that maxTemp are needed by args of
macro"
    [nTemps ← maxTemp. ↑self newTemp]
"forfromdo/forfromtodo"
separator: c
"Class fieldNamesInto"
trailer: s
"Class fieldNamesInto"
unbalance: nTemps
"Parser cascade"
unjuggle: oldTemps
    [maxTemp ← oldTemps max: maxTemp]
"Parser macro"
↵
SystemOrganization classify: ↗Generator under: 'Compiler'.↵
```

"ParsedAssignment"

```
Class new title: 'ParsedAssignment'
   subclassof: Object
   fields: 'var expr elide'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent an assignment of an expression to a variable.

Initialization

```
var: var expr: expr
```

Code generation

```
emitForEffect: code on: stack
   [expr emitForValue: code on: stack. stack pop: 1.
   elide⇒ ["var begins the next statement" code next ← toSmash]
   code next ← toSmashPop.
   var emitBytes: code]
emitForValue: code on: stack
   [expr emitForValue: code on: stack.
   code next ← toSmash.
   var emitBytes: code]
emittedVariable
   [↑var]
firstPush
   [↑expr firstPush]
sizeForEffect: nextPush
   [↑expr sizeForValue + 1 + [elide ← nextPush≡var⇒ [0] var sizeForValue]]
sizeForValue
   [↑expr sizeForValue + 1 + var sizeForValue]
```

Miscellaneous

```
printon: s
   [s append: '('; print: var; append: '←'; print: expr; append: ')']
↵
SystemOrganization classify: ↶ParsedAssignment under: 'Compiler'.↵
```

"ParsedBlock"

```
Class new title: 'ParsedBlock'
   subclassof: Stream
   fields: 'returns'
   declare: '';
   sharing: ByteCodes;
   asFollows
```

I am a stream to collect the statements of a block and then to become a node in a compiler parse tree.

Initialization

default
```
   [limit ← 1. array ← Vector new: 1. position ← 0. returns ← false]
```
doesReturn
```
   [returns ← true]
```
mustReturn: fromMethod
```
   [returns⇒ []
   [fromMethod⇒
      [position>0 and: (array∘position) emitsLoad⇒ [array∘position ←
toLoadSelf] self next ← toLoadSelf]].
   self doesReturn]
```

Code generation

emitForEffect: code on: stack
```
   [returns⇒ [self emitForValue: code on: stack. stack pop: 1]
   self emitExceptLast: code on: stack.
   (array∘position) emitForEffect: code on: stack]
```
emitForValue: code on: stack
```
   [self emitExceptLast: code on: stack.
   (array∘position) emitForValue: code on: stack.
   returns⇒ [code next ← toReturn]]
```
firstPush
```
   [↑(array∘1) firstPush]
```
sizeForEffect: nextPush
```
   [returns⇒ [↑self sizeForValue]
   ↑self sizeExceptLast + ((array∘position) sizeForEffect: nextPush)]
```
sizeForTruth: trueSkip falsity: falseSkip
```
   [returns⇒ [↑self sizeForValue]
   ↑self sizeExceptLast + (array∘position sizeForTruth: trueSkip falsity:
falseSkip)]
```
sizeForValue
```
   [↑self sizeExceptLast + (array∘position) sizeForValue + [returns⇒ [1] 0]]
```

Miscellaneous

printon: s | i
```
   [s append: '['.
   for: i to: position-1 do: [s print: (array∘i); append: '. '].
   [returns⇒ [s append: '↑']].
   [position>0⇒ [s print: (array∘position)]]].
```

```
    s append: ']']
quickCode | t v
    [position=1 and: (returns and: (v←array@1) emitsLoad)⇒
        [v=toLoadSelf⇒ [t ← String new: 2. to1←0; o2←1. ↑t];
            < toLoadTemp⇒ [t ← String new: 5. to1←0; o2←40; o3←0; o4←0; o5←v.
↑t]
        ↑false]
    ↑false]
returns
    [↑returns]
```

Private

```
emitExceptLast: code on: stack
        | i
    [for: i to: position-1 do: [(array@i) emitForEffect: code on: stack]]
sizeExceptLast
        | i next nextPush size
    [size ← 0. next ← array@position.
    for: i to: position-1 do:
        [nextPush ← next firstPush. next ← array@(position-i).
        size ← size + (next sizeForEffect: nextPush)].
    ↑size]
```

As yet unclassified

```
emitForTruth: trueSkip falsity: falseSkip into: code on: stack
    [returns⇒ [self emitForValue: code on: stack]
    self emitExceptLast: code on: stack.
    (array@position) emitForTruth: trueSkip falsity: falseSkip into: code on:
stack]
┘
SystemOrganization classify: ↗ParsedBlock under: 'Compiler'.┘
```

"ParsedConditional"

```
Class new title: 'ParsedConditional'
   subclassof: Object
   fields: 'ifExpr thenExpr elseExpr thenSize elseSize jmpSize'
   declare: '';
   asFollows_!
```

I am a node in a compiler parse tree. I represent a condition and two alternatives.

Initialization

```
ifExpr: ifExpr thenExpr: thenExpr elseExpr: elseExpr
```

Code generation

```
emitForEffect: code on: stack
   [ifExpr emitForValue: code on: stack.
   thenSize emitBfp: code on: stack.
   thenExpr emitForEffect: code on: stack.
   [jmpSize>0⇒ [elseSize emitJmp: code on: stack]].
   elseExpr emitForEffect: code on: stack]
emitForValue: code on: stack
   [ifExpr emitForValue: code on: stack.
   thenSize emitBfp: code on: stack.
   thenExpr emitForValue: code on: stack.
   stack pop: 1.
   [jmpSize>0⇒ [elseSize emitJmp: code on: stack]].
   elseExpr emitForValue: code on: stack]
firstPush
   [↑ifExpr firstPush]
sizeForEffect: nextPush
   [elseSize ← elseExpr sizeForEffect: nextPush.
   jmpSize ← [thenExpr returns⇒ [0] elseSize jmpSize].
   thenSize ← (thenExpr sizeForEffect: ¯1) + jmpSize.
   ↑ifExpr sizeForValue + thenSize bfpSize + thenSize + elseSize]
sizeForValue
   [elseSize ← elseExpr sizeForValue.
   jmpSize ← [thenExpr returns⇒ [0] elseSize jmpSize].
   thenSize ← thenExpr sizeForValue + jmpSize.
   ↑ifExpr sizeForValue + thenSize bfpSize + thenSize + elseSize]
```

Miscellaneous

```
printon: s
   [s append: 'if: '; print: ifExpr; append: 'then: '; print: thenExpr; append:
'else: '; print: elseExpr]
returns
   [↑thenExpr returns and: elseExpr returns]
_!
SystemOrganization classify: ↗ParsedConditional under: 'Compiler'._!
```

"ParsedConjunct"

```
Class new title: 'ParsedConjunct'
   subclassof: Object
   fields: 'left right rightSize'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent (left and& right) and try to optimize the code generation thereof.

Initialization

left: left right: right

Code generation

emitForEffect: code on: stack
```
   [left emitForValue: code on: stack.
   rightSize emitBfp: code on: stack.
   right emitForEffect: code on: stack]
```
emitForTruth: trueSkip falsity: falseSkip into: code on: stack
```
   [left emitForTruth: 0 falsity: rightSize+falseSkip into: code on: stack.
   right emitForTruth: trueSkip falsity: falseSkip into: code on: stack]
```
emitForValue: code on: stack
```
   [left emitForValue: code on: stack.
   rightSize emitBfp: code on: stack.
   right emitForValue: code on: stack.
   1 emitJmp: code on: stack.
   code next ← toLoadFalse]
```
firstPush
```
   [↑left firstPush]
```
sizeForEffect: nextPush
```
   [rightSize ← right sizeForEffect: ¯1.
   ↑left sizeForValue + rightSize bfpSize + rightSize]
```
sizeForTruth: trueSkip falsity: falseSkip
```
   [rightSize ← right sizeForTruth: trueSkip falsity: falseSkip.
   ↑(left sizeForTruth: 0 falsity: rightSize+falseSkip) + rightSize]
```
sizeForValue
```
   [rightSize ← right sizeForValue + 1.
   ↑left sizeForValue + rightSize bfpSize + rightSize + 1]
```

Miscellaneous

emittedReceiver
```
   [↑left]
```
emittedReceiver ← left

printon: s
```
   [s append: '('; print: left; append: ' and& '; print: right; append: ')']
↵
SystemOrganization classify: ↗ParsedConjunct under: 'Compiler'.↵
```

"ParsedDisjunct"

```
Class new title: 'ParsedDisjunct'
   subclassof: Object
   fields: 'left right rightSize'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent (left or: right) and try to optimize the code generation thereof.

Initialization

```
left: left right: right
```

Code generation

```
emitForEffect: code on: stack
   [left emitForValue: code on: stack.
   rightSize jmpSize emitBfp: code on: stack.
   rightSize emitJmp: code on: stack.
   right emitForEffect: code on: stack]
emitForTruth: trueSkip falsity: falseSkip into: code on: stack
   [left emitForTruth: rightSize+trueSkip falsity: 0 into: code on: stack.
   right emitForTruth: trueSkip falsity: falseSkip into: code on: stack]
emitForValue: code on: stack
   [left emitForValue: code on: stack.
   (1 + rightSize jmpSize) emitBfp: code on: stack.
   code next ← toLoadTrue.
   rightSize emitJmp: code on: stack.
   right emitForValue: code on: stack]
firstPush
   [↑left firstPush]
sizeForEffect: nextPush
   [rightSize ← right sizeForEffect: ¯1.
   ↑left sizeForValue + 1 + rightSize jmpSize + rightSize]
sizeForTruth: trueSkip falsity: falseSkip
   [rightSize ← right sizeForTruth: trueSkip falsity: falseSkip.
   ↑(left sizeForTruth: rightSize+trueSkip falsity: 0) + rightSize]
sizeForValue
   [rightSize ← right sizeForValue.
   ↑left sizeForValue + 2 + rightSize jmpSize + rightSize]
```

Miscellaneous

```
emittedReceiver
   [↑left]
emittedReceiver ← left
printon: s
   [s append: '('; print: left; append: ' or '; print: right; append: ')']
↵
SystemOrganization classify: ↷ParsedDisjunct under: 'Compiler'.↵
```

```
                                                    "ParsedFieldReferenc
                                                    e"
Class new title: 'ParsedFieldReference'
   subclassof: Object
   fields: 'var toLoadVar toLoadFieldReference toObjectOffset'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent a remote argument which is a reference to a method or instance variable.

Initialization

```
var: var
```

Code generation

```
emitForValue: code on: stack
   [([var isField⇒ [toLoadSelf] toLoadTempframe]) emitForValue: code on: stack.
   toLoadVar emitForValue: code on: stack.
   toLoadFieldReference emitForValue: code on: stack.
   code next ← toNew. toObjectOffset emitBytes: code..
   stack pop: 2]
local
   [↑var]
remote: generator
   [toLoadVar ← generator literal: (var land: 0177)+1.
   toLoadFieldReference ← generator literal: FieldReference.
   toObjectOffset ← generator encodeSel: ↶object:offset:]
sizeForValue
   [↑ 2 + toLoadVar sizeForValue +
      toLoadFieldReference sizeForValue + toObjectOffset sizeForValue]
```

Miscellaneous

```
printon: s
   [s append: 'FLD=> '; print: var]
↵
SystemOrganization classify: ↶ParsedFieldReference under: 'Compiler'.↵
```

"ParsedLoop"

```
Class new title: 'ParsedLoop'
   subclassof: Object
   fields: 'whileExpr doExpr whileSize doSize'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent that part of an in-line loop statement that can be expressed in the while-do form.

Initialization

```
whileExpr: whileExpr doExpr: doExpr
```

Code generation

```
emitForEffect: code on: stack
   [whileExpr emitForTruth: 0 falsity: doSize into: code on: stack.
   doExpr emitForEffect: code on: stack.
   0 - doSize - whileSize emitJmp: code on: stack]
emitForValue: code on: stack
   [self emitForEffect: code on: stack.
   toLoadNil emitForValue: code on: stack]
firstPush
   [↑whileExpr firstPush]
sizeForEffect: nextPush
   [doSize ← (doExpr sizeForEffect: ¯1) + 2.
   whileSize ← whileExpr sizeForTruth: 0 falsity: doSize.
   ↑whileSize + doSize]
sizeForValue
   [↑(self sizeForEffect: ¯1) + 1]
```

Miscellaneous

```
printon: s
   [s append: 'while: '; print: whileExpr; append: 'do: '; print: doExpr]
↵
SystemOrganization classify: ↷ParsedLoop under: 'Compiler'.↵
```

"ParsedMessage"

```
Class new title: 'ParsedMessage'
   subclassof: Object
   fields: 'rcvr op args "false if no args, Vector if many args"'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent an expression consisting of a receiver (rcvr), a selector byte code (op), and an argument list (args) which is false for no arguments, a vector of parse trees for 2 or more arguments, or the argument parse tree for one argument.

Initialization

```
rcvr: rcvr op: op args: args
   [op=toEq and: ((toLoadFalse≡rcvr) or: (toLoadFalse≡args))⇒
      [↑ParsedNegation new rcvr: rcvr op: op args: args]]
```

Code generation

```
emitForEffect: code on: stack
   [self emitForValue: code on: stack. code next ← toPop. stack pop: 1]
emitForValue: code on: stack
   [args emitForValue: code on: stack.
   rcvr emitForValue: code on: stack.
   [rcvr≡toSuper⇒ [code next←rcvr]].
   op emitBytes: code. args argsOff: stack]
firstPush
   [↑([args⇒ [args] rcvr]) firstPush]
sizeForEffect: nextPush
   [↑self sizeForValue+1]
sizeForValue
   [↑args sizeForValue + rcvr sizeForValue + op sizeForValue + [rcvr≡toSuper⇒
[1]0]]
```

Miscellaneous

```
emittedReceiver
   [↑rcvr]
emittedReceiver ← rcvr
printon: s
   [s append: '('; print: rcvr; space; print: op.
   [args⇒ [s space; print: args]].
   s append: ')']
↵
SystemOrganization classify: ↶ParsedMessage under: 'Compiler'.↵
```

"ParsedNegation"

```
Class new title: 'ParsedNegation'
   subclassof: ParsedMessage
   fields: ''
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I am a parsed messsage in which the selector is ≡ and one of the participants is false.

Initialization

rcvr: rcvr op: op args: args

Code generation

emitForTruth: trueSkip falsity: falseSkip into: code on: stack

```
   [([toLoadFalse≡rcvr⇒ [args] rcvr]) emitForTruth: falseSkip falsity: trueSkip
into: code on: stack]
```

sizeForTruth: trueSkip falsity: falseSkip

```
   [↑([toLoadFalse≡rcvr⇒ [args] rcvr]) sizeForTruth: falseSkip falsity: trueSkip]
```

Miscellaneous

printon: s

```
   [s append: '(negation)'. super printon: s]
↵
SystemOrganization classify: ↗ParsedNegation under: 'Compiler'.↵
```

"ParsedObjectRefere
nce"

```
Class new title: 'ParsedObjectReference'
   subclassof: Object
   fields: 'var'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent a remote argument which is a reference to a class or pool variable.

Initialization

```
var: var
```

Code generation

```
emitForValue: code on: stack  "Turn literal indirect into literal direct"
   [(var-256) emitForValue: code on: stack]
local
   [↑var]
remote: generator
sizeForValue
   [↑(var-256) sizeForValue]
```

Miscellaneous

```
printon: s
   [s append: 'OBJ=> '; print: var]
↵
SystemOrganization classify: ↷ParsedObjectReference under: 'Compiler'.↵
```

"ParsedRemote"

```
Class new title: 'ParsedRemote'
   subclassof: Object
   fields: 'expr esize toRemoteCopy'
   declare: '';
   sharing: ByteCodes;
   asFollows↵
```

I am a node in a compiler parse tree. I represent an argument that is to be passed unevaluated.

Initialization

```
expr: expr
```

Code generation

```
emitForValue: code on: stack
   [toLoadThisCtxt emitForValue: code on: stack.
   toRemoteCopy emitBytes: code.
   code emitLong: toLongJmp by: esize.
   expr emitForValue: code on: stack.
   code next ← toEnd. stack pop: 1.
   (0-esize) emitJmp: code on: stack]
local
   [↑expr]
remote: generator
   [toRemoteCopy ← generator encodeSel: #remoteCopy]
sizeForValue
   [esize ← expr sizeForValue + 3.
   ↑esize + toRemoteCopy sizeForValue + 3]
```

Miscellaneous

```
printon: s
   [s append: '↑'; print: expr]
↵
SystemOrganization classify: #ParsedRemote under: 'Compiler'.↵
```

"Parser"

```
Class new title: 'Parser'
   subclassof: Object
   fields: 'source dest oppositeCourt type token mark keep'
   declare: '';
   sharing: TokenCodes;
   asFollows_!
```

I parse tokens from source (a stream). I report each parse node to dest (e.g., a code generator). When an error occurs, I back up source to mark and notify dest. I have two kinds of methods, token suppliers and token consumers, which coroutine with each other. The coroutine that is not in control is suspended in the context, oppositeCourt. Token suppliers are driven by an instance of class Reader that scans source and classifies each token by type. Token consumers analyze the syntax and report it to dest. The variable "keep" is no longer used.

Initialization

```
from: source to: dest
   [oppositeCourt ← thisContext.
   mark ← source position. type ← 1.
   (Reader new of: source) readInto: self]
"Generator compile|Generator evaluate"
```

Token suppliers

```
comment: s
   [mark ← source position]
contents
   [type ← 0. mark ← source position + 1.
   thisContext sender ← nil.
   while: true do: [self resume. self notify: 'MORE EXPECTED']]
float: i fraction: f exp: e
   [token ← (i+'.'+f+'e'+e) asFloat.
   type ← aNumber. self resume]
identifier: token
   [type ← aWord. self resume]
integer: s
   [token ← s asInteger.
   type ← aNumber. self resume]
keyword: token
   [type ← aKeyword. self resume]
leftparen
   [type ← aLeftPar. self resume]
onechar: token
   [type ←
      [token=056⇒ [aPeriod];
         =0133⇒ [aLeftBrack];
         =0135⇒ [aRightBrack];
         =033⇒ [aCondArrow];
         =0137⇒ [aLeftArrow];
```

```
        =021⇒ [aReturnArrow];
        =073⇒ [aSemicolon];
        =017⇒ [aHand]
      aBinary].
  self resume]
otheratom: token
  [type ← aGibberish. self resume]
rightparen
  [type ← aRightPar. self resume]
separator: c
string: token
  [type ← aString. self resume]
trailer: s
  [mark ← source position]
```

Method syntax

```
body: block  | p  "return the primitive number, or 0"
  [type=aLeftBrack⇒
    [self block: block.
    type=aKeyword and: token='primitive:'⇒
      [self advance. type=aNumber⇒ [p ← token. self advance. ↑p]
      self notify: 'EXPECTED A NUMBER']
    ↑0]
  ↑0]
"Generator compile"
declaration: block
  [type≠aWord⇒ [self notify: 'EXPECTED AN ARGUMENT NAME']
  dest declaration: block name: token asArg: true.
  self advance]
"pattern"
pattern: block
      | selector
  [selector ← Stream default.
  [type=aWord⇒
      [selector append: token. self advance];
    =aBinary⇒
      [selector append: (LIST1 o(token+1)). self advance; declaration: block]
  while: type = aKeyword do: [selector append: token. self advance;
declaration: block].
  selector empty⇒ [self notify: 'EXPECTED A SELECTOR']].
  [type=aLeftArrow⇒
    [selector append: '←'. self advance; declaration: block]].
  ↑selector contents unique]
"Generator compile/Context variableNamesInto"
temporaries: block
  [type=aBinary and: token=0174⇒ "|"
    [self advance.
    while: type=aWord do:
      [dest declaration: block name: token asArg: false. self advance]]]
"Generator compile/Context variableNamesInto"
```

Statement syntax

```
alternatives: ifExpr "⇒ [..] ..."
        | thenExpr elseExpr
    [self advance.
    type≠aLeftBrack⇒ [self notify: 'EXPECTED A [BLOCK]']
    thenExpr ← self block: dest block. elseExpr ← dest block.
    [type=aSemicolon⇒ [self cascade: elseExpr after: ifExpr] self statements:
elseExpr].
    ↑dest ifExpr: ifExpr thenExpr: thenExpr elseExpr: elseExpr]
"cascade"
block: block
    [self advance; statements: block.
    type=aRightBrack⇒ [self advance. ↑block]
    self notify: 'PERIOD OR RIGHT BRACKET WAS EXPECTED']
"body/alternatives/expression/keywordMessage/binaryMessage"
cascade: block after: expr  | val var oldTemps
    [var ← dest receivingVar: expr.
    oldTemps ← dest balance.
    while§ type=aSemicolon do§
        [self advance.
        (val ← self messageChain: var)≡var⇒
            [self notify: 'MESSAGE EXPECTED'].
        type=aCondArrow⇒
            [block next ← self alternatives: val. dest unbalance: oldTemps. ↑self]
        block next ← val].
    dest unbalance: oldTemps]
"statement/alternatives"
loopStmt: block  | oldMark
    [oldMark ← mark. self keywordMessage: false loop: block⇒ [↑self]
    mark ← oldMark. self notify: 'UNKNOWN CONTROL MESSAGE']
"statement"
macro: block  | oldMark oldTemps val
    [oldMark ← mark. oldTemps ← dest juggle.
    val ← self keywordMessage: false macro: block.
    dest unjuggle: oldTemps.
    val⇒ [↑block]
    mark ← oldMark. self notify: 'UNKNOWN CONTROL MESSAGE']
"statement"
statement: block | expr
    [type=aReturnArrow⇒
            [self advance. block next ← self expression. block doesReturn.
            [type=aPeriod⇒ [self advance]].
            type≠aRightBrack⇒ [self notify:'SHOULDN''T FOLLOW RETURN']];
        = aKeyword⇒ [self macro: block. type>aPeriod⇒ [self statement: block]];
        ≤aPeriod⇒ [dest nullStatement: block] "doit eof aRightBrack aPeriod"
    expr ← self expression.
    type=aCondArrow⇒ [block next ← self alternatives: expr]
    block next ← expr.
    type=aSemicolon⇒ [self cascade: block after: expr]]
"statements"
statements: block
    [self statement: block.
```

```
    while: type=aPeriod do:
        [self advance. self statement: block]]
"alternatives/block/Generator evaluate"
```

Expression syntax

```
binaryMessage: rcvr assign: assign "binarySelector ..."
        | sel args
    [sel ← token. self advance.
    args ← [type=aLeftBrack⇒ [self block: dest block] self factor].
    [assign and: type=aLeftArrow⇒
        [self advance.
        args ← [(Vector new: 2)o1←args; o2←self expression; itself].
        sel ← [(String new: 2)o1←sel; o2←0137"←"; itself]]].
    ↑dest rcvr: rcvr selector: sel args: args]
"term/messageChain"
expression
        | var
    [type=aLeftBrack⇒ [↑self block: dest block];
            =aKeyword⇒ [↑self macro: dest block];
            ≠aWord⇒ [↑self messageChain: self primary]
    "It begins with a variable name" var ← dest variable: token. self advance.
    type≠aLeftArrow⇒ [↑self messageChain: var]
    "It is a variable assignment" self advance.
    ↑dest assignment: var expr: self expression]
"unaryMessage/binaryMessage/keywordMessage/statement/subExpression"
factor
        | expr
    [expr ← self primary.
    while: type=aWord do: [expr ← self unaryMessage: expr assign: false].
    ↑expr]
"term/binaryMessage"
keywordMessage: rcvr macro: block "keyword ..."
        | sel args arg
    [sel ← Stream default. args ← (Vector new: 4) asStream.
    while: type = aKeyword do:
        [sel append: token. self advance.
        arg ← [type=aLeftBrack⇒ [self block: dest block] self term].
        args next ← [sel last=03⇒ [dest noEvalKeyword: arg] dest evalKeyword:
arg]].
    [type=aLeftArrow⇒ [sel append: '←'. args next ← [self advance; expression]]].
    sel ← sel contents. args ← [args position=1⇒ [args last] args contents].
    block⇒ [↑dest macro: block selector: sel args: args]
    ↑dest keywordMessage: rcvr selector: sel args: args]
"macro/messageChain"
messageChain: rcvr
    [while: type=aWord do: [rcvr ← self unaryMessage: rcvr assign: true].
    while: type=aBinary do: [rcvr ← self binaryMessage: rcvr assign: true].
    [type = aKeyword⇒ [rcvr ← self keywordMessage: rcvr macro: false]].
    ↑rcvr]
"cascade/expression"
term | rcvr
    [rcvr ← self factor.
```

```
    whileΣ type=aBinary doΣ [rcvr ← self binaryMessage: rcvr assign: false].
    ↑rcvr]
"keywordMessage"
unaryMessage: rcvr assign: assign "word ..."
        | sel args
    [sel ← token. self advance.
    args ←    [assign andΣ type=aLeftArrow⇒ [sel ← sel + '←'. self advance;
expression] false].
    ↑dest rcvr: rcvr selector: sel args: args]
"factor/messageChain"
```

Primary syntax

```
literal "A Vector, UniqueString, String, or Number"
        | t oldMark
    [type
        =aLeftPar⇒
            [oldMark ← mark. self advance.
            t ← self read. type=aRightPar⇒ [self advance. ↑t]
            mark ← oldMark. self notify: 'UNMATCHED']
    t ← [type≥aKeyword⇒ [token unique]; ≤aBinary⇒ [UST1∘(token+1)] token].
    self advance. ↑t]
"primary/read"
primary
        | t
    [type
        =aWord⇒ [t ← dest variable: token. self advance. ↑t];
        =aLeftPar⇒ [↑self subExpression];
        =aNumber⇒ [t ← dest literal: token. self advance. ↑t];
        =aString⇒ [t ← dest literal: token. self advance. ↑t];
        =aHand⇒ [self advance.
            type=aRightPar orΣ type=0⇒ [self notify: 'EXPECTED LITERAL']
            ↑dest literal: self literal]
    self notify: 'OBJECT EXPECTED']
"factor/expression"
read "A sequence of literals"
        | s
    [s ← (Vector new: 10) asStream.
    untilΣ (type=aRightPar orΣ type=0) doΣ [s next ← self literal].
    ↑s contents]
"literal"
subExpression "(...)"
        | expr
    [self advance. expr ← self expression.
    type=aRightPar⇒ [self advance. ↑expr]
    self notify: 'NOT EXPECTED IN A (SUBEXPRESSION)']
"primary"
```

Suspension

```
advance "Switch from the parser to the reader to obtain another token."
    [mark ← source position- [type>aBinary⇒ [1]0].
    oppositeCourt ← thisContext swapSender: oppositeCourt]
```

```
mustBeDone
   [type=0⇒ [self terminate] self notify: 'UNEXPECTED CONSTRUCT']
"Generator compile|Generator evaluate"
notify: errorString | delims
   [source skip: mark - source position.
   delims ← ↗(011 012  014 015 040).
   while: (delims has: source peek) do: [source next].
   [source myend≡false⇒ [source skip: 1]].
   dest abortWith: errorString]
resume "The reader has supplied another token; resume the parser."
   [oppositeCourt ← thisContext swapSender: oppositeCourt]
terminate
   [[dest≡nil⇒ [] dest terminate. dest ← nil].
   [oppositeCourt≡nil⇒ [] oppositeCourt release. oppositeCourt ← nil]]
"mustBeDone|Generator abortWith|Context variableNamesInto"
┘
SystemOrganization classify: ↗Parser under: 'Compiler'.┘
```

"ParseStack"

```
Class new title: 'ParseStack'
   subclassof: Object
   fields: 'position length'
   declare: '';
   asFollows↵
```

I keep track of the current and high position of the stack that will be needed by code being compiled.

Initialization

```
init
   [length ← position ← 0]
```

Changes

```
pop: n
   [(position ← position - n) < 0⇒ [user notify: 'Parse stack underflow']]
push: n
   [(position ← position + n) > length⇒ [length ← position]]
```

Results

```
length
   [↑length]
position
   [↑position]
↵
SystemOrganization classify: ↶ParseStack under: 'Compiler'.↵
```

```
                                                        "BravoPrinter"
Class new title: 'BravoPrinter'
   subclassof: ParagraphPrinter
   fields: 'eject "Eject page before next paragraph if true"
      '
   declare: '';
   asFollows↵

Prints Paragraphs in Bravo format

Initialization
init
   [super init. eject ← false]

Writing
eject
   [strm next ← 014; cr]
nextpage
   [eject⇒ [self eject] eject ← true]
print: para | runs l r
   [[eject⇒ [self eject. eject ← false]].
   strm append: para text.
   runs ← para bravoRuns.
   l ← frame origin x.
   r ← frame corner x.
   strm next← 032.        "↑Z"
   [l≠self defaultframe origin x⇒
      [strm append: 'l'; print: l]].
   [r≠self defaultframe corner x⇒
      [strm append: 'z'; print: r]].
   [leading≠self defaultleading⇒
      [strm append: 'e'; print: leading]].
   runs ← (runs∘(2 to: runs length-1)) copy.          "Drop ↑Z and CR"
   "Transform runs for compatibility with old filin"
   [runs='\gi'⇒ [runs ← '\ig'];
      ='\gbf5 '⇒ [runs ← '\f5bg']].
   strm append: runs; cr.
   ]
↵
SystemOrganization classify: ↷BravoPrinter under: 'Paragraph printing'.↵
```

"ParagraphPrinter"

```
Class new title: 'ParagraphPrinter'
   subclassof: Object
   fields: 'frame "<Rectangle> usable area on page"
      leading "<Integer> paragraph leading"
      style "<TextStyle> for paragraphs"
      strm "<Stream> for output"
      '
   declare: 'defaultleading defaultframe ';
   asFollows↓
```

Provides a stream-like interface for printing a succession of paragraphs on a Bravo or Press file. The margins, leading, and style are settable instance variables. BravoPrinter and PressPrinter each override some messages

Initialization

```
classInit | inch
   [inch ← 2540.        "1 inch in micas"
   defaultframe ←
      (0.75*inch) asInteger⊙(1*inch) rect: (7.75*inch) asInteger⊙(10*inch).
   defaultleading ← 0]
init
   [self frame ← self defaultframe.
   self leading ← defaultleading.
   self style ← DefaultTextStyle]
of: strm
```

Access to state

```
defaultframe [↑defaultframe]
defaultleading [↑defaultleading]
frame [↑frame]
frame ← frame
leading ← leading
style ← style
```

Writing

```
print: para        "A dummy, subclasses will override"
   [strm append: para text]
```

Class stuff

```
printchanges: lis | s selector class old heading h
   [for: s from: lis do:
      [user show: s; cr.
      s ← s asStream.
      class ← Smalltalk∘(s upto: 040) unique.
      selector ← (s upto: 040) unique.
         [class=old→[]
            [old≡nil→[] old endCategoryOn: self; endChangesOn: self].
         class startChangesOn: self.
```

```
        old ← class.heading ← 'As yet unclassified'].
      h ← class organization invert: selector.
      [h≠heading⇒[class startCategory: (heading ← h) on: self]].
      class printMethod: selector on: self].
  class≠nil⇒ [class endCategoryOn: self; endChangesOn: self]]
printclass: class | c
  [class is: Vector⇒
    [for: c from: class do:
      [self printclass: c; nextpage]]
  [class is: UniqueString⇒
    [class ← Smalltalk∘class]].
  class paraprinton: self]
stamp | s t [
  t ← user now "date and time".
  s ← Stream default.
  s append: '''From '; append: user version;
    append: ' on '; print: to1;
    append: ' at '; print: to2;
    append: '.''␊'; cr.
  self print: s contents asParagraph]
```

Closing

```
close [strm shorten; close]
␊
SystemOrganization classify: #ParagraphPrinter under: 'Paragraph printing'.␊
ParagraphPrinter classInit␊
```

"PressPrinter"

```
Class new title: 'PressPrinter'
   subclassof: ParagraphPrinter
   fields: 'page "<Integer> current page number"
      ypos "<Integer> current y position on page"
      press "<PressFile> for output"
      '
   declare: 'defaultframe ';
   asFollows_!
```

Prints Paragraphs in Press format

Initialization

```
classInit | inch
   [inch ← 2540.        "1 inch in micas"
   defaultframe ←
      (1.1*inch) asInteger⊙(1*inch) rect: (7.75*inch) asInteger⊙(10*inch)]
defaultframe [↑defaultframe]
init [super init. page ← 1]
press: press
```

Writing

```
nextpage [self nextpage: true]
nextpage: h | n [
   press page.
   page ← page+1.
   ypos ← frame maxY.
   h⇒ [
      n ← page asString.
      press setp: frame maxX ⊙ (ypos + 160);
         selectfont: (press fontindex: 0 style: DefaultTextStyle) - 1;
         append: n;
         showchars: n length;
         closeEntity: 0]]
print: para [
   "First, some deferred initialization"
   [press≡nil⇒ [press ← PressFile new of: strm]].
   [ypos≡nil⇒ [ypos ← frame maxY]].
   self print: para in: (
      Rectangle new origin: frame origin corner: frame maxX ⊙ ypos)]
print: para in: rect | result i t r [
   i ← 1.
   "should be same for para and leftover pieces"
   leading ← (para style indentation * press scale) asRectangle.
   while$ [
      r ← rect +
         (leading minX ⊙ 0 rect:
         leading maxX ⊙ [
         frame maxY = rect maxY⇒ ["no leading at top?" 0]
         0 - leading minY]).
```

```
        press entityorigin: r minX ⊙ 0.
        result ← para presson: press in: r.
        t ← result is: para class.
        para hidePress: press first: i
            length: para length - [t⇒ [result length] 0].
        t]
    do: [
        i ← i+1.
        self nextpage.
        para ← result.
        rect ← rect minX ⊙ frame minY rect:
            rect maxX ⊙ frame maxY].

    "add bottom leading"
    ypos ← result + leading maxY]
```

Closing

```
close [press close]
press [↑press]
toPrinter [press toPrinter]
```

Projector behavior

```
rectOf: r [
    "projector-like transformation, but rectangle extends to bottom of page"
    ↑Rectangle new
        origin: (self xOf: r minX) ⊙ frame minY
        corner: (self xOf: r maxX) ⊙ (self yOf: r minY)]
xOf: x [↑frame minX + (x * press scale)]
yOf: y [↑frame maxY - (y * press scale)]
ypos [↑ypos]
↵
SystemOrganization classify: ↶PressPrinter under: 'Paragraph printing'.↵
PressPrinter classInit↵
```

"BitRect"

```
Class new title: 'BitRect'
   subclassof: Rectangle
   fields: 'title "<String> title of picture"
      stripheight "<Integer> scan lines in a buffer (private)"
      data "<Vector> of Strings. Saves the bits in the Rectangle"'
   declare: 'defaultpic ';
   asFollows_!
```

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 and starts it up.
The editor is explained in BitRectEditor.

Initialization

```
classInit
   ["the default picture is a gray rectangle"
   defaultpic ← BitRect new filin: 'defaultpic']
default [↑defaultpic recopy]
fromuser
   [self title: 'BitRect' in: Rectangle new fromuser.
   self saveScreenBits]
origin: origin corner: corner title: title stripheight: stripheight data: data
title: title in: rect | nStrips i strips
   [origin←rect origin. corner←rect corner.
   "the strip height is chosen so that each bitstring is about 2048 bytes"
   stripheight←1023/((self extent x + 15)/16).
   nStrips←(self extent y+stripheight-1)/stripheight.
   data←Vector new: nStrips.
   strips←self strips.
   for: i to: nStrips do:
      [data↓i←String new: (strips↓i) bitStringLength]]
```

Access to parts

```
data [↑data]
title [↑title]
```

Rectangle Protocol

```
= x [↑self=x]
bitsOntoStream: strm | bits
   [for: bits from: data do: [strm append: bits]]
corner←x [self growby: x-corner]
extent←x [self growby: x-self extent]
growby: change | old
   [old←BitRect new origin: origin corner: corner title: title
      stripheight: stripheight data: data.
   self title: title in: (origin rect: corner+change).
   self copyBitsFrom: old]
growto: x [self growby: x-corner]
```

```
height←h [self growby: 0⊙(h-self extent y)]
printon: strm
   [strm append: 'a BitRect']
width←w [self growby: (w-self extent x)⊙0]
```

Editing

```
copyBitsFrom: other
      | clippedStrip i j myStrips otherStrips myStrip otherStrip
   ["copy all bits from other that are within my area"
   myStrips←self strips.  otherStrips←other strips.
   for: i to: myStrips length do:
      [for: j to: otherStrips length do:
         [myStrip←myStrips∘i.  otherStrip←otherStrips∘j.
         clippedStrip←myStrip intersect: otherStrip.
         clippedStrip empty⇒[]
         BitBlt init function←0;
            destbase←data∘i;
            destraster←myStrip width+15/16;
            dest←clippedStrip origin-myStrip origin;
            extent←clippedStrip extent;
            sourcebase←other data∘j;
            sourceraster←otherStrip width+15/16;
            source←clippedStrip origin-otherStrip origin;
            checksandcall]]]
edit | a
   [user topWindow leave.
   a←BitRectEditor new picture: self.
   a takeCursor; enter.
   user restartup: a]
```

Showing

```
saveScreenBits | strips i
   [strips←self strips.
   for: i to: strips length do:
      [strips∘i bitsIntoString: data∘i mode: storing clippedBy: nil]]
show | strips i
   [strips←self strips.
   for: i to: strips length do:
      [strips∘i bitsFromString: data∘i]]
strips   "return a vector of strips (Rectangles)"
      | nStrips strips stripOrigin stripExtent i
   [(nStrips←data length)=1⇒[↑self inVector]
   strips←Vector new: nStrips.
   stripOrigin←origin.  stripExtent←self width⊙stripheight.
   for: i to: nStrips-1 do:
      [strips∘i←Rectangle new origin: stripOrigin extent: stripExtent.
      stripOrigin←stripOrigin+(0⊙stripheight)].
   strips∘nStrips←Rectangle new origin: stripOrigin corner: corner.
   ↑strips]
```

Filin and filout

```
filin: title | f i x y rect strips "read bits from a file"
   [f←dp0 file: (title concat: '.pic.').
   f end⇒[user notify: 'no data']
   x←f nextword. y←f nextword.
   rect←Rectangle new origin: [origin is: Point⇒[origin] 0⊙0] extent: x⊙y.
   self title: title in: rect.
   stripheight≠f nextword⇒[user notify: 'strip heights dont match']
   strips ← self strips.
   for: i to: strips length do:
      [f into: data∘i].
   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]
```

Projectors

```
deselected
   ["I dont care"]
growthThru: proj within: domain
   [↑self height-domain height]
saveScreenBitsFrom: projector within: domain
      | strips i
   [strips←self strips: domain origin. "domain origin is the offset"
   for: i to: strips length do:
      [projector saveBits: data∘i in: strips∘i clippedBy: domain]]
selected
   ["I dont care"]
selectionSpecies [↑BitRectSelection]
showThru: projector within: domain [
   projector paint: white in: domain with: storing;
   "domain origin is the offset"
      showBitRect: self in: (self strips: domain origin) clippedBy: domain]
strips: offset "return a vector of strips (Rectangles) offset by offset"
      | nStrips strips stripOrigin stripExtent i
   [(nStrips←data length)=1⇒[↑(self+offset) inVector]
   strips←Vector new: nStrips.
   stripOrigin←origin+offset. stripExtent←self width⊙stripheight.
   for: i to: nStrips-1 do:
      [strips∘i←Rectangle new origin: stripOrigin extent: stripExtent.
      stripOrigin←stripOrigin+(0⊙stripheight)].
   strips∘nStrips←Rectangle new origin: stripOrigin corner: corner+offset.
   ↑strips]
```

Press

```
complete []
fromPress: press name: code value: s [
   "use control info from a press file to construct instance"
```

```
    code
    =10⇒ [
        s ← s viewer.
        "width and height"
        self title: title in:
            (Rectangle new
                origin: [origin is: Point⇒[origin]0⊙0]
                extent: s nextPoint).

        "number of strips.
        refers to same bits just read by showdots (a kludge)"
        stripheight ≠ s nextword⇒ [user notify: 'strip heights dont match']
        (press ← press data) skip: 0 - self bitStringLength.
        for: s to: data length do: [press into: data∘s]]

    ↑false]
hidePress: press first: n length: len | s [

    len=0⇒ []
    (s ← Set new string: 6)
        nextPoint ← self extent;
        nextword ← stripheight.
    press skipcode: 10 data: s;
        closeEntity]
length [↑self bitStringLength]
presson: press in: r | w h hs scale w16 y [
    scale ← press scale.
    h ← self height.
    (hs  ← scale*h) > r height⇒ [
        "not enough room left on current page.
        assume for now that it will at least fit on an entire page"
        ↑self]

    w ← self width.
    w16 ← w + 15 | 16 "width to next word boundary".
    "with w, prints on viola but not on spruce.
    with w16, prints on spruce with garbage on end"
    press setp: 0⊙(y ← r corner y - hs);
        dots: [
            press setcoding: 1 "bitmap" dots: w16 lines: h;
                setmode: 3 "to right and to bottom";
                setsizewidth: scale*w16 height: hs;
                setwindowwidth: w16 height: h;
                dotsfollow.
            self bitsOntoStream: press data].
    ↑y]
style ["for leading info" ↑DefaultStyle]
_!
SystemOrganization classify: ↗BitRect under: 'Picture Editor'._!
BitRect classInit_!
```

"BitRectEditor"

```
Class new title: 'BitRectEditor'
   subclassof: Window
   fields: 'tool  "<BitRectTool> the current tool"
      picture  "<BitRect> the picture we are working on"
      dirty  "false if picture has not been modified"
      saveActionPic saveToolPic  "buffers for saving background" '
   declare: 'tools toolpic actionbuttons actionpic windowmenu ';
   asFollows_!
```

BitRectEditor edits BitRects.
To create, say:
BitRect new fromuser edit.
This installs a BitRectEditor in the scheduler and starts it up.
The editing tools are to the left of the picture. (The first one looks like a doodle). They are: draw-thin, erase, straightedge, gray-block, paintbrush, magnifier. The actions for the tools are displayed above the picture.
See BitRectTool for explanations of the actions.

CAUTION: this ordering is arbitrary. It is currently possible to set a new action for any of the tools, so that if you are not careful, the straightedge will start being a magnifier or whatever. This should get fixed eventually.

tools = a RadioButtons. Each button owns a BitRectTool (the active one is held in tool).
actionbuttons = a Vector of RadioButtons. The groups of buttons are: action, mode, pen width, gray, and grid.
toolpic = BitRect of icons for the tools (at side of picture).
actionpic = BitRect of icons for the parts of a tool (above picture)
windowmenu = menu for bluebug.

To edit a copy of the tool picture, say
newpic←(BitRectEditor∘↗toolpic) recopy.
newpic edit.
To install this copy as the menu picture, say
BitRectEditor new toolpic: newpic recopy.
Do the analogous thing to edit the action picture.
Caution: the editor blows up if you edit the tool picture itself and not a copy.

Initialization

```
actionpic: a [actionpic ← a]
classInit | 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
```

```
printbits'.
   actionpic←BitRect new filin: 'actionpic'.
   toolpic←BitRect new filin: 'toolpic'.
   self initmenu1]
initmenu1 | s z
   [s ← Vector new: 5. z ← 20.
   so1 ← (RadioButtons new) vec: c↗(setbrush paint block draw line blowup) at:
0⊙0 height: z. "action"
   so2 ← (RadioButtons new) vec: (black, dkgray, gray, ltgray, white) at: 0⊙0
height: z. "tone"
   so3 ← (RadioButtons new) vec: (0, 1, 2, 3) at: 0⊙0 height: z. "mode"
   so4 ← (RadioButtons new) vec: (1, 2, 4, 8) at: 0⊙0 height: z. "width"
   so5 ← (RadioButtons new) vec: (1, 2, 4, 8, 16, 32) at: 0⊙0 height: z. "grid"
   actionbuttons ← s.]
picture: picture
   [tool ← tools push: 1.
   self frame: (picture origin rect: picture corner)]
toolpic: a [toolpic ← a]
```

Window protocol

```
bluebug |
   [windowmenu bug
    =1 ⇒[self leave. ↑exitflag ← false]; "under"
   =2 ⇒[self leave; newframe; enter]; "move"
   =3 ⇒[self grow "grow"];
   =4 ⇒[self leave; erase. "close"
      user unschedule: self. ↑false];
   =5 ⇒[self leave. picture filout. self enter]; "filout"
   =6 ⇒[self print]    "press file"]
enter | start pt b
   ["Periodically check if the mouse is still in the frame.
      If not, stop showing the picture"
   super show. self lostMouse⇒[↑false]
   picture show. dirty←false. self lostMouse⇒[↑false]
   tools reset.
   for: b from: actionbuttons do: [b reset].
   "show action menu above the picture"
   start←frame origin-1.
   pt ← start-(0⊙actionpic extent y).
   actionpic moveto: pt.
   saveActionPic←actionpic bitsIntoString.
   self lostMouse⇒[↑false]
   "last point I can return before having to restore bits under menus"
   actionpic show.
   pt ← actionbuttonso1 moveto: pt. "action"
   pt ← actionbuttonso3 moveto: pt. "mode"
   pt ← actionbuttonso4 moveto: pt. "width"
   "show the next bank of action buttons"
   pt ← start-(0⊙(actionpic extent y+1/2)).
   pt ← actionbuttonso2 moveto: pt. "tone"
   pt ← actionbuttonso5 moveto: pt. "grid"
   tool brushpt: pt+1.
```

```
    "show tool menu to the left of the picture"
    pt ← start-(toolpic extent x⊙0).
    toolpic moveto: pt.
    saveToolPic←toolpic bitsIntoString.
    toolpic show.
    tools moveto: pt; setvalue: tool. "tools"
    tool frame: frame; showon: actionbuttons]
fixframe: r
    [picture moveto: r origin.
    r corner←picture corner.
    ↑r]
grow | oldframe newframe pt r
    [self leave.
    newframe←picture origin rect: picture corner.
    CornerCursor showwhile:
        [pt←user mp+16.
        while: user nobug do:
            [newframe corner←pt. newframe comp.
            pt←user mp+16. newframe comp].
        while: user anybug do:
            [newframe corner←pt. newframe comp.
            pt←user mp+16. newframe comp]].
    "clear unused areas from old picture to background,
        and clear new picture areas to white"
    oldframe←picture inset: ¯2⊙¯2. "¯2 is for erasing old border"
    for: r from: (oldframe minus: newframe) do: [r clear: background].
    for: r from: (newframe minus: picture) do: [r clear: white].
    picture title: picture title in: newframe; saveScreenBits.
    self frame: newframe; show; takeCursor; enter]
leave
    [[nil≡saveActionPic⇒[]
     actionpic bitsFromString: saveActionPic.
     toolpic bitsFromString: saveToolPic.
     saveActionPic←saveToolPic←nil].
    dirty⇒[picture saveScreenBits. dirty ← false]]
lostMouse [↑(frame has: user mp)≡false]
outside | pt
    [toolpic has: (pt←user mp)⇒
        [user redbug⇒
            [tool←tools bug: pt. tool frame: frame; showon: actionbuttons]]
    actionpic has: pt⇒
        [user redbug⇒
            [tool setfrom: actionbuttons]]
    ↑false]
redbug [dirty←true. tool redbug]
title [↑picture title]
↵
SystemOrganization classify: ↶BitRectEditor under: 'Picture Editor'.↵
BitRectEditor classInit↵
```

"BitRectSelection"

```
Class new title: 'BitRectSelection'
   subclassof: Selection
   fields: 'tool  "<BitRectTool> the current tool"
      dirty  "false if entity has not been modified"
      saveActionPic saveToolPic  "buffers for saving background" '
   declare: 'tools toolpic editmenu actionbuttons actionpic ';
   asFollows_!
```

BitRectSelections edit BitRects in documents. The selection is always the entire BitRect.
The editing tools are to the left of the picture. (The first one looks like a doodle). They are: draw-thin, erase, straightedge, gray-block, paintbrush, blowup. The actions for the tools are displayed above the picture. See BitRectTool for explanations of the actions.

CAUTION: this ordering is arbitrary. It is currently possible to set a new action for any of the tools, so that if you are not careful, the straightedge will start being an eraser or whatever. This should get fixed.

tools = a RadioButtons. Each button owns a BitRectTool (the active one is held in tool).
actionbuttons = a Vector of RadioButtons. The groups of buttons are: action, mode, pen width, gray, and grid.
toolpic = BitRect of icons for the tools (at side of picture).
actionpic = BitRect of icons for the parts of a tool (above picture)
editmenu = menu for yellowbug.

To edit a copy of the tool picture, say
*   newpic←(BitRectSelection∘↗toolpic) recopy.*
*   newpic edit.*
To install this copy as the menu picture, say
*   BitRectSelection new toolpic: newpic recopy.*
Do the analogous thing to edit the action picture.
Caution: the editor blows up if you edit the tool picture itself and not a copy.

Selection Protocol

```
complement []
of: entity formerly: oldEntity at: index from: pt thru: proj
   [doc←proj slide. self edit]
```

Editing

```
edit  "flop out my menus and edit my entity until mouse is lost"
      | domain screenRectOfEntity pt keepEditing growthFlag
   [domain←doc frameOf: index.
   screenRectOfEntity←proj visibleScreenRectOf:
      (domain intersect: (entity+domain origin)).
   dirty←false. [nil≡tool⊃[tool←tools vec∘1]].
   tool frame: screenRectOfEntity.
   self showMenus: screenRectOfEntity origin.
```

```
    keepEditing←true. growthFlag←false.
    while: keepEditing do:
      [user bluebug⇒[keepEditing←false]
      user yellowbug⇒
        [editmenu bug=1⇒
          [self grow. growthFlag←true. keepEditing←false]]
      user redbug⇒
        [pt←user mp.
        screenRectOfEntity has: pt⇒
          [dirty←true. tool redbug]
        toolpic has: pt⇒
          [tool ← tools bug: pt. tool frame: screenRectOfEntity.
          tool showon: actionbuttons]
        actionpic has: pt⇒
          [tool setfrom: actionbuttons]
        "lost mouse - return control"
        keepEditing←false]]
    self hideMenus.
    [dirty⇒[entity saveScreenBitsFrom: proj within: domain.
          dirty←false]].
    "if I grew, re-show the document"
    growthFlag⇒
      [doc showEdited: index in: proj.
      proj clear. doc showThru: proj]]
grow  "kludge - grow by getting new bits from screen"
      | domain visibleDomain newframe p
    [domain←doc frameOf: index.
    visibleDomain←proj visibleScreenRectOf: domain.
    newframe←visibleDomain recopy.
    CornerCursor showwhile:
      [while: user nobug do:
        [newframe corner←user mp.
        newframe flash].
      while: (p ← user mpnext) do:
        [newframe corner←p.
        newframe flash]].
    newframe clear: white.
    entity showThru: proj within: domain.
    entity title: entity title in: newframe; saveScreenBits.
    entity moveto: newframe origin-visibleDomain origin.
    dirty←false]
hideMenus
    [actionpic bitsFromString: saveActionPic.
    toolpic bitsFromString: saveToolPic.
    saveActionPic←saveToolPic←nil]
showMenus: screenOrigin | pt b
    [tools reset.
    for: b from: actionbuttons do: [b reset].
    "show action menu above the picture"
    pt ← screenOrigin - (0⊙actionpic extent y).
    actionpic moveto: pt.
    saveActionPic←actionpic bitsIntoString.
```

```
    actionpic show.
    pt ← actionbuttons◦1 moveto: pt.  "action"
    pt ← actionbuttons◦3 moveto: pt.  "mode"
    pt ← actionbuttons◦4 moveto: pt.  "width"
    "show the next bank of action buttons"
    pt ← screenOrigin - (0⊙(actionpic extent y+1/2)).
    pt ← actionbuttons◦2 moveto: pt.  "tone"
    pt ← actionbuttons◦5 moveto: pt.  "grid"
    tool brushpt: pt+1.
    "show tool menu to the left of the picture"
    pt ← screenOrigin - (toolpic extent x⊙0).
    toolpic moveto: pt.
    saveToolPic←toolpic bitsIntoString.
    toolpic show.
    tools moveto: pt; setvalue: tool.  "tools"
    tool showon: actionbuttons]
```

Class init and Access

```
actionpic: a [actionpic ← a]
classInit | t i
    [t ← Vector new: 6.
    for§ i to: t length do§ [t◦i ← BitRectTool init].
    tools ← (RadioButtons new) vec: t at: 0⊙0 width: 20.
    actionpic←BitRect new filin: 'actionpic'.
    toolpic←BitRect new filin: 'toolpic'.
    editmenu←Menu new string: 'grow'.
    self initActionMenu]
initActionMenu | s z
    [s ← Vector new: 5. z ← 20.
    so1 ← (RadioButtons new) vec: ↱(setbrush paint block draw line blowup) at:
0⊙0 height: z. "action"
    so2 ← (RadioButtons new) vec: (black, dkgray, gray, ltgray, white) at: 0⊙0
height: z. "tone"
    so3 ← (RadioButtons new) vec: (0, 1, 2, 3) at: 0⊙0 height: z. "mode"
    so4 ← (RadioButtons new) vec: (1, 2, 4, 8) at: 0⊙0 height: z. "width"
    so5 ← (RadioButtons new) vec: (1, 2, 4, 8, 16, 32) at: 0⊙0 height: z. "grid"
    actionbuttons ← s.]
toolpic: a [toolpic ← a]
↵
SystemOrganization classify: ↱BitRectSelection under: 'Picture Editor'.↵
BitRectSelection classInit↵
```

"BitRectTool"

```
Class new title: 'BitRectTool'
   subclassof: Object
   fields: 'action "<UniqueString> the current action"
      pencil  "<Turtle> used for draw or straight-edge"
      brush  "<BitRect> source for painting"
      mode  "<Integer> how brush combines with the destination"
      tone  "<Integer> a spatial half-tone color (4 bits by 4 bits)"
      grid  "<Integer> all mouse points are rounded to this"'
   declare: 'blowupScale graypens brushpt ';
   asFollows_!
```

A BitRectTool paints on the screen.
A tool is a combination of action, mode, pen-width, gray, and grid.
action is one of: make-brush, paint, block-of-gray, draw, straight-edge, magnify.
mode is one of: store, or, xor, and. (how tool is combined with picture)
pen-width is 1, 2, 4, or 8. (width of the pen)
gray is one of: black, darkgray, gray, lightgray, white.
grid is one of: 1, 2, 4, 8, 16, 32. (minimum spacing of mouse points)
Menus for each part of a tool appear above the picture (in the same order).
Some actions do not use certain of the other parts of a tool.
(example: Block-of-gray does not use pen-width.)

brushpt = Point in the menu where brush is shown.
graypens = Vector of Strings of bits in pens.

Tool action

```
block [self getRectangle color: tone mode: mode]
blowup | smallRect bigRectFrame
   [smallRect←self getRectangle.
   bigRectFrame ← Rectangle new origin: smallRect corner
      extent: 4⊙4 + (smallRect extent*blowupScale).
   smallRect empty or: bigRectFrame bitStringLength>2000⇒
      [pencil frame flash. ∩nil].
   [user screenrect has: bigRectFrame corner⇒[]
      bigRectFrame moveto: smallRect origin-bigRectFrame extent.
      user screenrect has: bigRectFrame origin⇒[]
      "can't find a space for blown up image"
      pencil frame flash. ∩nil].
   self blowup: smallRect to: bigRectFrame]
blowup: smallRect to: bigRectFrame
      | bigRect box pt i turt flag bits
   [bits ← bigRectFrame bitsIntoString.
   bigRect ← bigRectFrame inset: 2⊙2.
   smallRect blowup: bigRect origin by: blowupScale.
   turt←Turtle init.
   box ← 0⊙0 rect: (blowupScale-1)⊙(blowupScale-1).
   "keep editing in blowup mode until the user presses a button
      outside the big rect"
```

```
    while: flag do:
      [bigRect has: (pt ← user mp)⇒
        [box moveto: bigRect origin + (i ← pt-bigRect origin|blowupScale).
        turt place: smallRect origin + (i/blowupScale).
        user redbug⇒[box color: black mode: storing.
          turt black; go: 0]
        user yellowbug⇒[box color: white mode: storing.
          turt white; go: 0]
        user bluebug⇒[bigRect flash]]
      user anybug ⇒
        [(bigRect inset: ¯5⊙¯5) has: pt⇒[bigRect flash]
        "quit" flag←false]].
    bigRectFrame bitsFromString: bits]
brush: sourceRect  "use the bits in the BitRect sourceRect as a brush"
      | minpt maxpt pt offset
    ["The inner painting loop should be fast - all the extra foliage below
      is to move tests outside of the inner loop"
    sourceRect moveto: brushpt; show.
    minpt←self frame origin.
    maxpt←self frame corner-sourceRect extent.
    offset←sourceRect extent/2.
    "If mode is storing or oring, use brush command, otherwise blt.
      Use the unclipped form of brushing  and grid=1 when possible"
    [mode<xoring and: grid=1⇒
      [while: user redbug do:
        [minpt≤(pt←user mp-offset) and: pt≤maxpt⇒
          [sourceRect brush: pt mode: mode color: tone]
        sourceRect brush: pt mode: mode color: tone clippedBy: self frame]]
     mode≥xoring and: grid=1⇒
      [while: user redbug do:
        [minpt≤(pt←user mp-offset) and: pt≤maxpt⇒
          [sourceRect blt: pt mode: mode]
        sourceRect blt: pt mode: mode clippedBy: self frame]]
     mode<xoring⇒  "grid is > 1"
      [while: user redbug do:
        [minpt≤(pt←self mpOnGrid-offset) and: pt≤maxpt⇒
          [sourceRect brush: pt mode: mode color: tone]
        sourceRect brush: pt mode: mode color: tone clippedBy: self frame]]
     "grid is > 1 and mode≥xoring"
      while: user redbug do:
        [minpt≤(pt←self mpOnGrid-offset) and: pt≤maxpt⇒
          [sourceRect blt: pt mode: mode]
        sourceRect blt: pt mode: mode clippedBy: self frame]].
    sourceRect clear: white]
draw
    [tone=white or: tone=black⇒
      [pencil place: self mpOnGrid-pencil frame origin.
      grid=1⇒        "make drawing with grid 1 fast"
        [while: user redbug do:
          [pencil goto: user mp-pencil frame origin]]
      while: user redbug do:
        [pencil goto: self mpOnGrid-pencil frame origin]]
```

```
    self brush: graypens∘pencil width]
getRectangle | rect newrect start t    "rect must be in my frame"
    ["the rect-newrect stuff is so that the complementing stays
      on for a while"
    start←self mpOnGrid.
    rect←newrect←(Rectangle new origin: start corner: start)
      intersect: self frame.
    "move the cursor slightly so that the user will notice the rectangle
      being complemented"
    user cursorloc←start+4.
    while: user anybug do:
      [rect←newrect.
      rect comp.
      t←self mpOnGrid.
      newrect←(Rectangle new origin: (start min: t) corner: (start max: t))
        intersect: self frame.
      rect comp].
    ↑rect]
line | start end width
    [start←self mpOnGrid-pencil frame origin.
    width←pencil width.  pencil xor; width: 1.
    while: user redbug do:
      [end←self mpOnGrid-pencil frame origin.
      pencil xor; place: start; goto: end; place: start; goto: end].
    [tone=white⇒[pencil white] pencil black].
    pencil width: width; place: start; goto: end]
mpOnGrid    "return mouse point rounded to grid"
    [↑user mp+(grid/2) | grid]
paint
    [self brush: brush]
redbug [self perform: action]
setbrush | rect
    [rect←self getRectangle.
    rect empty or: 50⊙50<rect extent⇒[pencil frame flash]
    brush title: 'brush' in: rect; saveScreenBits.
    brush moveto: 0⊙0.
    action ← ↗paint]
```

Tool selection

```
brushpt: pt    "set the point at which the current brush will be shown"
    [brushpt←pt]
frame [↑pencil frame]
frame: f [pencil frame: f]
setfrom: butvec | pt
    [butvec∘1 has: (pt ← user mp) ⇒
      [action ← butvec∘1 bug: pt]
    butvec∘2 has: pt ⇒[tone ← butvec∘2 bug: pt.
      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)]
    butvec∘5 has: pt ⇒[grid ← butvec∘5 bug: pt]
    ]
```

```
showOn: butVec
   [butVec@1 setValue: action.
   butVec@2 setValue: tone.
   butVec@3 setValue: mode.
   butVec@4 setValue: pencil width.
   butVec@5 setValue: grid]
```

Class initialization

```
classInit | rect saveBits t i
   [blowupScale←5.
   "make a vector of gray pens"
   rect ← 0@0 rect: 9@9.
   saveBits←rect bitsIntoString.
   t ← Turtle init.
   graypens ← Vector new: 8.
   for: i to: 8 do:
      [t width: i.
      rect clear: white.
      t place: 4@4; go: 0.
      graypens@i ← BitRect new title: 'graypen' in: rect.
      (graypens@i) saveScreenBits].
   rect bitsFromString: saveBits]
init
   [(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]
↵
SystemOrganization classify: ↗BitRectTool under: 'Picture Editor'.↵
BitRectTool classInit↵
```

"RadioButtons"

```
Class new title: 'RadioButtons'
   subclassof: Object
   fields: 'vec  "<Vector> values corresponding to the buttons"
      cur  "<Integer> button currently selected"
      rect  "<Rectangle> contains all the buttons"
      size  "<Integer> width or height of a button"'
   declare: '';
   asFollows↵
```

A RadioButtons is a row or column of square regions ("buttons") on the display screen. There is always exactly one button pushed. (RadioButtons is a model of the station selection buttons on a car radio.) The pushed button has a black box around it. Each button has a value associated with it, which is returned when the button is pressed. RadioButtons will not destroy a menu picture (BitRect) displayed in its area, but the RadioButtons has no knowledge of the picture.

Pushing a Button

```
bug: pt | r a
   [r ← (pt - rect origin - (1⊙1)) / size.
   a ← r x + r y + 1.
   ↑self push: a]
push: a
   [self release: cur thenPush: a.
   ↑vec⊙(cur ← a)]
setvalue: v | i
   ["if value has been lost, set self to 1"
   i←(vec find: v) max: 1.
   self push: i. ↑i]
```

Init and State

```
has: pt [↑rect has: pt]
moveto: pt
   [rect moveto: pt.
   cur ← 0.
   ↑rect corner x ⊙ rect origin y]
reset [cur←0]
value [↑vec⊙cur]
vec [↑vec]
vec: vec at: r height: size
   [rect ← r rect: r+ ((vec length ⊙ 1)*size).
   cur ← 0]
vec: vec at: r width: size
   [rect ← r rect: r+ ((1 ⊙ vec length)*size).
   cur ← 0]
```

Private

```
release: a thenPush: b | boxer offset
   [a=b⊃[]
```

```
    offset ← [size=rect extent y⇒[size⊙0] 0⊙size].
    [a≠0⇒[boxer ← Rectangle new origin: (offset*(a-1)+rect origin+1)
            extent: size⊙size-1.  boxer comp.  (boxer inset: 2⊙2) comp]].
    b≠0⇒[boxer ← Rectangle new origin: (offset*(b-1)+rect origin+1)
            extent: size⊙size-1.  boxer comp.  (boxer inset: 2⊙2) comp]]
↵
SystemOrganization classify: ↗RadioButtons under: 'Picture Editor'.↵
```

"Document"

```
Class new title: 'Document'
   subclassof: Object
   fields: 'entities windows'
   declare: '';
   asFollows_!
```

A collection of entities shown through a collection of windows. The windows are currently stored in a vector. Subclasses may override many of my messages.

Initialization

```
init
   [windows ← ↗()]
```

Aspects

```
o index
   [↑entities∘index]
entities
   [↑entities]
entityIndexAt: pt
   [user subclassJob]
entityIndexOf: entity [↑(entities find: entity) > 0]
frame
   [user subclassJob]
frameOf: index
   [user subclassJob]
height
   [↑self frame height]
```

Viewing

```
isntSeenIn: w
   [windows ← windows delete: w]
isSeenIn: w
   [windows ← windows , w]
```

Showing

```
showEdited: index in: proj | dY domain [
   domain ← self frameOf: index.
   dY ← entities∘index growthThru: proj within: domain.
   dY=0⇒ [proj imprint]
   [dY < 0⇒ [proj paint: white in: domain with: storing]].
   self imprint: index grown: dY in: proj]
showThru: proj
   [user subclassJob]
typeset: index in: proj
   [entities∘index typesetIn: proj within: (self frameOf: index)]
```

Editing

```
replace: index by: ents
    [entities ← entities replace: index to: index by: ents]
```

As entity

```
deselected
    ["I don't care"]
growthThru: proj within: domain
    [↑self height - domain height]
selected
    ["I don't care"]
selectionSpecies
    [↑Selection]
```

Changing

```
entityIndex: index oldEntity: entity
⌟
SystemOrganization classify: ↗Document under: 'Documents'.⌟
```

"Galley"

```
Class new title: 'Galley'
   subclassof: Document
   fields: 'width bottoms top press edited'
   declare: '';
   asFollows↵
```

A single column width wide of contiguous entities. The top of the first entity is at y=top; the bottom of the i-th entity is at y=bottom∘i.

Initialization

```
with: entities width: width [
   self with: entities width: width bottoms: bottoms top: 0]
with: entities width: width bottoms: bottoms top: top [
   super init.
   edited ← false.
   [top≡nil⇒ [top ← 0]].
   [width≡nil⇒ [width ← 515 "~6.5 inches"]].
   [bottoms≡nil⇒ [self computebottoms]].
   press≡nil⇒ [press ← false]]
```

Aspects

```
bottoms
   [↑bottoms]
computebottoms | proj y frame i [
   [bottoms≡nil or: bottoms length≠entities length⇒ [
      bottoms ← Vector new: entities length]].
   bottoms∘1 ← y ← top.
   frame ← self frameOf: 1.
   proj ← Projector new projecting: y asPoint of: nil onto: 0 asRectangle.
   for: i to: entities length do: [
      bottoms∘i ← y ← y +
         (entities∘i growthThru: proj within: frame)].
   ↑bottoms]
frame
   [↑self minX⊙self minY rect: self maxX⊙self maxY]
frameOf: index
   [↑ Rectangle new
      origin: (Point new x: 0
         y: [index=1⇒ [top] bottoms∘(index-1)])
      corner: (Point new x: width y: bottoms∘index)]
framesShifted: dXY | t b l dY y
   [l ← dXY x. dY ← dXY y. b ← top+dY.
   ↑bottoms transform: y to: [t ← b. b ← y+dY. (l⊙t rect: l+width⊙b)]]
height [↑self maxY-self minY]
maxX
   [↑width]
maxY
   [↑[bottoms length=0⇒ [top] bottoms last]]
minX
```

```
  [↑0]
minY
  [↑top]
```

Showing

```
imprint: index grown: dY in: proj  | i y
  [y ← bottoms∘index.
  self bubble: index by: dY.
  proj receive: (0@(y+dY) rect: 32000 asPoint) from: proj at: 0@y.
  proj imprint]
show [
  user restartup: (
    GalleyWindow new on: self named: [press⇒ [press name] 'unnamed'])]
showThru: proj | i fr [
  fr ← proj aperture.
  i ← bottoms findSorted: fr minY.
  ↑self showThru: proj entityIndices: (i to: (
    (bottoms findSorted: fr maxY) min: bottoms length))]
showThru: proj entityIndices: index [
  for: index from: index do: [
    entities∘index showThru: proj within: (self frameOf: index)]]
showThru: proj within: frame
```

Selection

```
entityIndexAt: pt
  [pt y between: self minY and: self maxY⇒ [↑bottoms findSorted: pt y]
   ↑false]
```

Editing

```
bubble: index by: dY  | i
  [dY=0⇒ []
  for: i from: index to: bottoms length do:
    [bottoms∘i ← bottoms∘i + dY]]
insert: ent height: ht after: index  | t  "index=0 for beginning"
  [t ← [index=0⇒ [top] bottoms∘index].
  self bubble: index+1 by: ht.
  entities ← entities replace: index+1 to: index by: ent inVector.
  bottoms ← bottoms replace: index+1 to: index by: (t+ht) inVector]
replace: index by: pairs  | f  "pairs form is ((ent ... ent) (frame ... frame))"
  [super replace: index by: pairs∘1.
   bottoms ← bottoms replace: index to: index by:
    (pairs∘2 transform: f to: f maxY)]
```

Press

```
fromPress: press name: c value: v [
  c=0⇒ [↙
    v≡nil⇒ [
      "init, read, end"
      user displayoffwhile: [
      (entities ← Set new vector: 50) append: press.
      self with: entities width: width bottoms: bottoms top: top]]
```

```
      user notify: 'illegal user of code 0'];

   "read special stuff from press part"
   =¯6⇒ [
      width ← υ nextword.
      bottoms ← Vector new: υ nextword "number of entities".
      for: c to: bottoms length do: [bottoms∘c ← υ nextword]]
   user notify: 'illegal code']
hardcopy [
   self save.
   press toPrinter]
save [
   edited⇒ [
      self toPress.
      edited ← false]]
toPress | p i w [
   user displayoffwhile: [
   [press⇒ [press reset]
   press ← dp0 pressfile: (user request: 'type a press file name')].

   w ← width * press scale.
   i ← 2540 "inch in micas".
   p ← PressPrinter init.
   p press: press;
      frame ← (Rectangle new "in micas"
         "try equal left/right margins for now"
         origin: (8.5*i) asInteger - w /2 ⊙ (1*i)
         extent: w ⊙ (9*i)).

   "for now, print everything"
   for: i from: entities do: [p print: i].

   "write additional control info"
   p ← press data.
   press
      part: [
         p nextword ← width;
            nextword ← bottoms length.
         for: i from: bottoms do: [p nextword ← i]] code: ¯6;
      close]

   "an alternative way of doing things is to define a projector-like
   printer.  see printer.st and printerchanges.st for some likely code"
   "p ← Printer new projecting: self onto: press.
   self showThru: p entityIndices: (1 to: entities length)."]
```

Bravo

```
readBravo: strm width: width | e s   "width is in points (72 per inch)"
   [e ← (Vector new: 100) asStream.
   s ← Dictionary init.  s insert: 0 with: Style default.          "for sharing style"
   until: strm end do:
      [e next ← TextEntity new readBravo: strm styles: s]
```

```
    self with: e contents width: width]
writeBravo: f | e
    [for: e from: entities do:
        [e writeBravo: f]]
!
SystemOrganization classify: #Galley under: 'Documents'.!
```

"Page"

```
Class new title: 'Page'
   subclassof: Document
   fields: 'pattern frames '
   declare: '';
   asFollows↵
```

A bounded rectangular area defined by pattern (which will be a pattern page eventually, but now is just a rectangle) on which entityoi is positioned within framesoi.

Initialization

```
pattern: pattern
   [entities ← frames ← windows ← ↱()]
with: entities in: frames
```

Showing

```
frame
   [↑pattern "temporarily"]
frameOf: index
   [↑framesoindex]
height
   [↑pattern height]
imprint: index grown: dY in: proj  "for now, allow growth"
   [framesoindex growby: dY. proj imprint]
showThru: proj
   [self showThru: proj translated: 0]
showThru: proj translated: dXY  | index frame aperture
   [aperture ← proj aperture.
   for: index to: entities length do:
      [frame ← framesoindex + dXY.
      frame intersects: aperture⇒
         [entitiesoindex showThru: proj within: frame]]]
showThru: proj within: domain
   [self showThru: proj translated: domain origin-pattern origin]
```

Editing

```
entityIndexAt: pt  | i
   [for: i to: entities length do:
      [framesoi has: pt⇒ [↑i]].
    ↑false]
occupies: rect | frame
   [for: frame from: frames do:
      [frame intersects: rect⇒ [↑true]].
    ↑false]
replace: index by: pairs "pairs form is ((ent ... ent) (frame ... frame))"
   [super replace: index by: pairso1.
    frames ← frames replace: index to: index by: pairso2]
↵
SystemOrganization classify: ↱Page under: 'Documents'.↵
```

"ParagraphEditor"

```
Class new title: 'ParagraphEditor'
   subclassof: TextSelection
   fields: ''
   declare: '';
   asFollows_!
```

This version of ParagraphEditor (for use in CodePanes) is based on TextSelection

Initialization

```
para: entity frame: frame [self frame ← frame]
```

Scheduling

```
enter [
   typein ← false.
   self show; select]
leave [self complement: off]
```

Public Messages

```
fixframe: f | dy window [
   f≡nil⇒ []
   dy ← [proj≡nil⇒ [0] self frameoffset].
   ([proj≡nil⇒ [proj ← Projector new] proj])
      projecting: 0⊙0 of: nil onto: (window ← f copy);
      typeset: entity in: (
         Rectangle new origin: 2⊙dy extent: f width-2 ⊙ 9999)
      as: DefaultTextStyle.
   ↑window]
formerly [↑oldEntity]
formerly: oldEntity
frame← f [self fixframe: f]
keyset
Scrap ← s [Scrap ← s]
scrollby: n | p [
   n ← (n * self lineheight) max: self frameoffset.
   p ← 0⊙(0-n).
   proj translate: p;
      screenFrame moveby: p.
   self show; select]
scrollPos | t [
   t ← self height - self lineheight.
   t=0⇒ [↑0.0]
   ↑0.0 - self frameoffset / t]
scrollTo: f [self scrollUp: self frameoffset + (f* self height) - 4]
scrollUp: n [self scrollby: n/self lineheight]
select: t [
   self complement: off.
   c1 ← c2 ← t.
   self selectAndScroll]
selectAndScroll | l dy c1y [
```

```
    l ← self lineheight.
    self select.
    c1y ← (proj screenHairBeforeChar: c1) minY.
    dy ← c1y - proj screenAperture minY.
    [dy ≥ 0⇒ [
        dy ← c1y + l - 1 - proj screenAperture maxY max: 0]].
    dy≠ 0⇒ [self scrollby: (dy abs+l-1) / l * dy sign]]
selecting | t pt [
    t ← proj charNearScreenPt: (pt ← user mp).
    self complement: off; fintype.
    t = c1 and: (c1=c2 and: c1≠1)⇒ "double bug"
        [↑self selectword select]
    ↑self selecting: pt]
selectionAsStream [↑Stream new of: entity text from: c1 to: c2-1]
selectword          "Select bracketed or word range for double-click"
    | a b dir t level open close s
    [a← b← dir← ¯1.
    open ← '([{< ''"
'.
    close ← ')]}> ''"
'.
        [c1> entity length⇒[t←c1-1]
        c1≤1⇒[dir←1. t←c1]
        t←open find: (a← entity∘(c1-1)). t>0⇒          "delim on left"
            [dir←1. b←close∘t. t←c1-1]              "match to the right"
        t←close find: (a← entity∘c1). t>0⇒             "delim on right"
            [dir←¯1. b←open∘t. t←c1]               "match to the left"
        a← ¯1. t←c1].                    "no delims - select a token"
    level←1. s ← entity text.
    until: (level=0 or: [dir=1⇒[t≥s length] t≤1]) 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⇒[c1← dir← 1. t← c2]]
            dir=¯1⇒[c1 ← t+1. dir←1. t←c2]      "then right"
            level← 0]]
    [level≠0⇒[t← t+dir]].
    dir=1⇒[c2← t] c1← t+1]
show [
    proj imprint.
    sel ← off]
typing [self kbd]
unselect [self complement: off]
```

Private

```
frameoffset [
    "a useful number"
    ↑proj screenFrame minY - proj screenAperture minY]
height [
    ↑(proj screenHairBeforeChar: entity length+1) maxY -
        proj screenFrame minY]
lineheight [↑proj textStyle lineheight]
```

```
select [self selectIn: nil]
selectIn: w [
    [c1 ≡nil⇒ [c1 ← c2 ← 1]].
    self complement: on]
↵
SystemOrganization classify: #ParagraphEditor under: 'Documents'.↵
```

"Projector"

```
Class new title: 'Projector'
   subclassof: Object
   fields: 'screenFrame para textStyle selOrigin selCorner
screenAperture screen aperture slide symbolism reduction
translation'
   declare: '';
   asFollows
```

I project the Rectangle 'aperture' of a slide 'slide' (see below) onto the Rectangle 'screenAperture' of a Screen 'screen', symbolizing each entity of the slide according to a symbolism 'symbolism' understood only by the slide.

A slide is any object (usually a document, or part of a document) that responds suitably to the message:
showThru: proj
where proj is a Projector such as me. In response to that message, the slide's displayable entities should pass me messages to tell me to display them. To display a Paragraph 'para' aligned in a Rectangle 'frame' according to a TextStyle 'textStyle', I first must be prepared by telling me:
proj typeset: para in: frame as: textStyle
after which I can be told to display the last typeset paragraph by:
proj imprint
or to find out where the char'th character would be displayed by:
proj hairBeforeChar: char
or to find out by how much frame must be grown (or shrunk) in height in order to display all of para by:
proj frameGrowth
or to do other actions of that kind. Caution: the messages 'show' and 'reshow' should not intervene, because they may cause other paragraphs to be typeset.

All coordinates in this class are slide coordinates except where the word 'screen' appears in the name of the variable or message, or where the name is 'eraseRect(s)' or 'keepRect'.

For efficiency, I remember certain frequently needed values, as follows.

The Point or Integer 'reduction' records the ratio of the size of aperture to the size of screenAperture. The x and y components of reduction may differ, but both must be integral. The most common reductions are 1 and 2.

The Point 'translation' records the distance between screenAperture and the reduced aperture.

Several operations require an ink argument; the alternatives are storing, oring, xoring, and erasing. Some require a tone argument, such as white, black, gray, ltgray, dkgray, or background. For a more complete description of the encoding, see class Rectangle.

Operations are provided to aid text selection. Some deal with the ends of a highlighted selection, rectangles of zero width, which are here called 'hairlines'.

Class Screen does not yet exist, so my paint operations are currently being performed by class Rectangle.

I provide a large number of services, some of which may prove superfluous as experience is gained. At some future time, such services may be eliminated.

Initialization

projecting: apertureOrigin of: slide onto: screenAperture
["*Make me project into screenAperture that portion of slide originating at apertureOrigin. Default my screen, reduction, and symbolism.*"
self projecting: apertureOrigin of: slide onto: screenAperture of: nil reduced: 1 symbolism: nil]
projecting: apertureOrigin of: slide onto: screenAperture of: screen reduced: reduction symbolism: symbolism
["*Make me project into screenAperture of screen that portion of slide originating at apertureOrigin, with the specified reduction and symbolism.*"
aperture ← apertureOrigin rect: (screenAperture extent*reduction + apertureOrigin).
self computeTranslation.
screenFrame ← 0 asRectangle.
para ← 'You forgot to typeset.' asParagraph.
textStyle ← DefaultTextStyle]

Text Primitives

charNearPt: pt
["*In the most recently typeset Paragraph, find the character whose left edge is nearest to pt and return its character index. Exceptions: if pt is past the end of a line, return the index of the first character on the next line; if past the end of the Paragraph, return one plus the length of the Paragraph.*"
↑self charNearScreenPt: (self screenPtOf: pt)]
charNearScreenPt: screenPt
["*In the most recently typeset Paragraph, find the character whose left edge is nearest to screenPt and return its character index. Exceptions: if screenPt is past the end of a line, return the index of the first character on the next line; if past the end of the Paragraph, return one plus the length of the Paragraph.*"
user croak] primitive: 58
complementChars: char1 to: char2 | screenHair1 screenHair2
["*Complement the slide dots corresponding to the the lines and part-lines of the most recently typeset paragraph between the left edge of char1 and the left edge of char2. If char1 = char2, this is a no-op. If char1 > char2, this is undefined.*"
screenHair1 ← self screenHairBeforeChar: char1.
screenHair2 ← [char2=char1⇒[screenHair1 +(1 ⊙0)] self screenHairBeforeChar: char2].
self complementScreenHairs: screenHair1 to: screenHair2]
complementHairs: hair1 to: hair2 | screenHair1 screenHair2
["*Complement the screen dots corresponding to the lines and part-lines of typeset text between hair1 inclusive and hair 2 exclusive. If hair1 = hair2, this is a no-op. If hair1 > hair2, this is undefined.*"
screenHair1 ← self screenRectOf: hair1.

```
    screenHair2 ← self screenRectOf: hair2.
    self complementScreenHairs: screenHair1 to: screenHair2]
complementScreenHairs: hair1 to: hair2 | screenRect
["Complement the screen dots corresponding to the lines and part-lines of the
most recently typeset paragraph between hair1 inclusive and hair2 exclusive.  If
hair1 = hair2, this is a no-op.  If hair1 > hair2, they are reversed.  This
complementing happens in three parts, A, B, and C, between points 1 and 2,
according to the following illustration:
                1AAA
                BBBB
                BBBB
                BBBB
                CCC2
unless there is just one line involved, as in:
                1DD2
"
    "one line case"
    hair1 minY = hair2 minY⇒
        [self complementScreenRect: [
        hair1 minX ≤ hair2 minX⇒ [hair1 origin rect: hair2 corner]
        hair2 origin rect: hair1 corner]]

    [hair1 minY > hair2 minY⇒ [
        screenRect ← hair1. hair1 ← hair2. hair2 ← screenRect]].

    screenRect ← (screenFrame minX ⊙ hair1 maxY) rect: (screenFrame maxX ⊙
hair2 minY).
    self complementScreenRect: (hair1 origin rect: (screenRect maxX ⊙
screenRect minY));
        complementScreenRect: screenRect;
        complementScreenRect: ((screenRect minX ⊙ screenRect maxY) rect: hair2
corner)]
frameGrowth | screenDY
["Adjust the height of screenFrame to be exactly enough to display all of the
most recently typeset paragraph.  Return the difference (in slide coordinates)
between the new height and the old height.  If the answer is 0, the frame was
just the right height; if positive, the frame was too short; if negative, the
frame was too tall."
    screenFrame growby: 0 ⊙ 820.
    screenDY ← (self screenHairBeforeChar: 1 + para length) corner y -
screenFrame corner y.
    screenFrame growby: 0⊙screenDY.
    ↑screenDY+820 * reduction asPtY]
hairBeforeChar: char
["Return a zero-width Rectangle along the left edge of the char character of the
most recently typeset Paragraph.  Exception: if char is one greater than the
length of the Paragraph, return a zero-width Rectangle along the right edge of
the last character."
    ↑self rectOf: (self screenHairBeforeChar: char)]
hairBeforeThatChar
["Return a zero-width Rectangle along the left edge of that character of the
most recently typeset Paragraph last involved in a -HairBeforeChar or a
```

```
charNear-Pt operation."
    ↑(self ptOf: selOrigin) rect: (self ptOf: selCorner)]
imprint
["Display the most recently typeset Paragraph, para, according to textStyle,
aligned to the frame, screenFrame, and clipped by screenAperture.  Stop
displaying if the bottom of screenFrame is encountered."
    user croak] primitive: 57
screenHairBeforeChar: char
["Return a zero-width Rectangle along the left edge of the char character of the
most recently typeset Paragraph.  Exception: if char is one greater than the
length of the Paragraph, return a zero-width Rectangle along the right edge of
the last character."
    self selectChar: char.
    ↑Rectangle new origin: selOrigin corner: selCorner]
screenHairBeforeThatChar
["Return a zero-width Rectangle along the left edge of that character of the
most recently typeset Paragraph last involved in a -HairBeforeChar or a
charNear-Pt operation."
    ↑Rectangle new origin: selOrigin corner: selCorner]
typeset: para in: rect as: textStyle
["Typeset the Paragraph para so it can be shown aligned in frame using the
specified textStyle, which had better take reduction into account."
    screenFrame ← self screenRectOf: rect]
```

Graphics

```
clear
["Paint my screenAperture white."
    screenAperture color: white mode: storing]
complementScreenRect: screenRect
["Complement those screen dots in screenRect that appear in screenAperture."
    (screenAperture intersect: screenRect) comp]
flash
["Flash my screenAperture momentarily."
    screenAperture flash]
paint: tone in: rect with: ink
["Fill the visible part of rect with tone, using the logical operation, ink."
    (self visibleScreenRectOf: rect) color: tone mode: ink]
saveBits: str in: rect clippedBy: clipRect
["save the bits in Rectangle rect clipped by Rectangle cliprect,
by bitting them into the string str.
str must be the right length for holding all the bits in rect."
(self screenRectOf: rect) bitsIntoString: str mode: storing
    clippedBy: (self visibleScreenRectOf: (clipRect intersect: rect))]
showBitRect: b in: rects clippedBy: clipRect | i [
    "show a BitRect (Vector of Strings and Rectangles)"
    b ← b data.
    for: i to: b length do: [
        self showBits: b[i] in: rects[i] clippedBy: clipRect]]
showBits: str in: rect clippedBy: clipRect
["show the bitstring str in Rectangle rect clipped by Rectangle cliprect.
str must be the right length for holding all the bits in rect."
(self screenRectOf: rect) bitsFromString: str mode: storing
```

```
    clippedBy: (self visibleScreenRectOf: (clipRect intersect: rect))]
```

Showing

receive: rect from: proj at: pt | screenRect screenDXY destProj keepRect eraseRects

["*Reshow that part of the material in rect that also falls in aperture. Take advantage of the fact that proj (which may or may not be me, but which has the same symbolism and reduction as me) has recently displayed (its part of) the same material at its pt. Move as much of that image as possible using Blt. Form a vector of disjoint screen Rectangles which, if painted white, would erase that old image without affecting the new image. Erase them and return the vector for the caller's possible use.*"

```
    screenRect ← self screenRectOf: rect.
    screenDXY ← screenRect origin - (proj screenPtOf: pt).
    destProj ← self subProjectorFrom: rect of: slide.
    keepRect ← proj screenAperture intersect: destProj screenAperture -
screenDXY.
    eraseRects ← (proj screenAperture intersect: screenRect - screenDXY)
minus: destProj screenAperture.
    destProj reshowBlting: keepRect by: screenDXY erasing: eraseRects.
    ↑eraseRects]
```

reshow

["*Display all of aperture in screenAperture, clearing it first.*"

```
    self clear; show]
```

show

["*Display all of aperture in screenAperture, without clearing it first.*"

```
    slide showThru: self]
```

Shifting

aperture ← rect

["*Change my aperture to rect, thereby changing its position and/or size. Adjust my state accordingly. Do not affect the display.*"

```
    screenFrame moveby: (self screenDXYOf: aperture corner - rect corner).
    aperture ← rect.
    self computeTranslation.
    screenAperture growto: (self screenPtOf: aperture corner)]
```

moveInto: screenRect | eraseRects screenDXY keepRect

["*Change my screenAperture to screenRect, thereby changing its position and/or size. Adjust my state accordingly. show my new image, trying to take advantage of the fact that my old image corresponded with that in aperture. Form a vector of disjoint screen Rectangles which, if painted white, would erase my old image without affecting my new image. Erase them and return the vector so the caller will be able to regenerate other images that have been uncovered. This is faster than shiftToProject.*"

```
    eraseRects ← screenAperture minus: screenRect.
    screenDXY ← screenRect origin - screenAperture origin.
    keepRect ← screenAperture intersect: screenRect - screenDXY.
    self screenAperture ← screenRect.
    self reshowBlting: keepRect by: screenDXY erasing: eraseRects.
    ↑eraseRects]
```

screenAperture ← screenRect

```
["Change my screenAperture to screenRect, thereby changing its position
and/or size.  Adjust my state accordingly.  Do not affect the display."
   screenFrame moveby: screenRect origin - screenAperture origin.
   screenAperture ← screenRect.
   self computeTranslation.
   aperture growto: (self ptOf: screenRect corner)]
scrollBy: dXY
["Move aperture by dXY, and reshow.  Try to take advantage of the fact that
the image in screenAperture corresponds to that in the old aperture."
   self scrollTo: aperture origin + dXY]
scrollPos | h
   [h ← slide height.
   h=0⇒ [⋂0.0]
   ⋂aperture minY asFloat/h]
scrollTo: apertureOrigin | screenDXY keepRect eraseRects
["Move aperture's upper left corner to apertureOrigin, and reshow.  Try to take
advantage of the fact that the image in screenAperture corresponds to that in
the old aperture.  This is faster than shiftToProject."
   apertureOrigin ← (apertureOrigin max: 0⊙0) min: 0⊙(slide height-30).
   screenDXY ← screenAperture origin - (self screenPtOf: apertureOrigin).
   aperture moveto: apertureOrigin.
   self computeTranslation.
   screenFrame moveby: screenDXY.
   keepRect ← screenAperture intersect: (screenAperture - screenDXY).
   eraseRects ← screenAperture minus: (keepRect + screenDXY).
   self reshowBlting: keepRect by: screenDXY erasing: eraseRects]
shiftToProject: apertureOrigin onto: screenRect showingIf: doShow | rect
screenDXY proj
["Move aperture's upper left corner to apertureOrigin, and change my
screenAperture to screenRect, thereby changing its position and/or size.
Adjust my state accordingly.  If doShow, also: (1) reshow me, trying to take
advantage of the fact that the image in the old screenAperture corresponds to
that in the old aperture; (2) clear a set of Rectangles that will erase my old
image without affecting my new image; and (3) return those cleared
Rectangles in a Vector.  This message is very general, but if only the slide or
screen is involved in the shift, scrollTo or moveInto operates faster."
   [doShow⇒ [proj ← self subProjectorFrom: aperture of: slide]].
   rect ← aperture.
   aperture ← apertureOrigin rect: (screenRect extent * reduction).
   rect ← aperture union: rect.
   screenDXY ← screenRect origin - screenAperture origin.
   screenAperture ← screenRect.
   self computeTranslation.
   screenFrame moveby: screenDXY.
   [doShow⇒ [⋂self receive: rect from: proj at: rect origin]]]
translate: dXY
["Adjust my internal state to account for the fact that slide has translated its
coordinate system so that what used to be at (0,0) is now at dXY.  Do not do
anything to the display."
   aperture moveby: dXY.
   self computeTranslation]
```

Projection

screenDXOf: dX
["*Return the screen dX proportional to dX.*"
↑dX/reduction asPtX]
screenDXYOf: dXY
["*Return the screen dXY proportional to dXY.*"
↑dXY/reduction]
screenDYOf: dY
["*Return the screen dY proportional to dY.*"
↑dY/reduction asPtY]
screenPtOf: pt [
"Return the screen Point onto which I project pt. If pt is outside aperture, then the Point I return will be outside screenAperture."
↑Point new x: (self screenXOf: pt x) y: (self screenYOf: pt y)]
screenRectOf: rect
["*Return the screen Rectangle onto which I project rect. If rect protrudes from aperture, then the Rectangle I return will protrude from screenAperture.*"
↑Rectangle new
origin: (self screenPtOf: rect origin)
corner: (self screenPtOf: rect corner)]
screenXOf: x
["*Return the screen X onto which I project x.*"
↑x/reduction asPtX + translation x]
screenYOf: y
["*Return the screen Y onto which I project y.*"
↑y/reduction asPtY + translation y]
subProjectorFrom: subAperture of: subSlide | rect
["*Return a new projector just like me but projecting (at most) subAperture of subSlide onto the corresponding screen rectangle.*"
rect ← aperture intersect: subAperture.
↑Projector new projecting: (rect origin) of: subSlide onto: (self screenRectOf: rect) of: screen reduced: reduction symbolism: symbolism]
visibleScreenRectOf: rect
["*Return the screen rectangle into which I project and clip rect.*"
↑screenAperture intersect: (self screenRectOf: rect)]

Reverse Projection

dXOf: screenDX
["*Return the slide dX proportional to screenDX.*"
↑screenDX * reduction asPtX]
dXYOf: screenDXY
["*Return the slide dXY proportional to screenDXY.*"
↑screenDXY * reduction]
dYOf: screenDY
["*Return the slide dY proportional to screenDY.*"
↑screenDY * reduction asPtY]
ptOf: screenPt [
"Return the slide Point which I project onto screenPt. If screenPt is outside screenAperture, then the Point I return will be outside aperture."
↑Point new x: (self xOf: screenPt x) y: (self yOf: screenPt y)]
rectOf: screenRect
["*Return the slide Rectangle which I project onto screenRect. If screenRect*

protrudes from screenAperture, then the Rectangle I return will protrude from aperture."
↑Rectangle new
origin: (self ptOf: screenRect origin)
corner: (self ptOf: screenRect corner)]
subProjectorTo: subScreenAperture of: subSlide | screenRect
["*Return a new projector just like me but projecting a corresponding slide rectangle of subSlide onto (at most) subScreenAperture.*"
screenRect ← screenAperture intersect: subScreenAperture.
↑Projector new projecting: (self ptOf: screenRect origin) of: subSlide
onto: screenRect of: screen reduced: reduction symbolism: symbolism]
visibleRectOf: screenRect
["*Return the rectangle which I project into screenRect clipped by screenAperture.*"
↑aperture intersect: (self rectOf: screenRect)]
xOf: screenX
["*Return the slide X which I project onto screenX.*"
↑screenX-translation x * reduction asPtX]
yOf: screenY
["*Return the slide Y which I project onto screenY.*"
↑screenY-translation y * reduction asPtY]

Access To Parts

aperture [↑aperture]
reduction [↑reduction]
screenAperture [↑screenAperture]
screenFrame [↑screenFrame]
slide [↑slide]
symbolism [↑symbolism]
textStyle [↑textStyle]

Utilities

printon: stream
["*Print a short description of me on stream.*"
stream append: 'A Projector from '; print: aperture; append: ' to '; print: screenAperture]

Private Operations

computeTranslation
["*Compute my translation.*"
translation ← screenAperture origin - (aperture origin / reduction)]
reshowBlting: screenRect by: screenDXY erasing: eraseRects | eraseRect
["*Use Blt to copy screenRect to a position screenDXY from where it is now; this will accomplish part of a show of me. Accomplish the rest by creating subProjectors and telling them to show. Paint all the Rectangles in the Vector 'eraseRects' white after the Blt and before the regeneration; this will accomplish an unshow of an old image.*"
screenRect blt: (screenRect origin + screenDXY) mode: storing.
for: eraseRect from: eraseRects do: [eraseRect color: white mode: storing].
eraseRects ← screenAperture minus: screenRect + screenDXY.
for: eraseRect from: eraseRects do: [(self subProjectorTo: eraseRect of: slide)

```
reshow]]
selectChar: char
["Set selOrigin and selCorner to the top and bottom points of a hairline along
the left edge of the char'th character of the last typeset Paragraph."
    char is: Integer⇒ [user croak]
    self selectChar: char asInteger] primitive: 59
┘
SystemOrganization classify: #Projector under: 'Documents'.┘
```

"Selection"

```
Class new title: 'Selection'
   subclassof: Object
   fields: 'proj "a Projector"
         doc "a Document, such as a Galley"
         entity "a TextEntity, for example, or false if no selection"
         oldEntity "When you want to edit entity, oldEntity (the
original) is kept around until you force the edit to take effect"
         index "entitys subscript in doc, or false if no selection"'
   declare: 'on off ';
   asFollows_[
```

I refer to an entity (or part of one, specified by a subclass) in doc at a location found quickly by index. I can be edited, and the results shown through proj, by keeping around oldEntity to compare the old and new entity size.

Initialization

```
classInit [on ← 1. off ← 0]
thru: proj
   [doc ← proj slide.
   entity ← oldEntity ← index ← false]
```

Defaults

```
complement [self complement: on]
complement: t
   [entity⇒ [entity complementIn: proj]]
enterInner
kbd
   [user kbd. proj screenAperture flash]
of: entity formerly: oldEntity  at: index from: pt thru: proj
   [self complement]
yellowbug
   [proj screenAperture flash]
```

Window Protocol

```
close
   [[entity⇒ [entity deselected]].
   self sendchanges.
   ↑ Selection new thru: proj]
enter "The window was just reentered"
   [entity ≡ false⇒ [ ]
      (index ← doc entityIndexOf: entity) ≡ false⇒
            [↑ Selection new thru: proj "That entity ceased to exist"]
      self enterInner; complement: on]
leave [self sendchanges; complement: off]
oldEntity [↑oldEntity]
oldEntity: oldEntity
redbug | pt previousEntity ind [
   pt ← user mp.
```

```
    self complement: off.
    previousEntity ← entity.
    (ind ← doc entityIndexAt: (proj ptOf: pt)) and:
    (entity ← doc∘ind)=previousEntity⇒
        [ſſself of: entity formerly: oldEntity at: ind from: pt thru: proj]
    [previousEntity⇒ [previousEntity deselected]].
    self sendchanges "if any".
    (index ← ind)⇒
        [entity selected.
        ſſentity selectionSpecies new of: entity formerly: false
        at: index from: pt thru: proj]]
sendchanges
    [oldEntity and: index⇒
        [doc entityIndex: index oldEntity: oldEntity.
        oldEntity ← false]]
┘
SystemOrganization classify: ↶Selection under: 'Documents'.┘
Selection classInit┘
```

"Style"

```
Class new title: 'Style'
   subclassof: Object
   fields: 'indentation textStyle'
   declare: '';
   asFollows↵
```

The font and indentation can be specified with Style; default is indentation=0, textStyle=DefaultTextStyle. Eventually, we will have class StyleSheet, which will be a collection of all the Styles needed for a particular document.

Initialization

```
default
   [indentation ← 0. textStyle ← DefaultTextStyle]
indentation: leftIndent right: rightIndent top: topLeading bottom: bottomLeading
   [indentation ← (leftIndent ⊙ topLeading) rect: ((0 - rightIndent) ⊙ (0 -
bottomLeading))]
textStyle: textStyle
withfont: name [
   indentation ← 0.
   textStyle ← TextStyle default setfont: 0 name: name]
```

Access To Parts

```
indentation
   [↑indentation]
textStyle
   [↑textStyle]
↵
SystemOrganization classify: ↗Style under: 'Documents'.↵
```

"TextEntity"

```
Class new title: 'TextEntity'
   subclassof: Paragraph
   fields: 'style'
   declare: '';
   asFollows_!
```

A galley (see class Galley) is a single column of contiguous entities such as TextEntities and BitRects.

Initialization

```
copy
   [↑super copy style: style]
copy: a to: b
   [↑(super copy: a to: b) style: style]
default [
   text ← ''.
   alignment ← 0.
   style ← DefaultStyle]
recopy
   [↑(self class new
      text: text copy
      runs: runs copy
      alignment: alignment)
   style: style]
style: style
text: text style: style [alignment ← 0]
```

File Conversion

```
readBravo: strm styles: dict | trailer
   [alignment ← 0.
   text ← strm upto: 032.  trailer ← strm upto: 015.
   self applyBravo: trailer at: 1 to: text length.
   style ← dict∘0.        "later look up indents in trailer"]
writeBravo: strm
   [strm append: text; append: self bravoRuns.
    "text asOop purge. runs asOops purge"]
```

Text Primitives

```
deselected
   ["I dont care"]
growthThru: proj within: domain
   [self typesetIn: proj within: domain. ↑proj frameGrowth]
selected
   ["I dont care"]
selectionSpecies
   [↑TextSelection]
showThru: proj within: domain
   ["self typesetIn: proj within: domain. proj imprint"
   proj typeset: self in: domain+style indentation as: style textStyle; imprint]
```

```
typesetIn: proj within: domain
   [proj typeset: self in: domain + style indentation as: style textStyle]
```

Array Protocol

```
= entity [↑self≡entity]
```

Formatting

```
allBold [self maskrunsunder: 1 to: 1]
allFont: n [
   [n is: String⇒ [n ← (style textStyle fontnames find: n) - 1]].
   self maskrunsunder: 0360 to: n*16]
allItalic [self maskrunsunder: 2 to: 2]
```

Aspects

```
asTextEntity
asVector [↑(Reader new of: (Stream new of: text)) read "text asVector"]
style [↑style]
```

Press

```
complete [
   "non-standard alignment signals TextEntity straddling Press pages"
   ↑alignment ≤ 4]
fromPress: press name: code value: s [
   "use control information from a press file to construct instance"
   code
   = 0⇒ [
      "text length (1 word).
      refers to same data just read by showchars (a kludge)"
      s ← s word: 1.
      s ← [(press data) skip: 0-s; next: s].
      text ← [text empty⇒ [s] text+s]];

   =1⇒ [
      "alignment"
      alignment ← s↑1];

   =2⇒ [
      "runs"
      runs ← s]
   "unrecognized code"
   ↑false]
SystemOrganization classify: ↗TextEntity under: 'Documents'.
```

"TextSelection"

```
Class new title: 'TextSelection'
   subclassof: Selection
   fields: 'c1 c2 "Integer character positions" typein sel'
   declare: 'cut esc paste Scrap bs Deletion editmenu ctlw ';
   asFollows
```

This is the text editor for a currently selected TextEntity.

Initialization

```
of: entity formerly: oldEntity at: index from: pt thru: proj [
   "init and draw out selection"
   typein ← false.
   (doc ← proj slide) typeset: index in: proj.
   self selecting: pt]
```

Showing

```
enterInner
   ["I have a selection, but it may be too big."
   typein ← false.
   c2 ← c2 min: entity length+1.
   c1 ← c1 min: c2.
   doc typeset: index in: proj]
leave
   [self fintype.
   super leave]
selecting: pt | h1 h2 c h drag2 [
   "draw out selection"
   c1 ← c2 ← proj charNearScreenPt: pt.
   h1 ← proj screenHairBeforeThatChar.
   h2 ← h1 + (1⊙0).
   proj complementScreenHairs: h1 to: h2.
   sel ← on.
   whileS user redbug doS [pt← user mp.
      c ← proj charNearScreenPt: pt.
      h ← proj screenHairBeforeThatChar.
      [c1 = c2⇒ [drag2 ← c ≥ c2]].
      [drag2⇒ [
         [c < c1⇒ [h ← proj screenHairBeforeChar: (c ← c1)]].
         proj complementScreenHairs: h to: h2.
         c2 ← c. h2 ← h]
      [c > c2⇒ [h ← proj screenHairBeforeChar: (c ← c2)]].
      proj complementScreenHairs: h1 to: h.
      c1 ← c. h1 ← h].
      h1 = h2⇒ [
         proj complementScreenHairs: h1 to: (h2 ← h1 + (1⊙0))]].
   drag2⇒ []
   "get rid of extra line in backwards select"
   proj complementScreenHairs: h2 - (1⊙0) to: h2]
show [
```

```
   doc showEdited: index in: proj.
   sel ← off]

Editing
again | t [
   [self fintype⇒ [Scrap ← Scrap text]].
   t ← entity findString: Deletion startingAt: c2.
   t=0⇒ [proj screenFrame flash]
   c1 ← t.
   c2 ← c1 + Deletion length.
   self paste]
contents [↑entity]
copy [Scrap ← self selection]
cut
   [self fintype; replace: nullString; complement. Scrap ← Deletion]
kbd | more char "key struck on the keyboard"
   [c1<c2 and: self checklooks⇒[↑self show complement]
   more ← Set new string: 16.
   [typein⇒[] Deletion ← self selection. typein ← c1].
   while: (char ← user kbdnext) do: [
      char
      =bs⇒ ["backspace"
         more empty⇒ [typein ← typein min: (c1 ← 1 max: c1-1)]
         more skip: ¯1];
      =cut⇒ [↑self cut];
      =paste⇒ [↑self paste];
      =ctlw⇒ ["ctl-w for backspace word"
         more reset.
         c1 ← 1 max: c1-1.
         while: [c1>1 and: (entity∘(c1-1)) tokenish] do: [c1 ← c1-1].
         typein ← typein min: c1];
      =esc⇒ ["select previous type-in"
         [more empty⇒[self complement: off]
         self replace: more. c1 ← c2].
         self fintype.
         c1 ← c2-Scrap length.
         ↑self complement]
      "just a normal character"
   more next← char].
   self replace: more.
   c1 ← c2.
   self selectAndScroll]
paste [self fintype; replace: Scrap; selectAndScroll]
realign [
   entity alignment ← ↷(1 2 4 0 0)∘(1+entity alignment).
   self show]
replace: t [
   [oldEntity⇒[] oldEntity ← entity copy].
   [typein≡false⇒[Deletion ← self selection]].
   "entity ←" entity replace: c1 to: c2-1 by: t.
   c2 ← c1 + t length.
   self show]
```

```
selectAndScroll [self complement: on]
selection [↑entity copy: c1 to: c2-1]
undo [self fintype; replace: Deletion; complement]
yellowbug [
   editmenu bug
      =1⇒[self undo];
      =2⇒[self copy];
      =3⇒[self cut];
      =4⇒[self paste];
      =5⇒[Scrap ← XeqCursor showwhile: "doit"
            (nil'sself selection) asTextEntity];
      =6⇒ [self realign]]
```

Private Messages

```
checklooks | t val mask [
   "see ParagraphEditor checklooks.
   substitute c1 for loc1, c2 for loc2, oldEntity for oldpara, entity for para"
   t ← ↗(166 150 137 151   230 214 201 215
      135 159 144 143 128 127 129 131 180 149
      199 223 208 207 192 191 240 226) find: user kbck.
   t=0⇒[↑false]
   user kbd.
   t=25⇒[self toBravo]; "ctl-T"
    =26⇒[self fromBravo]. "ctl-F"

   [oldEntity⇒[]oldEntity ← entity recopy].
   val ← ↗(1 2 4 256   ¯1 ¯2 ¯4 256  "ctl-b i - x   B I ¯ X"
      0 16 32 48 64 80 96 112 128 144  "ctl-0 1 ... 9"
      160 176 192 208 224 240)ot.  "ctl-shift-0 1 ... 5"
   [val=256⇒[mask← 0377.  val← 0]          "reset all"
      val<0⇒[mask← 0-val.  val← 0]         "reset emphasis"
      val>0 and: val<16⇒[mask← val]        "set emphasis"
      mask← 0360].                "set font"
   entity maskrun: c1 to: c2-1 under: mask to: val]
classInit [
   "see ParagraphEditor classInit"
   bs ← 8. ctlv ← 145. esc ← 160.
   cut ← 173. paste ← 158.

   Scrap ← Deletion ← ''.
   editmenu ← Menu new string:
'undo
copy
cut
paste
doit
align'.]
complement [self complement: on]
complement: nsel [
   nsel = sel⇒ ["already that way"]
   nsel = on and: (user rawkbck or: user redbug)⇒ ["slippage"]
   sel ← nsel.
```

```
    proj complementChars: c1 to: c2]
fintype
    [typein⇒
        [   [typein<c1⇒
                [Scrap ← entity copy: typein to: c1-1.
                c1 ← typein]].
        typein ← false]
    ⇑false]⌟
⌟
SystemOrganization classify: ↱TextSelection under: 'Documents'.⌟
TextSelection classInit⌟
```

```
Class new title: 'DefineVariables'
   subclassof: Object
   fields: ''
   declare: '';
   asFollows_!
```

A special-purpose piece of code to define some of the shared variables.
To be executed one time, automatically, as the class is read in.

Initialization

```
classInit
[Smalltalk declare: ↗(E EtherPool).
[EtherPool≡nil⇒ [EtherPool←SymbolTable new init: 32]].

EtherPool declare: ↗(NUMPACKETS  MAXNUMSOCKETS  ROUTINGTABLESIZE )
   as: (10,10,10).

EtherPool declare: ↗(EthHeaderBytes PupHeaderBytes BufHeaderBytes
   EthOvBytes PupOvBytes BufOvBytes )
   as: (4,20,24,4,22,26).

EtherPool declare: ↗(FirstDataByte FirstDataWord MaxDataBytes
   MaxDataWords MaxBufBytes MaxBufWords )
   as: (25,13,532, 266,558,279 ).

EtherPool declare: ↗(ethInPac ethInPacNext ethInPacNextPtr
   broadcastFilter printBadPost checkIncomingCS systemNotInited
   IntProcLevel InputProcLevel)
   as: (false,false,false,true,false,false,true,14,13).

EtherPool declare: ↗(
   NETNUM ALTONUM SOCSEED  DoubleZero
   soc0 soc2 soc5
   freeQ justArrivedQ nonPupInQ
   socketTable routingTable routingHopCount routingUpdateUser
   broadcastListener  IntProc InputProc
   IntLight InputLight OutputLight ExtraLight  ).

EtherPool declare: ↗(TYPEPUP IDLE TRANSMITTING RECEIVING ethIntBits )
   as: (01000, 0, 1, 2, 020).

EtherPool declare: ↗(
   ETHPOSTLOC ETHINTBITLOC ETHLOADLOC ETHINCNTLOC
   ETHRECLOC ETHOUTCNTLOC ETHOUTLOC ETHIDLOC)
   as: (0600,0601,0603,0604,0605,0606,0607,0610).

EtherPool declare: ↗( SIONOP SIOSTARTTRANS SIOSTARTREC SIORESET)
   as: (0,1,2,3).

EtherPool declare: ↗( ETHMICROSTAT
```

```
    ETHINPUTOK ETHOUTPUTOK ETHPACTOOLONG ETHCOLLISIONS
    ETHOBUF ETHRESET ETHBROKENPAC)
    as: (136,0377,0777,2,3,4,5,246).]
↵
SystemOrganization classify: ↗DefineVariables under: 'Ethernet Control'.↵
DefineVariables classInit↵
```

"EFTPReceiver"

```
Class new title: 'EFTPReceiver'
   subclassof: Socket
   fields: 'seqNum'
   declare: '';
   asFollows↵
```

This class has not yet been commented

As yet unclassified

```
↵
SystemOrganization classify: #EFTPReceiver under: 'Ethernet Control'.↵
```

"EFTPSender"

```
Class new title: 'EFTPSender'
   subclassof: Socket
   fields: 'seqNum outPac myStream eof abortTransfer
      reTransCount reTransMax reTransTimer ackOK'
   declare: '';
   asFollows_!
```

A specialized sub-class of Socket, designed to send files to an EFTP receiver. By convention, the receiver will be on socket 020 There can only be one outstanding packet at a time, called outPac.

Initialization

```
net: n host: h
   [
   "Each instance of an EFTPSender has a unique lclSocket, but
      always goes to socket 020 of the receiver"
   super net: n host: h soc: (Int32 new high: 0 low: 020).
   "unlike plain sockets, we only want acks from this dest."
   filterInput ← true.
   reTransTimer ← Timer new.
   reTransTimer for: 120 actions [self reTransmit].
   reTransMax ← 20.
   outPac ← false.
   ]
to: h
   [
   "convenient default if on my net"
   self net: NETNUM host: h.
   ]
```

Termination

```
close
   [
   self reset.
   super close.
   ]
reset
   [
   reTransTimer disable.
   [outPac ⇒ [freeQ next ← outPac. outPac ← false]].
   ]
```

Sending

```
send: myStream
   [
   "let the caller hand in a stream, or a file already opened"
   [outPac ⇒ []
      outPac ← freeQ next ⇒
         [outPac transmittedQ ← false] "no special Q for retransmission"
```

```
        user show: 'no packets on freeQ'. ↑false].
    seqNum ← 0.
    abortTransfer ← eof ← false.
    until: [eof or: abortTransfer] do: [self sendData. seqNum←seqNum+1].
    abortTransfer ⇒ [self reset. ↑false].
    "We hit the end of file, do the end sequence"
    self sendEnd1.
    abortTransfer ⇒ [self reset. ↑false].
    seqNum←seqNum+1.
    self sendEnd2.
    self reset.
    ↑myStream. "all done!"
    ]
sendData | i t buf
    [
    "long comment if necessary"
        outPac pupType ← 030. "Data"
        "fill the buffer, speed this up later, watch out here!!!"
        buf ← outPac pupString.
        i ← 25.  "data bytes are 1-512, 25-536"
        while: (i<537 and: eof≡false) do:
            [
            t←myStream next ⇒[buf↓i←t. i ← i+1.]
            eof←true.
            ]
        "set the packet length"
        outPac pupLength ← i-3.
        self sendPacket. "return"
    ]
sendEnd1
    [
        "send end, wait for ack"
        outPac pupType ← 032. "end"
        "set the packet length"
        outPac pupLength ← 22.
        self sendPacket. "return"
    ]
sendEnd2
    [
        "send the last gratuitous end"
        outPac pupType ← 032. "end"
        "set the packet length"
        outPac pupLength ← 22.
        outPac pupID ← Int32 new high: 0 low: seqNum.
        self setAddressesAndComplete: outPac.
        "don't try to retransmit"
    ]
sendPacket
    [
    "General routine to send packets, wait for ack, retransmit"
    outPac pupID ← Int32 new high: 0 low: seqNum.
    reTransCount ← 0.
```

```
ackOK ← false.
reTransTimer reset.
self setAddressesAndComplete: outPac.
"it's gone!,  set the retransmission timer"
until: [abortTransfer or: ackOK] do: [ ].
]
```

Timer Interrupts

reTransmit

```
[
"This piece of code only runs when a timer fires!
Thus, there is mututal exclusion between this and other timer code.
Don't do an active return"
reTransMax = (reTransCount ← reTransCount + 1) ⇒
   [
   user show: 'excessive retransmits, in EFTP retransmit'.
   abortTransfer ← true. "returns"
   ]
"go ahead and retransmit....."
self completePup: outPac.
reTransTimer reset. "returns"
]
```

Overwrite from Socket

socProcess: pac

```
[
"an input has arrived, we are running at a higher level.
Input has been filtered by source, need only check seqNum"
(pac pupType = 031) and: (pac pupID1 = seqNum) ⇒
   [reTransTimer disable. ackOK←true. freeQ next ← pac]
"bad ack"
freeQ next ← pac.
]
```

```
SystemOrganization classify: ↗EFTPSender under: 'Ethernet Control'.
```

"Etherworld"

```
Class new title: 'Etherworld'
   subclassof: Object
   fields: ''
   declare: '';
   sharing: EtherPool;
   asFollows↵
```

This is, of course, the class that controls all of the basic ethernet operations. There should not be more than one EtherWorld, and one, E, has to be defined for the system to work.

In this implementation, and due to timing considerations, it is expected that the transmitter will post quite quickly; thus, we disable interrupts and busy wait for its completion.

In general, the interrupt is only armed when we have started the receiver. The Etherworld currently uses these input processes in the PrioityScheduler:
IntProc, at IntProcLevel (14) -- awakened when the device interrupts
InputProc, at InputProcLevel (13) -- distributes packets to sockets, allowing each socket to then run.

Note that some of the timers may be on other levels.

Uses a whole bunch of global variables, initialized only once, when originally read into a system.

The message systemInit is used each time a user wants to begin using the communications facilities. This would have to be done anytime one moves to a different machine. EtherKill is its complement.

etherStop and etherStart are more gentle, temporarily stopping any communications, but preserving a lot of state.

Global boolean 'broadcastFilter', if true, prevents reception of packets sent to host zero.

The lights on the right side of the screen are Etherworld signals. They mean (from top to bottom) Etherworld awakened; packet addressed to the Alto received; packet being processed; output being sent; and input rejected.

Several variables point at important buffers: ethInPac, ethInPacNext, and ethInPacNextPtr are crucial for running the receiver.

Initialization

```
setLights
   [
   IntLight← Rectangle new origin: 592⊙100 corner: 608⊙116.
   IntLight comp.
   InputLight← Rectangle new origin: 592⊙220 corner: 608⊙236.
   InputLight comp.
```

```
    OutputLight←Rectangle new origin: 592⊙440 corner: 608⊙456.
    OutputLight comp.
    ExtraLight← Rectangle new origin: 592⊙550 corner: 608⊙566.
    ExtraLight comp.

    IntLight comp. InputLight comp.  OutputLight comp. ExtraLight comp
    ]
systemInit
    [
    self systemInitA.
    self systemInitB.
    self installIntProc.
    self installInputProc.
    self systemInitC.
    systemNotInited ← false.
    "start to collect a new routing table"
    routingUpdateUser update.
    ]
systemInitA
    [
    "May have been called when things were in a clean state,
    or perhaps right in the middle of running.  Thus, we assume the worst: "
    self etherStop.
    Top terminate: IntProcLevel; terminate: InputProcLevel.
    self freePackets.

    ALTONUM ← self getMachineID.
    self setMachineID: ALTONUM.
    NETNUM←0. "gets corrected later"
    SOCSEED ← 200.   "maybe use clock as seed"
    self setLights.

    freeQ← PQueue new of: (Vector new: (2*NUMPACKETS)) .
    justArrivedQ← PQueue new of: (Vector new: (2*NUMPACKETS)) .
    nonPupInQ← freeQ. "if not Pup, throw it away"
    ]
systemInitB | i
    [
    for: i to: NUMPACKETS-2 do: [freeQ next← Pacbuf init].
    mem∘ETHRECLOC ← (ethInPac ← Pacbuf init) lock.
    ethInPacNextPtr ← (ethInPacNext← Pacbuf init) lock.
    mem∘ETHINCNTLOC ← MaxBufWords.
    socketTable← Dictionary new init: MAXNUMSOCKETS.
    routingTable← String new: 255.
    routingTable all ← 0. "1-255, 0 is special"
    routingHopCount ← String new: 255.
    routingHopCount all ← 8.
    routingUpdateUser ← RoutingUpdater init.
    ]
systemInitC
    [
    IntProc enable.
```

```
    InputProc enable.
    self etherStart.
    ]

Input Interrupt Routines
installInputProc | inBuf destSoc
    [
    InputProc ← Top install:
        [
        while: [true] do:   "infinite loop for process in scheduler"
            [
            InputLight comp.
            while: [inBuf←justArrivedQ next] do:
                [
                "process each incoming buffer
                make sure it's a PUP"
                inBuf ethType ≠ 01000 ⇒ [nonPupInQ next← inBuf "done"]
                "verify the incoming checksum"
                checkIncomingCS and: ((inBuf checksumOK)≡false)⇒
                    ["reject it" freeQ next← inBuf "done"]
                "To be honest, we should check the destNet and destHost,
                but they generally have to be OK....."
                "OK to pass the packet on"
                (destSoc ← socketTable lookup: (inBuf destSocNum)) ⇒
                    [ destSoc acceptPacbuf: inBuf].
                "couldn't find a socket for it"
                 freeQ next← inBuf
                "done with this packet"
                ].
            InputLight comp.
            InputProc sleep   "last action in the loop"
            ]
        ]
    at: InputProcLevel
    ]
installIntProc | postcode i tempPac post
    [
    IntProc ← Top install:
        [
        while: [true] do:   "infinite loop for process in scheduler"
            [
            "Interrupt just happened, running at a high level, interface off.
            Something just happened, do the common cases first.
            ethInPac should be under the rec.
            Note: we can only come here if last action was to start the rec!!"

            IntLight comp.
            postcode←mem∘0600.
            "now, try to get the receiver started quickly"
            [ethInPacNext ⇒
                [
                "On deck packet OK, can now swap"
```

```
            mem o 0605 ← ethInPacNextPtr.
            self SIO: 2.
            tempPac ← ethInPac.
            ethInPac unlock.
            ethInPac← ethInPacNext.
            [(ethInPacNext ← freeQ next)⇒
               [ethInPacNextPtr←ethInPacNext lock]].
            "Now process that input we got, sitting in tempPac, check inputOK"
            [postcode=0377⇒
               [
               "let it through, unless the broadcast filter is on"
               broadcastFilter and: tempPac imEthDestHost=0 ⇒
                  [freeQ next← tempPac]
               justArrivedQ next← tempPac. Top wakeup: InputProcLevel
               ]
            if: [printBadPost] then:
               [user cr. user show: 'Bad Ethernet postcode: ' + postcode base8].
            freeQ next← tempPac.
            ]
            ]
         "On deck packet not OK, try to sort it out, don't process the input"
         ethInPac⇒
            [
            self SIO: 2.
            (ethInPacNext ← freeQ next)⇒
               [ethInPacNextPtr←ethInPacNext lock].
            "All done"
            ]
         "Things are really sour here -- posted with neither buffer OK!!"
         ethInPac ← freeQ next⇒
               [
               mem o 0605 ← ethInPac lock. self SIO: 2.
               (ethInPacNext ← freeQ next)⇒
                  [ethInPacNextPtr←ethInPacNext lock].
               "All done"
               ]
         "Couldn't get anything restarted, just give up....."
         ].

         IntLight comp.
         IntProc sleep    "last action in the loop"
         ].
      ]
   at: IntProcLevel.
   ]
```

Output Routines

```
doOutput [] primitive: 100
sendOutput: ethOutPac | post
   [
         "This is the one and only place from which we  send output.
         Only one packet gets passed in to us at a time.
```

```
        For performance, we wait here for the transmitter to post!!!!
        Nominally, we are running at level 0;  thus, this must be run
        at a Top critical, to protect from multiple calls."
        systemNotInited ⇒ [user notify: 'Ethernet not inited!!'].
        Top critical:
          [
          OutputLight comp.
          mem∘0606← (ethOutPac totLengthWords)."EthOutCntLoc"
          mem∘0607←(ethOutPac lock).  "EthOutLoc"
          [0 = (mem∘0605) ⇒
            [user cr. user show: 'Warning, will not start input after output. ']].
          [self doOutput ≠ 0777 ⇒
            [user cr. user show: 'Warning, bad output post: '+ post base8]].
          ethOutPac unlock.
          ethOutPac transmitted.
          OutputLight comp.
          ]. "end of the critical part"
    ]
```

User messages

```
etherIntDisable "disables the device interrupt, uses bit mask in ethIntBits"
    ["doesn't try to actually stop the device"
    mem∘ETHINTBITLOC ← 0]
etherIntEnable "enables the device interrupt, uses bit mask in ethIntBits"
    ["doesn't try to actually start the device"
    mem∘ETHINTBITLOC ← ethIntBits]
etherKill    "shuts down ethernet and PUP world completely"
    [
      "Should free up all of the storage, etc.....
      Would need to systemInit, to get started again.
      Device may have been running"
      self etherStop.
      Top terminate: IntProcLevel; terminate: InputProcLevel.
      self freePackets.
      freeQ ← nil.  justArrivedQ ← nil.  nonPupInQ ← nil.
      routingUpdateUser ← routingTable ← routingHopCount ← nil.
      systemNotInited ← true.
    ]
etherStart "allows ether to start running again"
    [
    "makes sure the interrupt is on, and kicks the device"
    [mem∘ETHINTBITLOC=0⇒ [mem∘ETHINTBITLOC←ethIntBits]].
    self SIO: SIORESET   "forces it to wake up again"
    ]
etherStop "temporarily shuts off the ether stuff"
    [
    "leaves input packet in place"
    Top critical:
      [
      mem∘ETHINTBITLOC←0.
      self SIO: SIORESET.
      mem∘ETHPOSTLOC ← 0.
```

```
    ]
  ]
freePackets "releases the special packets, sets names to false"
  [
    memoETHRECLOC ← 0.
    [ethInPac⇒
      [[ethInPac locked⇒[ethInPac unlock]]. ethInPac←false] ].
    [ethInPacNext⇒
      [[ethInPacNext locked⇒[ethInPacNext unlock]]. ethInPacNext←false] ].
  ]
```

Utility messages

```
error: str [user cr. user show: str]
getMachineID  [↑ (self SIO: SIONOP) \ 256]
printRoutingTable | i
  [
  for: i from: 1 to: 255 do:
    [
    routingTableoi ≠ 0 ⇒
      [
      user cr.  user show: 'To net ' + i asString +
        ' via host ' + (routingTableoi) asString + ', hop count = ' +
        (routingHopCountoi) asString.
      ]
    ]
  user cr.
  ]
setMachineID: ID [memoETHIDLOC ← ID]
SIO: sioArg [] primitive: 91
_!
SystemOrganization classify: ↗Etherworld under: 'Ethernet Control'._!
```

"Int32"

```
Class new title: 'Int32'
   subclassof: Number
   fields: 'high low'
   declare: '';
   asFollows↵
```

This class should probably be part of Number rather than Etherworld.
NOTE THAT + AND - SHOULD BE FIXED TO RETURN TO SMALLTALK IF TYPE OF ARG IS NOT INT32

Initialization

```
asInteger [high = 0 ⇒ [↑ low] ↑ false]
classInit [Integer understands: 'asInt32 [↑ Int32 new high: 0 low: self]'.]
high: high low: low
```

Info about self

```
hash [↑ low]
high [↑ high]
low [↑ low]
printon: strm
   [high printon: strm base: 8. strm append: '|'. low printon: strm base: 8 ]
```

Arithmetic

```
≠ arg [high= arg high ⇒[↑ low ≠ arg low] ↑ true]
+ arg [↑self + arg asInt32] primitive: 93
- arg [↑self + arg asInt32] primitive: 92
< arg [high > arg high ⇒[↑ false] high=arg high⇒[↑low<arg low]
   ↑ true]
= arg [high= arg high ⇒[↑ low = arg low] ↑ false]
> arg [high < arg high ⇒[↑ false] high=arg high⇒[↑low>arg low]
   ↑ true]
↵
SystemOrganization classify: ↗Int32 under: 'Ethernet Control'.↵
Int32 classInit↵
```

"Pacbuf"

```
Class new title: 'Pacbuf'
   subclassof: Object
   fields: 'transmittedQ pupString locked'
   declare: '';
   sharing: EtherPool;
   asFollows_!
```

This is the basic unit for building and interpreting packets for the ethernet. It contains the messages that allow fields of a packet to be filled and read. Most users will prefer to use the socket mechanisms

Initialization

```
init
   [pupString ← String new: MaxBufBytes.
   transmittedQ ← freeQ.
   locked ← false]
```

Ethernet header

```
ethType [↑pupString word: 2]
ethType← eT [pupString word: 2 ← eT]
imEthDestHost [↑pupString○1]
imEthDestHost← iEDH [pupString○1 ← iEDH]
imEthSrcHost [↑pupString○2]
imEthSrcHost← iESH [pupString○2 ← iESH]
```

PUP Header

```
addressBlock [↑pupString○(13 to: 24) ]
addressBlock← addBlock "for quickly setting the 6 fields"
   [pupString○(13 to: 24) ← addBlock]
destHost [↑pupString○14]
destHost← dH [pupString○14 ← dH]
destNet [↑pupString○13]
destNet← dN [pupString○13 ← dN]
destSoc0 [↑pupString word: 8]
destSoc0← i [↑pupString word: 8←i]
destSoc1 [↑pupString word: 9]
destSoc1← i [↑pupString word: 9←i]
destSocNum [↑Int32 new high: (pupString word: 8) low: (pupString word: 9) ]
destSocNum← dSN [pupString word: 8 ← dSN high.
                 pupString word: 9 ← dSN low]
pupID [↑Int32 new high: (pupString word: 5) low: (pupString word: 6) ]
pupID0 [↑pupString word: 5 ]
pupID0← pID [↑pupString word: 5 ← pID ]
pupID1 [↑pupString word: 6 ]
pupID1← pID [↑pupString word: 6 ← pID ]
pupID← pID [pupString word: 5 ← pID high.
                 pupString word: 6 ← pID low]
pupLength [↑pupString word: 3]
pupLength← pL [↑pupString word: 3 ← pL]
```

```
pupType [↑pupString↓8]
pupType← pT [pupString↓8 ← pT]
sourceHost [↑pupString↓20]
sourceHost← sH [pupString↓20 ← sH]
sourceNet [↑pupString↓19]
sourceNet← sN [pupString↓19 ← sN]
sourceSoc0 [↑pupString word: 11]
sourceSoc0← i [↑pupString word: 11←i]
sourceSoc1 [↑pupString word: 12]
sourceSoc1← i [↑pupString word: 12←i]
sourceSocNum [↑Int32 new high: (pupString word: 11) low: (pupString word:
12)]
sourceSocNum← sSN [pupString word: 11 ← sSN high.
                   pupString word: 12 ← sSN low]
swapPorts | i temp "try to do it quickly!!"
   [
   for: i from: 13 to: 18 do:
      [
      temp ← pupString↓i.
      pupString↓i ← pupString↓(i+6).
      pupString↓(i+6) ← temp.
      ]
   ]
totLengthWords [↑((self pupLength)+5)/2]
transportControl [↑pupString↓7]
transportControl← tC [pupString↓7 ← tC]
```

PUP Checksum

```
checksum  [↑pupString word: ((self pupLength+1)/2)+2]
checksumOK "Boolean, returns true or false"
   ["just look at the current packet"
   ↑self checksum = self doChecksum]
checksum← cs
    [pupString word: (((self pupLength+1)/2)+2) ← cs]
doChecksum | i cs
   [
   cs ← 0.
   for: i from: (3 to: (((self length + 1)/2)+2)) do: "does not work"
      [cs←cs+(pupString word: i).                    "for packets with carries"
      [cs <0⇒[cs ← (cs lshift: 1)+1] cs ← cs lshift: 1]].
   [cs=¯1⇒[cs ← 0]].
   ↑cs
   ] primitive: 94
```

Data

```
dataLength [↑self pupLength - 22]
dataString | i
   [↑(pupString↓(25 to: 24+self dataLength)) copy ]
dataString← str | i
   [i ← str length.
   i > MaxDataBytes ⇒ [user notify: 'Data string too big for PUP']
```

```
    self pupLength ← 22 + i.
    ↑pupString o (25 to: 24 + i) ← str.
    ]

Etc
header [↑pupString o (1 to: 24) ]
lock [locked⇒[user notify: 'trying to lock a buffer already locked']
        locked←true. ↑pupString lock]
locked [↑locked]
pupString [↑pupString]
pupString ← pupString [↑pupString]
transmitted
    [
    "packet has been transmitted, put on Q, if requested.
    Does nothing if transmittedQ has been set false"
    transmittedQ ⇒ [transmittedQ next← self].
    ]
transmittedQ [↑transmittedQ]
transmittedQ← transmittedQ
unlock [locked⇒[locked←false. pupString unlock]
        user notify: 'trying to unlock a buffer not locked']
┘
SystemOrganization classify: ↗Pacbuf under: 'Ethernet Control'.┘
```

"RoutingUpdater"

```
Class new title: 'RoutingUpdater'
   subclassof: Socket
   fields: 'timer'
   declare: '';
   asFollows_!
```

A specialized sub-class of Socket, designed to send out requests for the current routing info, and update the routing table.

Initialization

```
init
   [
   "create a new local soc number, broadcast to socket 2"
   super net: 0 host: 0 soc: (Int32 new high: 0 low: 02).
   timer ← Timer new.
   ]
```

Termination

```
close
   [
   self reset.
   super close.
   ]
reset
   [
   timer disable.
   ]
```

Sending

```
update | outPac i
   [
   outPac ← freeQ next ⇒
      [
      for: i to: 255 do:
         [
         routingTable•i ← 0. routingHopCount•i←8.
         ].
      outPac pupType ← 0200.
      outPac dataString ← ''.
      outPac transmittedQ ← freeQ.
      "enable me for input"
      self enable.
      self setAddressesAndComplete: outPac.
      timer for: 600 action: [self stopListening].
      timer reset.
      ]
   user show: 'freeQ empty, in RoutingUpdater'.
   ↑false.
   ]
```

Timer Interrupts

```
stopListening
   [
   "This piece of code only runs when a timer fires!
   Thus, there is mututal exclusion between this and other timer code.
   Don't do an active return.
   Takes me out of the socketTable"
   self disable. "returns"
   ]
```

Overwrite from Socket

```
socProcess: pac | block gateway net count i
   [
   "an input has arrived, we are running at a higher level.
   Check the packet type"
   if: pac pupType = 0201 then:
      [
      if: NETNUM=0 then: [NETNUM←pac sourceNet].
      block ← pac dataString.
      gateway ← pac sourceHost.
      for: i from: (1 to: block length by: 4) do:
         [
         net ← block∘i.
         count ← block∘(i+3) + 1.
         count < (routingHopCount∘net) ⇒
            [routingTable∘net ← gateway. routingHopCount∘net←count]
         ]
      ].
   freeQ next ← pac.
   ]
↵
SystemOrganization classify: ↗RoutingUpdater under: 'Ethernet Control'.↵
```

"SafeQ"

```
Class new title: 'SafeQ'
   subclassof: PQueue
   fields: ''
   declare: '';
   asFollows↵

checks all objects enqueued, to be sure not there already

As yet unclassified
length [↑position - readposition]
next ← arg | i "short comment"
   [
   for: i from: (readposition+1) to: position do:
      [
      (array:i)≡arg⇒[user notify: 'putting same guy on Q twice']
      ]
   super next ← arg
   ]
↵
SystemOrganization classify: ↱SafeQ under: 'Ethernet Control'.↵
```

"Socket"

```
Class new title: 'Socket'
   subclassof: Object
   fields: 'socNumber computeOutgoingCS filterInput outAddBlock
   lclSocNum frnNet frnHost frnSocNum'
   declare: '';
   sharing: EtherPool;
   asFollows_[
```

Sockets are used to do all communication through the net.
It is expected that a specialized server or process can have
its own subclass of Socket with its own definitions of the
'Overwrite by Subclass' operations. Note that subclasses will
have to access some global variables.

Each socket is identified by a 32-bit socNumber, which really
defines who we are.
In addition, aspects of the lcl and frn addresses are used to make
decisions about accepting
incoming packets, addressing outgoing packets, defaulting fields, etc.

The input distributor assures that an input was destined for our net
(not trying to
find a gateway) and our host (either explicitly or as broadcast,
if not filtered), and found us by socket number. Input need NOT be
filtered by the socket according to source, since the client may want
to see error messages from an intermediate address.

As a convenience, however, the socket can be asked to filterInput,
so it only accepts things which match the frnPort.
Thus, local and foreign attributes are primarily used to default
fields of an outgoing packet.

Initialization

default

```
   ["default local socket number and leave frn port open"
   socNumber ← Int32 new high: 0 low: (SOCSEED ← SOCSEED + 1).
   self from: socNumber net: 0 host: 0 soc: (Int32 new high: 0 low: 0)
   ]
```

from: socNumber

```
   ["set lcl soc number, leave frnPort open -- useful for creating
   a well-known socket as a listener"
   self from: socNumber net: 0 host: 0 soc: (Int32 new high: 0 low: 0)
   ]
```

from: socNumber net: frnNet host: frnHost soc: frnSocNum

```
    [
   "this is the most general initialization, both lcl soc# and frnPort given"
   outAddBlock ← String new: 12.
   lclSocNum ← socNumber.
   self setOutAddBlock.
```

```
    computeOutgoingCS ← filterInput ← false.
    self doMoreInit
    ]
net: frnNet host: frnHost soc: frnSocNum
    ["default the local socket number"
    socNumber ← Int32 new high: 0 low: (SOCSEED ← SOCSEED + 1).
    self from: socNumber net: frnNet host: frnHost soc: frnSocNum
    ]
setOutAddBlock
    [
    outAddBlock↑1 ← frnNet. outAddBlock↑2 ← frnHost.
    outAddBlock word: 2 ← frnSocNum high.
    outAddBlock word: 3 ← frnSocNum low.
    outAddBlock↑7 ← NETNUM. outAddBlock↑8 ← ALTONUM.
    outAddBlock word: 5 ← lclSocNum high.
    outAddBlock word: 6 ← lclSocNum low.
    ]
```

Process incoming packet

```
acceptPacbuf: lpac | temp
    ["if we get here, we know that the input distributer has verified the
    PUP dest as being us (or a broadcast, if broadcast filter is off).
    We do not have responsibility for verifying incoming checksum.
    First, check if we've been asked to filter by source:"
    filterInput and:
        (   (frnNet ≠ lpac sourceNet) or:
            ((frnHost ≠ lpac sourceHost) or: (frnSocNum ≠ lpac sourceSocNum))
        )
        ⇒ [↑self socDispose: lpac]
    "It's good, take it..."
    ↑self socProcess: lpac
    ]
```

Process outgoing packet

```
completePup: pac | t
    [
    "the user must have set all 6 address fields,ID, length, and type"
    "Now route the packet appropriately, assuming we have Ethernet..."
        [
        NETNUM = pac destNet ⇒ [pac imEthDestHost ←pac destHost]
            "most common case"
        0 = pac destNet ⇒ [pac imEthDestHost ← 0] "broadcast"
        0 = (t ← routingTable↑(pac destNet)) ⇒
            [
            user show: '
Inaccessible destination net: ' + pac destNet as String+ ', packet net sent.'.
            pac transmitted.
            ↑pac.
            ].
        pac imEthDestHost ← t.
        ].
```

```
    pac imEthSrcHost ← ALTONUM.
    pac ethType ← TYPEPUP.
    pac transportControl← 0.

    "as a socket we have an option about computing outgoing checksums"
    pac checksum ← [computeOutgoingCS⇒[pac doChecksum] ~1].

    "Fix this up later....."
    E sendOutput: pac.
    ↑pac
    ]
defaultAddresses: pac   "overwrites any fields which are 0"
    [
    [pac destNet = 0 ⇒ [pac destNet ← frnNet]].
    [pac destHost = 0 ⇒ [pac destHost ← frnHost]].
    [(pac destSoc0 = 0) and: (pac destSoc1=0) ⇒
        [pac destSocNum ← frnSocNum]].
    [pac sourceNet = 0 ⇒ [pac sourceNet ← NETNUM]].
    [pac sourceHost = 0 ⇒ [pac sourceHost ← ALTONUM]].
    [(pac sourceSoc0 = 0) and: (pac sourceSoc1 = 0) ⇒
        [pac sourceSocNum ← lclSocNum]].
    ]
defaultAndComplete: pac
    [
    self defaultAddresses: pac.
    self completePup: pac.
    ]
setAddresses: pac [pac addressBlock ← outAddBlock]
setAddressesAndComplete: pac
    [pac addressBlock ← outAddBlock. self completePup: pac]
```

Access to Parts

```
close       [self disable]
computeOutgoingCS [↑computeOutgoingCS]
computeOutgoingCS← computeOutgoingCS [↑computeOutgoingCS]
disable     [sockeTable lookup: socNumber⇒[sockeTable delete: socNumber]]
enable [sockeTable insert: socNumber with: self]
filterInput  [↑filterInput]
filterInput← filterInput [↑filterInput ]
frnHost [↑frnHost]
frnHost← frnHost [↑frnHost]
frnNet [↑frnNet]
frnNet← frnNet [↑frnNet]
frnSocNum [↑frnSocNum]
frnSocNum← frnSocNum [↑frnSocNum]
lclSocNum [↑lclSocNum]
lclSocNum← lclSocNum [↑lclSocNum]
socNumber  [↑socNumber]
socNumber ← socNumber  [↑socNumber]
```

Overwrite by Subclasses

```
doMoreInit
socDispose: lpac [freeQ next← lpac]
socProcess: lpac [ freeQ next← lpac]
↵
SystemOrganization classify: ↶Socket under: 'Ethernet Control'.↵
```