"Player"

```
Class new title: 'Player'
   subclassof: Object
   fields: 'location angle size costume script'
   declare: 'myTurtle random runflag r ';
   asFollows_┘
```

This class has not yet been commented

As yet unclassified

```
aShow
   [self erase.
    self perform: script.
    self show]
classInit
   [myTurtle ← Turtle init. myTurtle width: 2.
    r ← 40 to: 100.

"Add a new message to Point to go in a direction"
    Turtle understands:
      'loc [ ↑ x ⊙ y]'.
    Point declare: 'pointTurtle'.
    Point understands:
      'classInit [pointTurtle ← Turtle init. pointTurtle penup]'.
    Point classInit.
    Point understands:
      'go: dist at: angle
         [pointTurtle up; goto: self; turn: angle; go: dist.
          ↑ pointTurtle loc.]'

   ]
cycle [runflag ← false. self play]
erase [myTurtle white. self putOnCostume]
play
   [self aShow.
    runflag ⊃ [self after: 0.1 do: ↑play]]
putOnCostume
   [myTurtle penup; goto: location; up; turn: angle; pendn.
    self perform: costume]
run [runflag ← true. self play]
script [↑script]
script: script
show [myTurtle black. self putOnCostume]
stop [runflag ← false]
_┘
SystemOrganization classify: ↑Player under: 'Players and Boxes'._┘
Player classInit_┘
```

"Box"

```
Class new title: 'Box'
   subclassof: Player
   fields: 'Pix s'
   declare: '';
   asFollows_]
```

This class has not yet been commented

```
As yet unclassified
angle: angle
car1
   [self shape. self cartop]
car2
   [self shape2. self cartop]
cartop
   [myTurtle penup; goto: location; up; turn: angle;
    go: 6*s; pendn;
    turn: 132; go: 11*s; go: ¯11*s;
    turn: ¯272; go:  11*s.

    myTurtle penup; goto: location; up; turn: angle; pendn;
    turn: 100; go: 8*s;
    turn: 103; go: 18*s;
    turn: 130; go: 17*s;
    turn: 97; go: 7*s.
   ]
clr | R
   [R ← Rectangle new fromuser.
    R fillin: ltgray; fillin: ltgray; fillin: gray]
clr: clr | R
   [R ← Rectangle new fromuser fillin: clr]
costume: costume
drawing | i
   [Pix reset.
    for: i from: Pix do:
      [myTurtle line: i].
   ]
fetch | mx mn i ctr
   [Pix ← (Vector new: 50) asStream. myTurtle black.
    until: user bluebug do:
      [user redbug ⊃ [Pix next ← self line]].
    Pix close.

   Pix reset. mx ← 0⊙0. mn ← 800⊙800.
    for: i from: Pix do:
         [mx ← i max: mx.
          mn ← i min: mn].
    myTurtle frame: (mn rect: mx).

   Pix reset. ctr ← (mx - mn)/2
    for: i from: Pix do:
      [i x: (i x - ctr x) y: (i y - ctr y)].
   ]
go: dist
   [self erase.
    location ← location go: dist at: angle.
```

```
    self show]
goit: dist | R
[R ← Rectangle new fromuser clear.
  self go: dist.
  self clr: ltgray.
]
grow: s
  [self erase.
   size ← size + s.
   self show]
init
  [location ← (r random + 200) ⊙ (r random + 200).
   angle ← (r random - 40) * 6.
   s ← 2.
   costume ← ↗shape.
   self show]
line | start end width
  [start←user mp-myTurtle frame origin.
   width←myTurtle width.  myTurtle xor; width: 1.
   while: user redbug do:
     [end←user mp-myTurtle frame origin.
     myTurtle xor; place: start; goto: end; place: start; goto: end].
   myTurtle black.
   myTurtle width: 2; place: start; goto: end.
   ⋂ start line: end]
move
  [self erase.
   until: user redbug do: [].
   location ← user mp.
   self show]
placit | R
[R ← Rectangle new fromuser clear.
until: user anybug do: [].
location ← user mp.
self show; clr.
]
shape
  [myTurtle penup; go: 24*s; pendn;
   turn: 162; go: 50*s;
   turn: 130; go: 15*s;
   turn: ¯50; go: 15*s;
   turn: 131; go: 50*s.
  ]
shape2
  [myTurtle penup; go: 12*s; pendn;
   turn: 135; go: 11*s;
   turn: 45; go: 24*s;
   turn: 45; go: 10*s.
 myTurtle penup; goto: location; up; turn: angle;
   go: 12*s; pendn;
   turn: ¯135; go: 10*s;
   turn: ¯45; go: 24*s;
   turn: ¯45; go: 10*s.
 ]
square | i
  [myTurtle penup; go: size*2; pendn;
   turn: 165; go: size*3;
   turn: 136; go: size;
```

```
    turn: ¯60; go: size;
    turn: 136; go: size*3]
travel
   [self turn: (user mp x - 250)/10. self go: 10]
trick1
   [self grow: 2. self turn: 11]
trick2
   [self grow: ¯2. self turn: 13]
turn: a
   [self erase.
    angle ← angle + a.
    self show]
turnit: ang | R
[R ← Rectangle new fromuser clear.
   self turn: ang.
   self clr: ltgray.
]
SystemOrganization classify: ↑Box under: 'Players and Boxes'.
```

"SpaceShip"

```
Class new title: 'SpaceShip'
  subclassof: AnObsoleteBox
  fields: 'speed joystick'
  declare: '';
  veryspecial: 5;
  asFollows_J
```

This class has not yet been commented

```
As yet unclassified
init
  [speed ← 0 ⊙ 0.
   location ← 250 ⊙ 250.
   angle ← 0.
   size ← 10.
   costume ← ↗ship.
   self show]
joystick: joystick
rehearse
  ["SS1 ← SpaceShip init. SS1 joystick: joystick1.
    SS1 init.
    until: user bluebug do:[SS1 thrust]"

]
ship
  [myTurtle
    go: (size * 4) / 10;
    turn: 150; go: size;
    turn: 120; go: size;
    turn: 120; go: size.]
thrust
  [self erase.
"---------------"
   angle ← angle + joystick x.
   speed ← speed go: joystick y at: angle.
   location ← location + (speed/10).
"---------------"
  self show.]
_J
SystemOrganization classify: ↗SpaceShip under: 'Players and Boxes'._J
```

```
'From Smalltalk 5.20 on 4 May 1978 at 6:40:54 pm.'_!

                                                    "JoyStick"
Class new title: 'JoyStick'
  subclassof: Window
  fields: 'point scale title'
  declare: 'instancecount ';
  asFollows_!

This class has not yet been commented

As yet unclassified
botrect
  [↑ (frame origin x ⊙ (frame origin y+ (2 * frame extent y)/3)) rect:
       frame corner

  ]
eachtime
  [frame has: user mp⇒
    [user kbck⇒[↑self kbd]
    user anybug⇒
      [user redbug⇒[↑self redbug]
        [user anybug ⇒ [] ↑ self nobug].
      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"]
init
  [instancecount ← [instancecount≡nil ⇒ [1] instancecount+1].
   self title: 'joystick:'+ (instancecount asString).
   self newframe; show.
   scale ← 10.
   point ← 0 ⊙ 0.]
nobug
  [point ← 0 ⊙ 0.]
redbug |
  [point ← (user mp - frame center)/scale]

scale: scale
title [↑title]
title: title
toprect
  [↑ (frame origin) rect:
       (frame corner x ⊙
            (frame origin y +(frame extent y)/3)
            )

  ]
x
[user anybug and: (frame has: user mp) ⇒
  [user mp x < (frame origin x + (frame extent x/3))
       ⇒ [↑ ¯5]
   user mp x > (frame corner x - (frame extent x/3))
       ⇒ [↑ 5] ↑0]
     ↑ 0
```

```
  ]
xy [↑point]
y
[user redbug and: (frame has: user mp) ⇒
  [user mp y< (frame origin y + (frame extent y/3))
     ⇒[↑5]
   user mp y> (frame corner y - (frame extent y/3))
     ⇒[↑ 5] ↑0]

    ↑0
  ]
SystemOrganization classify: ↗JoyStick under: 'Players and Boxes'.
```

"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

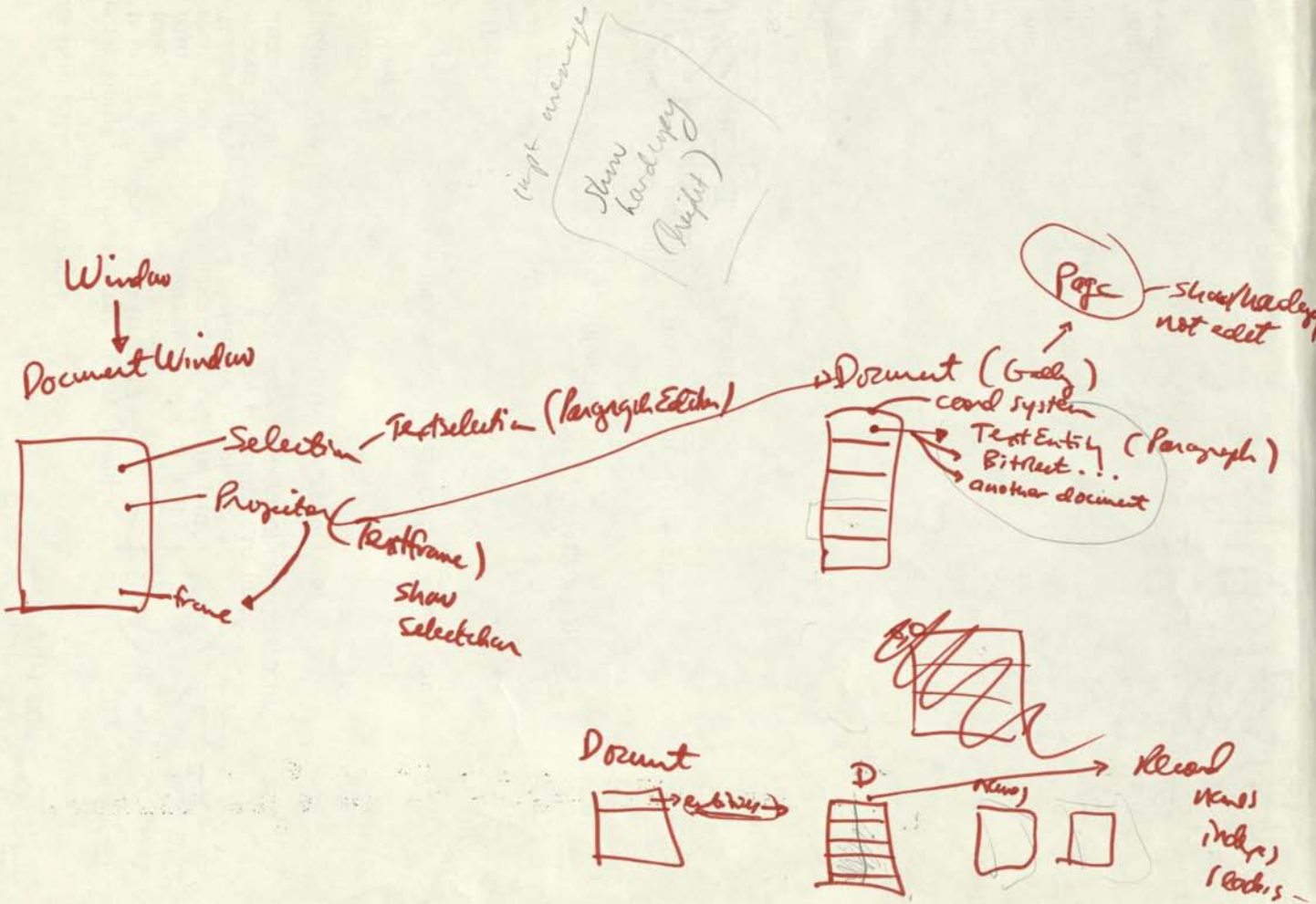
```
classInit
   [WidthDict ← Dictionary init]
lookup | key font file i
   [key ← name + pointsize asString + (↱('' 'I' 'B' 'BI')∘(face+1)).
   font ← WidthDict lookup: key⇒ [ℕfont]
   file ← dpO file: 'fonts.widths'.
   self fontfrom: file readonly.
   for: i from: ↱(011 015 040) do:
      [i between: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

Access

```
ascent [ℕascent]
descent [ℕdescent]
face [ℕface]
max [ℕmax]
min [ℕmin]
name [ℕname]
pointsize [ℕpointsize]
space [ℕ150]
tab [ℕ500]
widthof: char
   [char < min ⇒
      [char = 015 ⇒ [ℕ0]
      char = 040 ⇒ [ℕ0]
      user notify: 'char too low']
   char > max ⇒ [user notify: 'char too high']
   ℕwidths ∘ (char + 1 - min)]
```

Reading FONTS.WIDTHS

```
findfield: n on: file | IXH type len
```


Window
Document Window
Selection
Text selection (Paragraph Editor)
Projector (Textframe)
show selection
frame
Document (Graphic)
coord system
Text Entity (Paragraph)
Bitmap
another document
Page
show hardcopy
not edit
Document
D
Record
Keys
index
locks

E A R S

Filename: panes-and-menus.st

Creation Date: May 5, 1978 4:28 PM

Printed by: Weyer, Steve

'From Smalltalk 5.3c on 4 May 1978 at 3:50:19 pm.'_!

"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: pane
    [selectorPane ← pane asCitation]
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→[ñself kbd]
        user anybug→
            [user redbug→[ñself redbug]
            user yellowbug→[ñself yellowbug]
            user bluebug→[ñfalse]]
        user anykeys→[ñself keyset]]
    ñself 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 [pared hardcopy: p]
kbd
    [pared typing]
```

```
keyset
    [↑pared keyset]
leave
    [scrollBar hide]
outline
    [frame outline: 1]
outside
    [↑scrollBar startup]
picked
    [↑frame has: user m→]
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 value execute: pared selectionAsStream for: self)
asString asParagraph];
        =6→[pared formerly→
                [scrollBar hideWhile$ [selectorPane value 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.
    ifFalse]
oldContents
    [ifpared formerly]
reflects: selection  "am I trying to show the code of selectorPane's
selection?"
    [ifclass=nil and: selection>0]
```

Reclamation

```
keepCitations
    [selectorPane keep]
_]
SystemOrganization classify: ↑CodePane under: 'Panes and Menus'._]
CodePane classInit_]
```

```
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→[ñself kbd]
      user anybug→
         [user redbug→[ñself redbug]
         user yellowbug→[ñself yellowbug]
         user bluebug→[ñfalse]]
      user anykeys→[ñself keyset]]
   ñself outside]
enter
   [scrollBar show]
firsttime
   [window has: user mp→[self enter]
   ñfalse]
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 ← Cursor new frompage1.
    self scrollControl$ [user keyset=6⊃[2]; =12⊃[¯2]; =2⊃[1]; =8⊃[¯1] 0].
    c topage1]
lasttime
    [self leave]
leave
    [scrollBar hide]
outline
    [window outline: 1]
outside [ñscrollBar startup]
picked
    [ñwindow 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 linheight + firstShown.
    Xeq Cursor showwhile$ [self select: [newSel = selection⊃ [0] newSel]]]
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"
    [ñfalse]
locked "My subclasses may want to prohibit a change of selection"
    [ñ[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
    [ñ'————————————————————————————————']
fill | dY i len s "Given firstShown, compute lastShown and show me."
    {
    dY ← self linheight. len ← list length.
    lastShown ← firstShown-1 + (window extent y-4/dY) min: len+1.
    [self locked⊃
        [i ← (selection-lastShown max: 0) + (selection-firstShown min: 0).
        i≠0⊃ [para←nil. firstShown ← firstShown + i. lastShown ← lastShown
+ i]]].
    {frame ← window inset: 2) width ← 999.
    [para=nil⊃ "If para is not nil, refresh from it, else compute para."
        [s ← (String new: 200) asStream.
        for$ i from: (firstShown to: lastShown) do$
            [[0≤i and: i≤len⊃ [(list@i) printon: s] self dummy copyto: s].
            s cr].
```

```
        para ← s contents
        ]].
    self show]
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 bit: pt mode: storing. self show.
        (t← expr eval) abs $1→ [for$ delay to: chars/4 do$ [strm myend]. para
← nil. ñfalse]
            t*n<0→[ñfalse]].
    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 iv
    ["I have a selection. Return its highlighting rectangle."
    (iv ← window inset: 2) height ← h ← self lineheight.
    ñiv + (0@(selection-firstShown *h))]

Reclamation
keepCitations
    [scrollBar keepCitations]
```

```
As yet unclassified
scrollPos [↑0.0]
scrollTo: ignored
_!
SystemOrganization classify: ↑ListPane under: 'Panes and Menus'._!
```

"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: pane to: organizationPane
   [systemPane ← pane asCitation]
```

Window protocol

```
close
   [systemPane ← nil. super close]
keepCitations
   [systemPane keep]
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 value 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 value compile: parag]
dirty
   [↑organizationPane dirty]
noCode
   [selection=0⊃ [↑systemPane value noCode] ↑'']
┘
SystemOrganization classify: ⍎ClassPane under: 'Panes and Menus'._┘
ClassPane classInit_┘
```

"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 transformg each tog 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"
    ↑"buocursor showwhileg" self bug]
show
```

```
    [frame clear: black. text show.]

Internal
bugit | pt bits
    [user nobug ⊃
        [↑0]                                    "accidental bug returns 0"
    thisline comp.
    whileℓ true doℓ
        [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"
        untilℓ [text frame has: user mp] doℓ
            [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 bitsht>String.                 "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: pane to: selectorPane
    [classPane ← pane asCitation]
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."
    selections1⊃ [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: [selections2⊃ (Vector new: 0] class organization
category: list∘selection]]
```

Browser protocol

```
code: selector
    [↑class code: selector]
compile: parag
        | sel cat
    [class=nil ors selection=1⊃ [classPane value 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 ← listoselection.
    self revise: (self listFor: class) with: cat.
    selection>0→
        [selectorPane revise: (class organization category: cat) with: selector]]
noCode
    [class=nil→ [↑class?ane value 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]
```

Reclamation

```
keepCitations
    [classPane keep]
_]
SystemOrganization classify: ↑OrganizationPane under: 'Panes and
Menus'._]
OrganizationPane classInit_]
```

```
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, this is*

undefined. 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
```

"

```
    hair1 origin y = hair2 origin y→
        [self complementScreenRect: (hair1 origin rect: hair2 corner)]
    screenRect ← (screenFrame origin x ⊙ hair1 corner y) rect: (screenFrame
corner x ⊙ hair2 origin y).
    self complementScreenRect: (hair1 origin rect: (screenRect corner x ⊙
screenRect origin y)).
    self complementScreenRect: screenRect.
    self complementScreenRect: ((screenRect origin x ⊙ screenRect corner y)
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.
    ↑selOrigin rect: 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."

```
    ↑selOrigin rect: 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))]
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 groiVto: (self screenPtOf: aperture corner)]
moveInto: screenRect | eraseRects screenDXY keepRect
["Change my screenAperture to screenRect, thereby changang its position
and/or size. Adjust my state accordingly. show my neiv 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
iVhite, would erase my old image without affecting my neiv image. Erase
them and return the vector so the caller iVill 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 reshoiVBiting: 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 groiVto: (self ptOf: screenRect corner)]
scrollBy: dXY
["Move aperture by dXY, and reshoiV. 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 reshoiV. 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 reshoiVBiting: keepRect by: screenDXY erasing: eraseRects]
shiftToProject: apertureOrigin onto: screenRect showingIf: doShow | rect
screenDXY pt0]
["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) reshoiV 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
iVill erase my old image iVithout affecting my neiV 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⊃ [ñself 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."*
ñdX/reduction asPtX]

screenDXYOf: dXY
[*"Return the screen dXY proportional to dXY."*
ñdXY/reduction]

screenDYOf: dY
[*"Return the screen dY proportional to dY."*
ñdY/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."*
ñ(self screenXOf: pt x) @ (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."*
ñ(self screenPtOf: rect origin) rect: (self screenPtOf: rect corner)]

screenXOf: x
[*"Return the screen X onto which I project x."*
ñx/reduction asPtX + translation x]

screenYOf: y
[*"Return the screen Y onto which I project y."*
ñy/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.
ñProjector 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."*
ñscreenAperture intersect: (self screenRectOf: rect)]

Reverse Projection

dXOf: screenDX
[*"Return the slide dX proportional to screenDX."*
ñscreenDX * reduction asPtX]

dXYOf: screenDXY
["Return the slide dXY proportional to screenDXY."
ñscreenDXY * reduction]
dYOf: screenDY
["Return the slide dY proportional to screenDY."
ñscreenDY * 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."
ñ(self xOf: screenPt x) @ (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."
ñ(self ptOf: screenRect origin) rect: (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.
ñProjector 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."
ñaperture intersect: (self rectOf: screenRect)]
xOf: screenX
["Return the slide X which I project onto screenX."
ñscreenX-translation x * reduction asPtX]
yOf: screenY
["Return the slide Y which I project onto screenY."
ñscreenY-translation y * reduction asPtY]

Access To Parts

aperture [ñaperture]
reduction [ñreduction]
screenAperture [ñscreenAperture]
slide [ñslide]
symbolism [ñsymbolism]

Utilities

check
["Notify user if I have inconsistent internal state."
aperture ≠ (self rectOf: screenAperture)⊃ [user notify: 'sliderect ≠ screenrect']
translation ≠ (screenAperture origin-- aperture origin)⊃ [user notify: 'translation wrong']
(0 @ 0) > (screenAperture extent)⊃ [user notify: 'negative amt visible']]
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)]
reshowBitting: screenRect by: screenDXY erasing: eraseRects | eraseRect

```
["Use Bit 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 Bit and before the regeneration; this will
accomplish an unshow of an old image."
    screenRect bit: (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: 'Panes and Menus'.⌟
```

"ScrollBar"

```
Class new title: 'ScrollBar'
   subclassof: Object
   fields: 'rect bitstr owner position'
   declare: 'JumpCursor DownCursor UpCursor ';
   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
1100000000000000'.
   DownCursor ← Cursor new fromtext: '
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1111110000000000
1111100000000000
1111000000000000
1110000000000000
1100000000000000
1000000000000000'.
   JumpCursor ← Cursor new fromtext: '
0111000000000000
1111100000000000
1111100000000000
0111000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
```

```
ooooooooooooooo
ooooooooooooooo
ooooooooooooooo
ooooooooooooooo
ooooooooooooooo
ooooooooooooooo']
on: f from: o
    [self on: f from: o at: o scrollPos]
on: frame from: o at: f
    [owner ← o asCitation.
    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 value scrollTo: (position minY-rect
minY-4) asFloat/(rect height-12)]
                    user redbug⊃[self reposition:[owner value scrollUp: p y+16 -
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 value scrollTo: (position minY-rect minY-
4) asFloat/(rect height-12)]
                user redbug⊃[self reposition:[owner value 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.
                    while: user redbug do:
                        [self reshow:
                            [position moveto: position origin x@
                                ((user mp y max: rect origin y+4) min: rect corner y-
12)]]]]].
        ↑bug]
    ↑false]
```

Image

```
hide        "restore background"
    [bitstr=nil→ [user'notify: 'Attempt to hide unshown scrollbar']
    rect bitsFromString: bitstr]
hideWhile: expr | v
    [self hide. v ← expr eval. self show. ↑v]
reposition: expr
    [self reshow:
        [expr eval.
        position moveto: rect origin+
            (9⊙(4+(owner value scrollPos*(rect height-16))))]]
reshow: expr | r
    [r ← position inset: ¯2. expr eval.
    r clear: white. position outline]
show        "Save background and turn gray"
    [bitstr ← rect bitsIntoString.
    rect clear: black.
    (rect inset: 2⊙2 and: 1⊙2) clear: white.
    position outline]
```

Reclamation

```
keepCitations
    [owner keep]
↵
SystemOrganization classify: ↑ScrollBar under: '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: pane to: codePane
    [organizationPane ← pane asCitation]

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

ListPane protocol
deselected
    [codePane showing: organizationPane value noCode]
selected
    [codePane showing: (organizationPane value code: list∘selection)]

Browser protocol
compile: parag
    [↑organizationPane value 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]

Reclamation
keepCitations
    [organizationPane keep]
_]
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
   [(contextVarPane 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
    [ñcodePane and§ codePane dirty]
execute: parseStream for: codePane
    [ñproceed∘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.
    ñ[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: dummu1 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 ← listo(selection max: 1). ↑[context sender=nil→ [false] context
sender thisop]]
separator: c          "called by selected via Class fieldNamesInto"
spawn | mclass selector parag oldparag
    [mclass ← (listo(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"
```

Reclamation

```
keepCitations
    [list is: Vector→ [(listo1) keepCitations]]
_]
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]
leave      "I am up to date"
   [mySysOrgVersion ← user classNames. super leave]
yellowbug
   [selection<3→[window flash]
   scrollBar hideWhiles
      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→ [ñuser classNames];
             ≤2→ [ñVector new: 0]
   ñSystemOrganization 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 is: 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: 'All '+className+'s will become obsolete if you proceed...'.
   (Smalltalk@className) obsolete. Smalltalk delete: className.
   SystemOrganization delete: className.
   AllClassNames ← AllClassNames delete: className.
   classPane revise: self classes with: className]
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

```
keepCitations
    [values keepCitations]
_]
SystemOrganization classify: ↗VariablePane under: 'Panes and Menus'._]
VariablePane classInit_]
```

E A R S

Filename: documents.st

Creation Date: April 25, 1978 2:22 PM

Printed by: Weyer, Steve

'From Smalltalk 5.3b/xm on 24 April 1978 at 9:56:27 am.'↵

"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

```
∘ 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'
    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 | proj y frame i
    [super init.
    bottoms ← Vector new: entities length.
    y ← top ← 0.
    frame ← 0⊙y rect: width⊙y.
    proj ← Projector new projecting: y asPoint of: nil onto: 0 asRectangle.
    forₛ i to: entities length doₛ
        [bottoms∘i← y← y + (entities∘i growthThru: proj within: frame).
        frame origin y ← y.
        frame corner y ← y]]
with: entities width: width bottoms: bottoms top: top
    [super init]
```

File Conversion

```
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"
    untilₛ strm end doₛ
        [e next ← TextEntity new readBravo: strm styles: s]
    self with: e contents width: width]
writeBravo: f | e
    [forₛ e from: entities doₛ
        [e writeBravo: f]]
```

Aspects

```
bottoms
    [ñbottoms]
frame
    [ñself minX⊙self minY rect: self maxX⊙self maxY]
frameOf: index
    [ñ 0 ⊙ [index=1→ [top] bottoms∘(index-1)]
        rect: width ⊙ (bottoms∘index)]
framesShifted: dXY  | t b l dY y
    [l ← dXY x. dY ← dXY y. b ← top+dY.
    ñbottoms transformₛ y toₛ [t ← b. b ← y+dY.  (l⊙t rect: l+width⊙b)]]
height [ñself maxY-self minY]
maxX
    [ñwidth]
maxY
    [ñ[bottoms length=0→ [top] bottoms last]]
minX
    [ñ0]
minY
    [ñtop]
```

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]
showThru: proj | index i1 i2 fr
   [fr ← proj aperture.
   i1 ← bottoms findSorted: fr origin y.
   i2 ← (bottoms findSorted: fr corner y) min: bottoms length.
   for$ index from: (i1 to: i2) do$
      [entities∘index showThru: proj within: (self frameOf: index)]]
```

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)]
⌟
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]
hardcopy: p | i
    [p setrect: pattern.
    for: i to: entities length do:
        [entitiesoi presson: p in: framesoi].
    p page]
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'.↵
```

"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: '';
   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

```
thru: proj
   [doc ← proj slide.
   entity ← oldEntity ← index ← false]
```

Defaults

```
complement
   [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]
leave [self sendchanges complement]
redbug | pt previousEntity ind
   [self complement.
   previousEntity ← entity.
   pt ← proj ptOf: user mp.
   (ind ← doc entityIndexAt: pt) and:
   (entity ← doc@ind)=previousEntity⊃
      [↑self of: entity formerly: oldEntity at: ind from: pt thru: proj]
   [previousEntity⊃ [previousEntity deselected]].
   self sendchanges "if any".
   (index ← ind)⊃
```

```
        [entity selected.
        ↑entity 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'.┘
```

"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

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 ← Style default]
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]
┘
```

```
SystemOrganization classify: ↑TextEntity under: 'Documents'.↵
```

"TextSelection"

```
Class new title: 'TextSelection'
   subclassof: Selection
   fields: 'c1 c2 "Integer character positions" typein'
   declare: 'paste Scrap ctlw bs Deletion ctlchars esc cut
editmenu runvals ';
   asFollows_]
```

This is the text editor for a currently selected TextEntity.

Initialization

```
classInit | i
   [bs ← 8. ctlw ← 23. esc ← 27.
   cut ← 127. paste ← 2.
   ctlchars ← ↗(0342 0351 0255 0256 0362
      0260 0261 0262 0263 0264 0265 0266 0267 0270 0271
      0341 0342 0343 0344 0345 0346).
   runvals ← ↗(1 2 4 8 ¯1  "b i - ¯ r"
      0 1 2 3 4 5 6 7 8 9  "0 1 ... 9"
      10 11 12 13 14 15).  "a b ... f"
   for: i to: 16 do: [runvals∘(5+i) ← runvals∘(5+i)*16]
   editmenu ← Menu new string:
'undo
copy
cut
paste
doit
align'.]
of: entity formerly: oldEntity at: index from: pt thru: proj | h1 h2 cNew
hOld
      "init and draw out selection"
   [typein ← false.
   doc ← proj slide. doc typeset: index in: proj.
   c1 ← c2 ← proj charNearPt: pt.
   h1 ← proj screenHairBeforeThatChar. h2 ← h1 + (1⊙0).
   proj complementScreenHairs: h1 to: h2.
   while: user anybug do:
      [cNew ← c1 max: (proj charNearScreenPt: user mp).
      c2 = cNew⊃ [ ]
      hOld ← h2.
      h2 ← [c1 = cNew⊃ [h1 + (1⊙0)] proj screenHairBeforeThatChar].
      [cNew > c2⊃ [proj complementScreenHairs: hOld to: h2]
         proj complementScreenHairs: h2 to: hOld].
      c2 ← cNew]]
```

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]
show
```

```
    [doc showEdited: index in: proj]
```

Editing

```
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 ← Stream default.
    [typein⊃[] typein ← c1].
    while$ user kbck do$ [
        (char ← user kbd)
        =bs⊃ ["backspace"
            more empty⊃ [typein ← typein min: (c1 ← 1 max: c1-1)]
            more skip: ¯1];
        =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];
        =cut⊃ [⍝self cut];
        =paste⊃ [⍝self paste];
        =esc⊃ ["select previous type-in"
            [more empty⊃[] self replace: more contents. c1 ← c2].
            self fintype.
            c1 ← c2-Scrap length.
            ⍝self complement]
        "just a normal character"
    more next← char].
    self replace: more contents.
    c1 ← c2.
    user rawkbck⊃[]
    self complement]
paste
    [self replace: Scrap; complement]
replace: t
    [   [oldEntity⊃[] oldEntity ← entity copy].
        [c1<c2⊃[Deletion ← entity copy: c1 to: c2-1]].
    entity ← entity replace: c1 to: c2-1 by: t.
    c2 ← c1 + t length.
    self show]
selection [⍝entity copy: c1 to: c2-1]
yellowbug
    [entity ≡ false⊃ [ ]
    editmenu bug
        =1⊃[self fintype; replace: Deletion; complement "undo"];
        =2⊃[Scrap ← self selection "copy"];
        =3⊃[self cut];
        =4⊃[self paste];
        =5⊃[Scrap ← XeqCursor showwhile$ "doit"
                (nil'sself selection) asString asParagraph];
        =6⊃ [entity alignment ← ⍉(1 2 4 0 0)∘(1+entity alignment).
                self show]]
```

Private Messages

```
checklooks | t val mask
    [(t ← user rawkbck)<0241⊃[⍝false];
```

```
        =0324 "ctl-shift-T" ⊃[user rawkbd. self toBravo];
        =0306 "ctl-shift-F" ⊃[user rawkbd. self fromBravo].
    c1≥c2 ⊃ [ñfalse]
    (t ← ctlchars find: t)=0⊃[ñfalse]
    [oldEntity⊃ [] oldEntity ← entity recopy].
    user rawkbd.
    [(val← runvalsot)=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]
complement
    [proj complementChars: c1 to: c2]
fintype
    [typein⊃
        [   [typein<c1⊃
                [Scrap ← entity copy: typein to: c1-1.
                c1 ← typein]].
        typein ← false]
    ñfalse]⌟
⌟
SystemOrganization classify: ↑TextSelection under: 'Documents'.⌟
TextSelection classInit⌟
```

E A R S

Filename: windows.st

Creation Date: April 25, 1978 2:26 PM

Printed by: Weyer, Steve

'From Smalltalk 5.3b/xm on 24 April 1978 at 10:00:07 am.'_!

"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 fontsofontnumber.
    ]
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. ñfalse]]
        user kbck⊃
            [char ← user kbd. self setchar: char]
        ]
    user anybug⊃[ñfalse]
    ]
firsttime                       "upon entry"
    [
    clearframe has: user mp⊃[self show]
    ñfalse
    ]
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 lock) + 9.                             "lock font and get core ptr"
    bitsetter fill: storing color: color.                            "turn bit on/off in font"
    font unlock.                                                  "release font"
    ]
setchar: char
    [
    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 ← (fontº(1 to: fontxtabl - 1 * 2) concat: space).
            updatedfont ← (self appendxtable: updatedfont).
            ]
    updatedfont ←
        (fontº(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: charstro1.
    ]
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 lock) + 9.
```

```
    bitsetter sourcebase ← (font lock) + 9.
    bitsetter copy: storing.
        [                    "if char grown, clean out right side of char"
        delta< 0 ⊃[]
        bitsetter destx ← charx + charwid.
        bitsetter width ← delta.
        bitsetter fill: storing color: 0.
        ].
            "now copy remainder of font"
    bitsetter destx ← charx + charwid + delta.
    bitsetter width ← fontrightx - charx - charwid.
    bitsetter sourcex ← charx + charwid.
    bitsetter copy: storing.
    updatedfont unlock. font unlock.
        "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'sxtable on end of a grown/shrunk font"
    thefont ← thefont concat: font ∘ ((fontxtabl * 2 - 1) to: font length).
    ñthefont.
    ]
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 ← (dpO file: name + '.cu.').
    self updateseglength: font raster: fontraster.  self updatemaxwidth.
    svscale ← scale. scale ← cuscale.  svchar ← char.
    self cufixup.
    f nextword ← fontht*scale.
    f nextword ← (b.twidth ← (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"

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

        ascent ← ascent min: (fontht-1).                                "keep baseline
within char"
        (fontO(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⌟
```

```
Class new title: 'Window'
   subclassof: Object
   fields: 'frame collapsed titlepara growing exitflag '
   declare: 'border titleframe windowmenu titlerun titleloc ';
   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]
fixframe: f [↑f]
frame: f
   [frame ← self fixframe: f]
moveFrom: oldframe
   [(oldframe inset: ¯2) clear. self show]
```

```
newframe | a oldframe
    [user waitnobug.
    [frame≠nil⊃[] self aboutToframe. self erase].
    a ← OriginCursor showwhile: user waitbug.
    growing ← true.
    self frame: (frame ← self fixframe: (a rect: a+16)). self show.
    CornerCursor showwhile:
        [while: user anybug do:
            [oldframe ← frame copy.
            self frame: (frame ← self fixframe: (frame growto: user mp+16)).
            self moveFrom: oldframe]].
    growing ← false.
    user cursorloc ← frame center]
outline
    ["Clear and outline me."
    frame outline]
refresh
    [self show]
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 [ñ'unoccupied']
```

Default Event responses

```
aboutToframe
    ["My frame is about to change.  I dont care."]
bluebug
    |windowmenu bug
        =1⊃[ñexitflag ← false];
        =2⊃|self newframe. self enter];
        =3⊃[self close. self erase.
            user unschedule: self. ñfalse];
        =4⊃[self hardcopy];
        =5⊃[self print]]
close []
enter [self show]
hardcopy [frame flash]
kbd [user kbd. frame flash]
keyset [frame flash]
leave []
outside [ñfalse]
print
    [(dpO pressfile: (self title + '.press.') asFileName)
        screenout: frame scale: PressScale]
redbug
    [frame flash]
yellowbug
    [frame flash]
⌟
SystemOrganization classify: ↱Window under: 'Windows'.⌟
```

Window 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]
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 | v
    [self hideBars. v ← expr eval. self showBars. ↑v]
showBars
    [scrollBar show]
title [↑title]
```

Reclamation

```
keepCitations
    [scrollBar keepCitations]
⌟
SystemOrganization classify: ↗DocumentWindow under: 'Windows'.⌟
```

"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

```
hardcopy | p
   [user displayoffwhile:
      [p ← dpO pressfile: title+'.press'. p scale: 32.
      document hardcopy: p through: projector.
      p close].
   user quitThen: 'EMPRESS ' + title + '.press; RESUME SMALL.SV
'.
   dpO delete: title.]
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'.↵
```

"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

```
hardcopy | p
    [p ← dpO pressfile: title.
    p scale: 32.
    document hardcopy: p.
    p close.
    user quitThen: 'EMPRESS ' + title + '; RESUME SMALL.SV
'.
    dpO delete: title]
⌟
SystemOrganization classify: ↑PageWindow under: 'Windows'.⌟
```

```
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.
   fors pane from: panes do: [pane init]]
```

Window protocol

```
close | pane
   [fors pane from: panes do: [pane close].
   CitationReasons ← CitationReasons-1]
eachtime | pane
   [frame has: user mp⊃
      [user bluebug⊃[Ωself bluebug]
      fors pane from: panes do: [pane startup]]
   self outside⊃[]
   user anybug⊃[frame has: user mp⊃[] Ωfalse]
   user kbck⊃[user kbd. frame flash] "flush typing outside"]
enter | pane
   [super show.
   fors pane from: panes do: [pane windowenter]]
erase
   [self titlerect clear. super erase]
fixframe: f
   [ΩRectangle 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.
   fors 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 t
   [user displayoffwhile:
      [p ← dp0 pressfile: (t← (self title + '.press.') asFileName).
      self hardcopy: p.
      p close.
      user quitThen:
'Empress ' + t + '
Resume small.boot.
']]
hardcopy: p | pane w
   [self showtitle.
   fors w from: (titleframe window, titleframe frame) do:
      [w width← w width*11/10].
   titleframe hardcopy: p.
   fors pane from: panes do: [pane hardcopy: p]]
kbd | pane
```

```
    [(pane ← self pickedpane)⊃ [ñpane kbd]]
keyset | pane
    [(pane ← self pickedpane)⊃ [ñpane keyset]]
leave | pane
    [for: pane from: panes do: [pane windowleave]]
outline
    [frame outline: 1]
pickedpane | pane
    [for: pane from: panes do: [pane picked⊃ [ñpane]]
    frame flash. ñfalse]
redbug | pane
    [(pane ← self pickedpane)⊃ [ñpane redbug]]
show | pane
    [super show.
    for: pane from: panes do: [pane outline]]
takeCursor
    [(panes∘1) takeCursor]
title
    [ñtitle]
yellowbug | pane
    [(pane ← self pickedpane)⊃ [ñpane yellowbug]]
```

Pane services

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

Private

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

Reclamation

```
isCitationReason
keepCitations | pane
    [for: pane from: panes do: [pane keepCitations]]
⌟
SystemOrganization classify: ↱PanedWindow under: 'Windows'.⌟
```

"BrowseWindow"

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

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]
_J
SystemOrganization classify: ↗BrowseWindow under: 'Windows'._J
BrowseWindow classInit_J
```

"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]
```

As stream

```
append: text
    [(panes∘1) append: text]
cr
    [(panes∘1) append: 015 inString]
display
    [(panes∘1) display]
next ← char
    [(panes∘1) append: char inString]
print: x
    [(panes∘1) append: x asString]
space
    [(panes∘1) append: ' ']
_]
SystemOrganization classify: ↑CodeWindow under: 'Windows'._]
CodeWindow classInit_]
```

"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: 12@36), (12@0 rect: 36@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: 'smallTemplates bigTemplates smallFrame ';
   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: (Topolevel) 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: instanceVauePane. instanceValuePane from:
instanceVarPane.
    stackPane select: 0; deselected; fill. enoughpanes ← NotifyFlag ← true]
┘
SystemOrganization classify: ↑NotifyWindow under: 'Windows'.┘
NotifyWindow classInit┘
```

"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_┘
```

Memo from: KARLA GARCIA

Simulation —
release

Play sound
Simulate with
restart

XEROX

Simulation View

Mail

Simulations

AddCopier
CopyFlow
FourCopiers
junk
Landing
LoadingDock
Printing
Simulation
Versatec

Copying

Binding

Commands

restart
refresh
nolog
report
done
printBW
printC

Garbage

92

Category	Component	Interface
Simulation Station Worker Job	new AddCopier CopyFlow	layout arrivalSchedule reportSchedule

arrivalSchedule
"State the job, input schedule, and initial assignments (the names of stations the job will visit). Here, piles of papers go to Copying every 4 minutes"

[self use: Gossip new
startAt: 25
schedule:
(Inputschedule new
constant: 30)

cut
paste
copy
undo
cancel
doit
rectangle
accept

Performance Measures

Papers 13 Duration 17.19
Visited: Copying Garbage
Feature: (26 5)
Papers 14 Duration 15.75
Visited: Copying Garbage
Feature: (13 2)
Gossip 2 Duration 19.75
Visited: Copying
Papers 15 Duration 23.54
Visited: Copying Garbage
Feature: (16 4)
Papers 16 Duration 23.92
Visited: Copying Garbage
Feature: (17 5)
Papers 17 Duration 24.58
Visited: Copying Garbage
Feature: (20 4)

cut
paste
copy
undo
doit
print

E A R S

Filename: simulator.st

Creation Date: April 25, 1978 2:32 PM

Printed by: Weyer, Steve

```
'From Smalltalk 5.3b/xm on 24 April 1978 at 9:37:09 am.'_!

                                                        "CustQueue"
Class new title: 'CustQueue'
   subclassof: Queue
   fields: 'maxlength              "for reporting maximum queue length"
            showangle            "direction for displaying queue"
            location           "of first job in line"
             wrapper            '
   declare: '';
   asFollows_!
```

Job queue--adds statistics reporting and screen displaying.

```
Initialization
init |
   [super of: (Vector new: 4).
    maxlength←0.
    showangle ← (0⊙0).
    wrapper←0.]
showangle: showangle|
   []
showat: location

Show
hide |      "removes image of all jobs, beginning at the end"
   [self remove: 1]
nextPlace: job in: rect| i              j  pt "next screen position to place queue object"
   [super empty ⊃ [⇑ location]
    i ← array ∘ position.
    j ← i extent max: job extent.
    pt← (i where + (showangle*(j + (4⊙4)))).
    (rect has: pt) and$ (rect has: pt+job extent) ⊃ [⇑pt] ⇑ self wraparound:
wrapper+1]
remove: j | i       "remove images of jobs from end to j"
   [for$ i from: (position to: j by: ¯1) do$
      [(array ∘ i) hide]. ]
show: location in: rect| at  i  j
   "displays jobs in direction of showangle--tests for different widths"
   [at ← (location x ⊙ location y).
    wrapper←0.
    for$ i from: (readposition+1 to: position) do$
      [[(rect has: at)
         and$ (rect has: at+ (array∘i) extent) ⊃ []
        at ← self wraparound: wrapper+1].
       (array∘i) moveto: at.
       j ← [(i+1) ≤ position ⊃
                [(array∘i) extent max: (array∘(i+1)) extent]
                (array∘i) extent].
       at ← at + (showangle*(j+ (4⊙4)))].                ]
showin: rect |
   [self show: location in: rect]
wraparound: wrapper
   ["position next to first object in line"
    showangle x = 0 ⊃
      [⇑ (location + (((self maxwidth + 4)*wrapper) ⊙0))]
```

```
    ↑ (location + (0⊙((self maxheight + 4)*wrapper)))]
```

Queue protocol

```
enQ1: n |
    [Top critical: [super next ← n]. self setmaxlength.↑n]
push: x |
    [super push: x.
     self setmaxlength]
setmaxlength |
    [(position-readposition) > maxlength ⊃
          [maxlength ← (position-readposition)]]
```

Stream protocol

```
next ← x |
    [super next ← x.
     self setmaxlength.]
```

Query

```
maxheight| i j
    [j ← 0.
     for: i from: (readposition+1 to: position) do:
        [j< (array@i) height ⊃
            [j←(array@i) height]].
     ↑j]
maxlength|
    [↑maxlength]
maxwidth| i j
    [j ← 0.
     for: i from: (readposition+1 to: position) do:
        [j< (array@i) width ⊃
            [j←(array@i) width]].
     ↑j]
showangle|
    [↑showangle]
```

Report

```
printon: strm |
    [strm append: 'Maximum queue length '; print: maxlength.]
⌟
SystemOrganization classify: ↗CustQueue under: 'Simulator'.⌟
```

"Door"

```
Class new title: 'Door'
   subclassof: Object
   fields: 'jobsEntered jobsThrough totalTimeSpent location
maxtime mintime'
   declare: '';
   asFollows_┘
```

todos done

Keeps statistics for a ~~*Station*~~ Place or Environment

```
Initialization
at: location |
   [ ]
init |
   [jobsThrough ← 0. jobsEntered ← 0. totalTimeSpent ← 0.0. maxtime←0.0.
mintime←100000.0.]

Scheduling
enter |
   [jobsEntered ← jobsEntered + 1]
leave |
   [jobsThrough ← jobsThrough + 1]
time: x |
   [maxtime ← maxtime max: x.
    mintime ← mintime min: x.
    totalTimeSpent ← totalTimeSpent + x]

Query
location |
   [ñlocation ]
numberOfCustomersThrough |
   [ñ jobsThrough ]

Report
reporton: strm | str
   [str ← '
No. of jobs in: ' + jobsEntered asString + '
No. of jobs out: ' + jobsThrough asString.
   [jobsThrough > 0 ⊃
      [str← str+ '
Min time spent: ' + (mintime roundTo: 0.01) asString + '
Avg total time spent: ' +
   ((totalTimeSpent/jobsThrough) roundTo: 0.01) asString + '
Max time spent: ' + (maxtime roundTo: 0.01) asString]].
   strm append: str asParagraph]
┘
SystemOrganization classify: ↱Door under: 'Simulator'._┘
```

- line '__'
- ~~initialize~~ add: 'N to: '__'
- report: '__'
- report: '__'
  as: 0/0

"Event"

```
Class new title: 'Event'
   subclassof: Object
   fields: 'target selector simulatedTime nextEvent'
   declare: '';
   asFollows_┘
```

An event for scheduling by a TimeScheduler

Initialization and Access

```
next "Return next Event to happen"
   [↑nextEvent]
next ← nextEvent "Change next Event to happen"
of: target doing: selector in: simulatedTime before: nextEvent "Initialize an
Event"
printon: strm "Print"
   [strm append: 'Event: ';
         print: target;
         append: ' will get ';
         print: selector;
         append: ' after ';
         print: simulatedTime]
release
   [nextEvent≠nil ⊃ []
   target release.
   target←nil.
   nextEvent release.
   nextEvent ← nil.]
selector [↑selector]
simulatedTime [↑simulatedTime]
simulatedTime: simulatedTime
target [↑target]
```

Behavior

```
> time "Test if <time> will come before I expire"
   [simulatedTime≠nil⊃[↑true]
    ↑simulatedTime>time]
end "Is this the dummy last Event"
   [↑simulatedTime≠nil]
insert: targ doing: sel after: time
   [simulatedTime≠nil or: simulatedTime>time⊃
      [ [simulatedTime≠nil⊃[] simulatedTime← simulatedTime-time].
       ↑Event new of: targ doing: sel in: time before: self]
   nextEvent←
      nextEvent insert: targ doing: sel after: time-simulatedTime]
subtract: time "Shorten time to run"
   [simulatedTime≠nil⊃[]
    simulatedTime ← simulatedTime - time]
┘
SystemOrganization classify: ↑Event under: 'Simulator'._┘
```

"Form"

```
Class new title: 'Form'
   subclassof: Object
   fields: 'data "<String> containing bits, or false if a screen form"
      raster "<Integer> words per scan line"
      origin '<Point> relative to data"
      extent "<Point> relative to origin"
      '

   declare: 'copying painting filling bltter copycomping zero ';
   asFollows_]
```

Holds a graphical form, as a bit pattern, either stored in a string or on the screen

Initialization

```
at: rect | org cor
   [org ← rect origin.  cor ← rect corner.
   org≥zero and: cor≥org⊃
      [origin ← org x⊙org y.  extent ← cor-org.
      data ← false]
   user notify: 'Bad rectangle']
classInit
   [bltter ← BitBlt init.
   zero ← 0⊙0.
   "Define BitBlt functions"
   copying ← 0.  copycomping ← 4.  painting ← 8.  filling ← 12.
   ]
data: d bitsPerLine: b | r ye
   [[d is: String⊃ []; is: Vector⊃ [d ← d copyto: (String new: d length)]
      user notify: 'Bad data'.  d ← ''].
   r ← b+15/16.
   ye ← [r≠0⊃ [d length/(r*2)] 0].
   d length≠(r*ye*2)⊃ [user notify: 'Bad bitsPerLine']
   data ← d.  raster ← r.
   origin ← zero.  extent ← b⊙ye]
hull        "only needed for converting to Travellers"
   [ñString new: data length]
"
 | each.
for: each from: PictureOrganization do:
   [PictureOrganization∘each←
      Traveller new assign: PictureOrganization∘each]
"
init
   [self size: zero]
size: e
   [raster ← e x+15/16.
   data ← String new: raster*2*e y.
   origin ← zero.
   extent ← e x⊙e y]
```

Strike stuff

```
strike: strike | nx ras ht
   [(strike isnt: String) or: strike∘1/64≠2⊃ [user notify: 'Bad strike']
   nx ← (strike word: 3)-(strike word: 2)+3.
   ras ← strike word: 9.  ht ← (strike word: 6)+(strike word: 7).
```

```
    strike length<((strike word: 5)+4*2) or: strike length<(ht*ras+nx+9*2)⊃
        [user notify: 'Bad strike']
    data ← strike∘(19 to: 18+(ras*ht*2)).  raster ← ras.
    origin ← zero.  extent ← ras*16 ⊙ ht]
string: s font: i
    [self string: s font: i style: DefaultTextStyle]
string: s font: i style: style
    [self string: s strike: (style strikeofset: i)]
string: s strike: strike | font min max ht xtab old w i c
    [font ← Form new strike: strike.
    min ← strike word: 2.  max ← strike word: 3.
    ht← (strike word: 6) + (strike word: 7).
    xtab ← (strike word: 9)*ht+10-min.
    old← Rectangle new origin: origin extent: extent.
    font bltsource.
    for: i to: s length do:
        [c ← ((s∘i max: min) min: max)+xtab.
        bitter sourcex← strike word: c.
        w ← (strike word: c+1)-(strike word: c).
        extent← w⊙ht.
        self bltdest: copying+oring.
        origin ← origin + (w⊙0)].
    origin← old origin. extent← old extent.
    ]
```

Copying

```
copy
    [∩(Form new size: extent) copy: self mode: storing]
replaceBy: form
    [data ← form data.  raster ← form raster.
    origin ← form origin.  extent ← form extent]
```

Aspects

```
∘ pt | i j mask
    [data class≠String⊃ [∩((Form new size: 1⊙1) copy: (self sub: (pt rect:
pt+1))) data∘1/0200]
    j ← origin x+pt x.
    i ← origin y+pt y*raster*2+(j/8)+1.
    mask ← ↗(0200 0100 040 020 010 4 2  )∘(j\8+1).
    ∩[(data∘i land: mask)=0⊃ [0] 1]]
asRectangle
    [∩Rectangle new origin: origin extent: extent]
asRectCorner [∩origin+extent]
asRectOrigin [∩origin x⊙origin y]
corner [∩origin+extent]
extent [∩extent x⊙extent y]
origin [∩origin x⊙origin y]
sub: rect
    [rect origin≥zero and: rect cornerseextent⊃
        [∩[data⊃ [Form new data: data raster: raster origin: origin+rect origin
extent: rect extent] Form new at: rect+origin]]
    user notify: 'Bad subrectangle']
```

Altering

```
∘ pt ← val | i j mask
```

```
    [data isnt: String⊃
        [(self sub: (pt rect: pt+1)) clear:
            [val=0⊃[white]; =1⊃[black] gray]]
    j ← origin x+pt x.
    i ← origin y+pt y*raster*2+(j/8)+1.
    mask ← ↗(0200 0100 040 020 010 4 2 1)∘(j\8+1).
    dataoi ← [val=1⊃[dataoi lor: mask] dataoi land: ¯1-mask]]
clear: color
    [bitter color ← color.
    self bitdest: filling+storing]
color: color mode: mode
    [bitter color ← color.
    self bitdest: filling+(mode land: 3)]
comp
    [bitter color ← black.
    self bitdest: filling+xoring]
comp: color
    [bitter color ← color.
    self bitdest: filling+xoring]
copy: form
    [(extent≥form extent)≡false⊃ [user notify: 'Sizes not compatible']
    form bitsource.
    self bitdest: copying+storing]
copy: form mode: mode
    [(extent≥form extent)≡false⊃ [user notify: 'Sizes not compatible']
    form bitsource.
    self bitdest: copying+(mode land: 3)]
copycomp: form mode: mode
    [(extent≥form extent)≡false⊃ [user notify: 'Sizes not compatible']
    form bitsource.
    self bitdest: copycomping+(mode land: 3)]
paint: form color: color mode: mode
    [(extent≥form extent)≡false⊃ [user notify: 'Sizes not compatible']
    form bitsource.
    bitter color ← color.
    self bitdest: painting+(mode land: 3)]
```

Filout

```
asVector
    [data class≠String or: (origin≠zero or: extent x+15/16<raster)⊃ [ñself
copy asVector]
    ñdata copyto: (Vector new: data length)]
fullprinton: strm
    [strm append: '(Form new data: ↗'; print: self asVector;
        append: ' bitsPerLine: '; print: extent x;
        append: ')']
```

Private

```
bitdest: function | wordsPerLine halfHeight base slock dlock tfunction
    [bitter destx ← origin x;
        desty ← origin y;
        width ← extent x;
        height ← extent y.
    tfunction ← function lor: ((bitter function) land: 040).
    [data⊃ [bitter destbase ← data; destraster ← raster; function ←
tfunction]
    "Check against hardware parameters"
```

```
    wordsPerLine ← mem∘065 land: 0377.
    halfHeight ← user screenrect extent y / 2.
    bitter destbase ← mem∘066; destraster ← wordsPerLine;
            function ← tfunction lor: 020.
    origin x+extent x>(wordsPerLine*16) or: origin y+extent
y>(halfHeight*2)⊃
        ["user notify: 'Bad rectangle'." ↑self]].
    "Following junk is because BitBlt doesn't have separate strike flags for
source and destination"
    [(base ← bitter sourcebase) is: String⊃ [bitter sourcebase ← (slock ← base)
lock];
        is: Substring⊃ [bitter sourcebase ← (slock ← base asStream asArray)
"Only way to get the data string" lock+9]].
    [(base ← bitter destbase) is: String⊃ [bitter destbase ← (dlock ← base)
lock];
        is: Substring⊃ [bitter destbase ← (dlock ← base asStream asArray)
lock+9]].
    bitter callBLT.
    [slock≡nil⊃ [] slock unlock].
    [dlock≡nil⊃ [] dlock unlock].
    ]
blitsource
    [bitter sourcex ← origin x;
        sourcey ← origin y;
        width ← extent x;
        height ← extent y.
    data⊃ [bitter sourcebase ← data; sourceraster ← raster; function ← 0]
    "Don't bother to check -- no harm can result"
    bitter sourcebase ← mem∘066; sourceraster ← mem∘065 land: 0377;
function ← 040.
    ]
data [↑data]
data: data raster: raster origin: origin extent: extent
data← data
raster [↑raster]
⌟
SystemOrganization classify: ↑Form under: 'Simulator'.⌟
Form classInit⌟
```

"Inputschedule"

```
Class new title: 'Inputschedule'
   subclassof: Object
   fields: 'constant orderedSet distribution'
   declare: '';
   asFollows_]
```

This class returns arrival times according to stored or computed distributions

Distribution Initialization

```
distribution: distribution
   [constant ← orderedSet ← false]
exponential: m "m > 0.0"
   [self distribution: (Exponential new mean: m)]
from: dist size: n
   ["example: ...from: (Poisson new mean: 0.5) size: 200"
   "get a fixed random sample from some distribution"
   self data: (dist generateSample: n)]
geometric: p "0.0 < p ≤ 1.0"
   [self distribution: (Geometric new p: p)]
normal: m sdev: s "¯1.0e4000 < m < 1.0e4000. s > 0.0"
   [self distribution: (Normal new mean: m asFloat sdev: s asFloat)]
poisson: m "m > 0.0"
   [self distribution: (Poisson new mean: m)]
uniform: a to: b
   [self distribution: (Uniform new from: a to: b)]
uniform: a to: b by: c
   [self distribution: (SampleSpace new data: (a to: b by: c))]
```

Exp.
Poisson
Geometric
Normal
Unif
Sample Space

Other Initialization

```
constant: constant
   [orderedSet ← distribution ← false]
data: vec "Vector/Interval/Set... (anything indexable with o)"
   ["random sampling with replacement"
   self distribution: (SampleSpace new data: vec)]
orderedSet: vec "Vector/Interval/Set...(anything recognizing asStream and
next)"
   [constant ← distribution ← false.
   "sampling will be done without replacement"
   orderedSet ← vec asStream]
```

Aspects

```
set
   [orderedSet→ [ñtrue] ñfalse]
```

Sampling

```
next | t
   [constant→ [ñconstant]
   orderedSet→ [ñorderedSet next]
   distribution→
      [t ← distribution random.
      t is: Float→ [ñt round]
      ñt]
   ñfalse]
_]
```

```
SystemOrganization classify: ↑Inputschedule under: 'Simulator'.┘
```

"PictureOrganizer"

```
Class new title: 'PictureOrganizer'
   subclassof: Dictionary
   fields: ''
   declare: '';
   asFollows_⌟

This class holds names and icons.  Access through global
PictureOrganization

Simulation Protocol
filout: str | name f
   [f ← dp0 file: str.
    f asParagraphPrinter stamp.
    forg name from: self do:
       [f cr; cr; append: 'PictureOrganization insert: '.
        f append: name asString; append: ' with: '.
        self∘name fullprinton: f.
        f append: '._⌟']
    f shorten close]
names
   [↑super contents]
newImages: vec| i t
   [forg i from: (1 to: vec length by: 2) do:
      [t←Traveller new size: (16⊙16).
       t assign: vec∘(i+1).
       super insert: vec∘i with: t.]
]
⌟
SystemOrganization classify: ↗PictureOrganizer under: 'Simulator'._⌟
```

"Set"

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

From Findit, for storing/collecting, read by a SetReader

Initialization

```
default |
   [self vector: 8]
of: array to: position |
   [limit← array length]
string: limit |
   [position ← 0. array ← String new: limit]
vector: limit |
   [position←0. array ← Vector new: limit]
```

Viewing

```
o i |
   [↑ array o i]
asSet |
   []
asStream |
   [↑ SetReader new of: array from: 1 to: position]
find: v | i
   [for: i to: position do: [array o i = v ⊃ [↑i]].
   ↑0]
length |
   [↑ position]
species |
   [↑ array species]
viewRange: i to: j |
   [↑ (SetReader new of: array from: (1 max: i) to: (j min: position))]
```

Changing

```
o i ← val |
   [↑ array o i ← val]
append: x | i
   [for: i from: x do: [self next ← i]]
delete: x | i
   [for: i to: position do: [array o i=x ⊃[↑self deletel: i]].
   ↑ false]
pastend ← x |
   [self append: self grow; next ← x]
```

Set Operations

```
printon: strm | t
   [strm append: 'a Set:'.
    array is: String ⊃ [strm space append: self]
    for: t from: self do: [strm space print: t]]
```

Private

```
deletel: i | r
```

```
    [r←self viewRange: i+1 to: position.
     position ← i-1.
     self append: r.
     array o (position+1) ← nil.]
grow
    ["self grown and rest. returns another Set iwth 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]
⌟
SystemOrganization classify: ↗Set under: 'Simulator'.⌟
```

"SetReader"

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

From Findit, read a set; no edits occur to set.

Initialization

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

Reading

```
asSet |
   [↑ self copy]
copy |  "yield contents all at once as a Set"
   [↑ [Set new of:(array species new: limit-position); append: self]]
↵
SystemOrganization classify: ↗SetReader under: 'Simulator'.↵
```

"Simulation"

```
Class new title: 'Simulation'
   subclassof: Object
   fields: 'stations              "<Set> of <Stations>"
            corridor              "<Set> of <Jobs> travelling between stations"
            document              "a <ReportWindow> for collecting statistics"
            door                  "a <Door>, a statistics gatherer"
            clock                 "a <Textframe> to display simulated time"
            numberOfTasks                 "statistics"
            idnumber              "identifies order of job arrival"
            eventlist             "an <Event>"
            currentTime           "simulated Time"
'

   declare: 'logFlag ';
   asFollows_J
```

The layout organizer, holds references to all objects in the Simulation, controls the layout of stations and the scheduling of events. Any special case, such as a car wash, is a subclass. Need another class to act as controller (playground) and handle illustrating the message sending mechanism.

Initialization

```
constructStation: station |
   [stations next ← station.
    station in: self.
    station show.]
init |          "does not place the clock correctly--do so in subclass"
   [eventlist ← Event new.
    currentTime ← 0.0.
    stations ← Set default.
    corridor ← Set default.
    document ← ReportWindow new title: 'Performance Measures' frame:
(Rectangle new origin: (380@418) corner: (510@670)).
    door ← Door init.
    clock ← Textframe new para: ('0' asParagraph) frame: (Rectangle new
origin: (self clockLocation + (2@4)) extent: (36@14)).
    numberOfTasks ← 0.
    idnumber ← 0.
    self layout.
    (clock frame inset: (¯2)@(¯4)) outline.
    clock show.
    self arrivalSchedule.
    self reportSchedule = 0 ⊃[]
    self schedule: self todo: ↱makeReport after: self reportSchedule]
newJob: job label: id after: time |
   [job id: id in: self.
    self schedule: job todo: ↱firsttime after: time.]
newJob: job name: id |
   [job id: idnumber in: self.
    self schedule: job todo: ↱firsttime after: 0.0.]
release | i
   [stations≡nil ⊃[↑self]
    for: i from: stations do:
      [i release]. stations←nil.
```

```
    forS i from: corridor doS
      [i release]. corridor ← nil.
    document ← nil.
    door←nil.
    eventlist release.
    eventlist ← nil.]
use: job assignments: assignments data: data
      | strm j
   ["This message is used to start a simulation
    with a given sequence of jobs to be done.
    job is a job like all the ones to be entered,
    assignments is a vector of assignments for
    each job.
    data is a vector consisting of starting times
    and features for the individual jobs."
   idnumber←0.
   strm←data asStream.
   untilS strm end doS
      [idnumber←idnumber+1.
      j←job class init.
      j id: idnumber in: self.
      j feature: strm next.
      j setTasks: assignments.
      self schedule: j todo: ↗firsttime
         after: strm next]]
use: job startAt: time schedule: dist assignments: vec
   [job taskMaster: self startAt: time asFloat schedule: dist assignments:
vec]
```

Scheduling

```
cycle | target selector t  wind "Run next scheduled event"
   ["selector can be an atom or vector containing selector and
     one or two arguments"
    eventlist end⊃[↑false]
    "wind←Dispframe new rect: ((1@1) rect: (100@50)).
   eventlist printon: wind. wind show."
    t ← currentTime asInteger.
    target ← eventlist target.
    selector ← eventlist selector.
    currentTime ← currentTime + eventlist simulatedTime.
    eventlist ← eventlist next.
    [t=currentTime asInteger ⊃ [] self setclock: t.].
    selector is: UniqueString ⊃ [target perform: selector]
    selector length = 2 ⊃
      [selector∘1≡↗goto:⊃[target goto: selector∘2]                    "explicit for speed"
      target perform: selector∘1 with: selector∘2]
    target perform: selector∘1 with: selector∘2 with: selector∘3]
enter |
   [door enter]
enterCorridor: job |
   [corridor next ← job]
exit: job | strm i
   [door leave.
    door time: job time.
    numberOfTasks ← numberOfTasks + job numberOfTasks.
    [logFlag⊃[job finalReportOn: document.
      document display]].
    job comp.
```

```
    for: i to: 20 do: [job moveby: (2@0).
                        for: i to: 100 do:[].]
    job disappear.]
has: pt | i
   ["used to see if want information about object in the simulation--
returns object:  station, corridor or document"
      document has: pt⊃ [^document].
      for: i from: stations asStream do:
         [i has: pt ⊃ [^i]].
      for: i from: corridor asStream do:
         [i has: pt ⊃ [^i]].
      ^false
      ]
leaveCorridor: job |
   [corridor delete: job]
schedule: target todo: selector after: time
   ["Queue target to receive selector after time"
    eventlist ← eventlist insert: target doing: selector after: time asFloat]
setclock: oldtime |i
   [for: i from: (oldtime to: currentTime by: 1) do:
    [clock show: i asString]]
tick | m
   [eventlist end⊃
      [currentTime ← currentTime+1.
       self setclock: currentTime asInteger]
    "peep to see if a full tick is too big a grain
      and if so cycle until clock changes instead"
    m←currentTime asInteger + 1.
   while: eventlist simulatedTime +currentTime < m do: [self cycle].
   eventlist simulatedTime: (eventlist simulatedTime+currentTime)-m.
   currentTime ← m asFloat.
    self setclock: currentTime asInteger.]
```

File in Jobs

```
add: job at: time
   [j←job init id: idnumber asString in: self.
    self schedule: j todo: ↗firsttime after: time.]
add: job at: time feature: f
   [j←job init id: idnumber asString in: self.
    j feature: f.
    self schedule: j todo: ↗firsttime after: time.]
filin: str | f job
   [idnumber←0.
    f←dp0 file: str.
    until: f end do:
       [job ← f upto: 036.
        idnumber ← idnumber+1.
        self's ('self ' + job)].
    f close.]
```

Show

```
show | i
   [for: i from: stations do: [i show].
    for: i from: corridor do: [i show].
    self setclock: currentTime asInteger.
    (clock frame inset: (~2)@(~4)) outline.
    clock show.
```

```
    document refresh.]

Query
corridor
    [↑corridor]
document |
    [↑ document]
time |
    [↑ currentTime]
whichStation: station | i
    [for: i from: stations do:
        [station = i name ⊃ [↑i]].
    ↑ false]

Report
clockLocation
    ["Return the location point for
     the clock (top left corner).
     Default location is next to the
     command menu."
     ↑(535@368)]
log [logFlag← true]
logging [↑logFlag]
makeReport
    [   [logFlag⊃[self reporton: document]].
     self reportSchedule = 0 ⊃ []
     self schedule: self todo: ↗makeReport after: self reportSchedule.]
nolog [logFlag← false]
probe: p | source x t
    [for: source from: (corridor, stations) do:
        [for: x from: source do:
            [t←x probe: p⊃[document append: (('
Observer probe at '+(currentTime roundTo: 0.01) asString+':
') asParagraph maskrunsunder: 2 to: 2).
                t reporton: document. document display.
                ↑true]]]]

report |
    [self reporton: document.]
reporton: strm | temp i                "put information into the document window"
    [strm append: ('
Report at time ' + currentTime asInteger asString + '
') asParagraph allBold; display.
     door reporton: strm.
     strm append: '
Avg max queue length: ' +
        [temp ← 0.
        for: i from: stations do:
            [temp ← temp + i maxQueueLength].
        ((temp asFloat/stations length) roundTo: 0.1) asString] + '
Avg no. of stations visited: ' +
        [door numberOfCustomersThrough > 0 ⊃
            [((numberOfTasks asFloat/door numberOfCustomersThrough)
roundTo: 0.1) asString]
        '?'] + '
'; display.
    for: i from: stations do:
```

```
        [i reporton: strm. strm display].
    temp ← Set default.
    for: i from: corridor do:
        [i is: Worker ⊃ [temp next ← i]].
    temp empty ⊃ []
    strm append: ('
Workers Travelling' asParagraph allStyle: 4) + '
'; display.
    for: i from: temp do: [i reporton: strm. strm display].
    ]
```

Models

```
arrivalSchedule
    "State the job, input schedule,
    and initial assignments (the
    names of stations the job will
    visit). For example"

    [ self use: job new
            startAt: 0
            schedule:
              (inputschedule new
                        constant: 10)
            assignments: 'Station'.]
reportSchedule
    "Specify when a full report should
    be made. Return 0 if no reports
    should be scheduled.
    For example, the default is to get
    a report every 24 hours"

    [ ⇑ 1440]
```

Printing

```
hardcopy: p | x
    [for: x from: stations do: [x hardcopy: p].
    clock hardcopy: p.
    p entity: (p transrect: user screenrect) containing:
        [for: x from: corridor do: [x presson: p]]]
```

Parts

```
layout | var workers
    ["Create the stations with workers."
     "Provide space for the station."
 var ← Station init
        of: (100@100 rect: 200@250).                    "Get the workers."
 var with: (Worker init id: 1).
     "Now construct the station."
     self constructStation: var.]
┘
SystemOrganization classify: ↱Simulation under: 'Simulator'.┘
```

```
                                                                    "Station"
Class new title: 'Station'
   subclassof: Object
   fields: 'custQ              "<CustQueue> of waiting jobs"
           clerklist           "<Set> of all workers available to the station,
the"
                               "busy workers remember the jobs being served"
           stationRect             "<Rectangle> where station is located"
           label               "<String> name of station"
           door                "<Door> for entering and leaving the station"
           layout              "<Layout> in which station is constructed,
                                eg., surgery in a hospital"
           doorQ            "<CustQueue> of jobs at the door"
           threshold           "max length of waiting line before calls for
more help"
        checkThresholdLoop              "wire in the travelling worker model""
   declare: 'SX SY SZ ';
   asFollows_!
```

Station has a queue of waiting jobs and workers servicing other jobs.
Create with Station init name: <String> of: <Rectangle> in: <Simulation>.
Should also specify set of workers to be created, and then show the station in order to give the workers locations.

Initialization

```
addWorker: worker | c i
   [worker in: self.
    self hideWorkers.
    self showWorkers.
    layout leaveCorridor: worker.
    layout schedule: worker todo: ↑callNextJob after: 0.0.
    checkThresholdLoop ← true.]
classInit
   [SX←SY←SZ←nil]
in: layout
init |
   [clerklist ← Set default.
    threshold ← 1000.
    checkThresholdLoop ← true.
    door ← Door init.
    doorQ ← CustQueue init.
    doorQ of: (Vector new: 2); showangle: (¯1@0).
    custQ ← CustQueue init.
    custQ of: (Vector new: 2);
          showangle: (1@0).                "default line is horiz left to right"
    ]
name: label in: layout
name: label of: stationRect|
   [door at: (stationRect origin + (0@ (stationRect height-16))).
    doorQ showat: (door location).]
name: label of: stationRect in: layout |
   ["The name can be nil, the stationRect has to be a rectangle, large
      enough to accommodate workers, waiting line, and jobs being
      served.  The layout is an instance of class Simulation that handles
      scheduling, statistics and editing, and debugging."
```

```
    door at: (stationRect origin + (0 ⊙ (stationRect height-16))).
    doorQ showat: (door location).]
of: stationRect |
   [door at: (stationRect origin + (0⊙ (stationRect height-16))).
    doorQ showat: (door location).
    label ← self class title.]
release | i
   [clerklist=nil ⊃ [↑self]
    for: i from: clerklist do:
       [i at: label ⊃ [i release]].
    clerklist ← nil.
    for: i from: custQ do: [i release].
    custQ ← nil.
    for: i from: doorQ do: [i release].
    doorQ ← nil.
    layout ← nil.]
show | k i t
   [stationRect outline.
    t ← Textframe new para: nil
            frame: (Rectangle new origin: stationRect origin extent:
(stationRect width⊙16)).
    t show: label.
    t ← 0.
    for: i from: clerklist asStream do:
       [t < i height ⊃ [t ← i height].].                "t is max worker height"
    (Rectangle new
          origin: (stationRect origin + (4 ⊙(t+24))) extent: ((stationRect width-
8) ⊙ 8))
       clear: gray.
    k ← stationRect origin + (4 ⊙(t+20)).                "bottom left corner of first
worker"
    for: i from: clerklist asStream do:
       [i at: label ⊃
          [i place: k-(0⊙i height).
           k ← k + ((i width+4) ⊙ 0).]].
                "allows different height workers"
    i ← [(custQ showangle) x < 0 ⊃ [stationRect width -20] 4].
    t ← [(custQ showangle) y < 0 ⊃ [stationRect height - 20] t+64].
    custQ show: (stationRect origin + (i⊙t)) in: stationRect.
    doorQ show: (door location) in: user screenrect.]
showangle: pt |
   [custQ showangle: pt]
threshold: threshold
use: list |
   [list is: Worker ⊃
       [clerklist← Set default. clerklist next ← list.]
    list is: Set ⊃ [clerklist ← list copy]
    clerklist ← Set new of: list to: list length]
with: list | i
   [[list is: Worker ⊃
       [clerklist← Set default. clerklist next←list.]
    list is: Set ⊃ [clerklist ← list copy]
    clerklist ← Set new of: list to: list length].
    for: i from: clerklist do:
       [i in: self]]
```

Query

```
exit|
```

```
    "constant offset reflects assumption on max job height"
    [↑ (stationRect corner - (16⊙16))]
firstWorkerLoc | c i
    [c←0.
     for: i from: clerklist asStream do:
        [c< i height ⊃ [c ← i height]].
     ↑stationRect origin+ (4 ⊙(c+20))]
has: pt |
    [↑stationRect has: pt]
hideWorkers | i
    [for: i from: clerklist asStream do:
        [i at: label⊃ [i hide]]]
layout
    [↑layout]
location |        "position of the door"
    [↑ door location]
maxQueueLength |  "asks custQ"
    [↑ custQ maxlength ]
name |
    [↑label]
nextPlace: job |
    [↑doorQ nextPlace: job in: user screenrect]
showWorkers | k i
    [k←self firstWorkerLoc.
     for: i from: clerklist asStream do:
            [i at: label ⊃
                [i place: k-(0⊙i height).
                 k←k+
                 (((i width max: i jobwidth) +4)⊙0)]].
    ]
time|
    [↑layout time]
waiting| "at all or above threshold?"
    [↑ custQ length > 0]
whereLastWorker | i j "returns origin of last worker if any"
    [i ← false.
     for: j from: clerklist asStream do:
            [j at: label ⊃ [i ← j where]].
    ↑ i]
workerDoor
    [↑(stationRect origin+ (stationRect extent x/2 ⊙0))]
workers
    [↑ clerklist asStream]
workersPresent | i j
    [j ← 0.
     for: i from: clerklist asStream do:
        [i at: label ⊃ [j←j+1]].
     ↑ j]
```

Scheduling

```
askForService: job | idleWorker pt
    ["get service or go into waiting line"
     (idleWorker ← self interruptWorker) ⊃ [self giveService: job by:
idleWorker]
     job moveto: (custQ nextPlace: job in: stationRect).
     custQ enQ1: job.
     checkThresholdLoop ⊃[self checkThreshold].]
callNextJobs | idleWorker
```

```
"from waiting line in station"
   [custQ empty ⊃ []
    (idleWorker ← self interruptWorker)⊃
      [self giveServiceBy: idleWorker]]
checkDoor | job
   [doorQ empty ⊃ []
    job ← doorQ dequeue.
    self askForService: job.
    doorQ showin: user screenrect.]
checkThreshold | i
   [checkThresholdLoop ⊃                          "can do the check"
    [custQ length < threshold ⊃ []                   "but have not reached threshold"
     for: i from: clerklist do:                      "have threshold, find someone"
       [i busy ⊃ []
        i at: label ⊃ []            "ok here is an idle worker we can request"
       (i travelfrom: (i station) to: self)⊃[checkThresholdLoop←false. ↑true]
                                                                  "he is coming"
        ].
     "no one could come"
     layout schedule: self todo: ↗checkThreshold after: 1.
     ↑false]
   ↑ false]          "not allowed to do the check"
enter: job |       "get to the door"
   [job moveto: (doorQ nextPlace: job in: user screenrect).
    doorQ enQ1: job.
    door enter.
    layout schedule: self todo: ↗checkDoor after: (self atEntrance: job).]
giveService: job by: worker
   [worker batchSize = 1 ⊃
            [worker takeJob: job.
             layout schedule: worker
                        todo: ↗giveupJob
                        after: (worker serviceTime: job).]
    job moveto: (custQ nextPlace: job in: stationRect).
    custQ enQ1: job.
    self giveServiceBy: worker.]
giveServiceBy: worker| i s c
   [custQ empty ⊃ []
    custQ length < worker batchSize ⊃[]
    worker batchSize = 1 ⊃
       [c←custQ dequeue.
        worker takeJob: c.
        custQ showin: stationRect.
        layout schedule: worker
                 todo: ↗giveupJob
                 after: (worker serviceTime: c).]
    s ← Set default.
    for: i to: worker batchSize do:
       [c ← custQ dequeue.
        c comp.
        c moveto: (worker where+ (0@32)).
        custQ showin: stationRect.
        s next ← c.]
    worker takeNoJobs: s.
    layout schedule: worker todo: ↗giveupJob after: (worker serviceTime: s).]
insert: jobs | i
   [custQ hide.
    for: i from: jobs do:
```

```
        [custQ push: i].
    custQ showin: stationRect.]
interruptWorker | i
   [for: i from: clerklist asStream do:
        [i busy ⊃ [ ]
         i at: label ⊃[ñi]].
    ñfalse]
leave: job | y
   [door leave.
    door time: layout time - job enterTime.                    "gather some statistics"
    self atExit: job.            "see if want to modify job agenda"
    y←0.
    until: (job stepby: (0⊙y) wrt: layout corridor) do:
       [y←y+job height].
    self callNextJobs.
       "not scheduled because want the next job in line, not anyone who
might be waiting atDoor"]
removeWorker: worker | i j
   [i ← clerklist find: worker.
    i=0 ⊃ [ñfalse]
    worker moveto: (self workerDoor - ((worker width/2 ) ⊙ worker height
)).
    layout enterCorridor: worker.
    worker in: (Station init name: 'Corridor' in: layout).]
workersMoveOverFrom: k to: newx | j i
"in case the job is wider than the worker"
   [(k > self workersPresent) ⊃ [ñfalse]
    j ← self whereLastWorker.
    j x ≥ newx ⊃ [ñfalse]            "do not bother shift left"
    j ← (newx - j x)⊙0.
    for: i from: [clerklist length to: k by: ¯1] do:
       [(clerklist∘i) at: label ⊃ [(clerklist∘ i) moveby: j]].
]
```

Report

```
probe: p | x t
   [stationRect has: p⊃
       [for: x from: clerklist do:
           [(t←x probe: p)⊃[ñt]].
       for: x from: custQ contents do:
           [(t←x probe: p)⊃[ñt]].
       ñself]
   for: x from: doorQ contents do:
           [(t←x probe: p)⊃[ñt]].
   ñfalse]
reporton: strm | i par cr
   [cr← '
'.
   par← ParagraphEditor new
       para: (cr + 'Station: ' + label) asParagraph allBold
       frame: (0⊙0) asRectangle.
   door reporton: par.
   par append: '
Max queue length: ' + (self maxQueueLength) asString.
   par append: cr+cr.
   par append:
       [self workersPresent=0 ⊃
           ['Currently, no workers are in this station.'+cr]
```

```
        ('Worker  %use  busy served
') asParagraph allStyle: 4].
    for: i from: clerklist do:
        [i at: label← [i reporton: par]].
    strm append: par contents]
```

Models

```
atEntrance: job
    "The time the job waits at the
    door before getting attention.
    Must return a number."

    [ ⇑ 0]
atExit: job |
    "Modification of the list of stations
    that the job is to visit. Example:
    job addTask: 'Station'.  No code
    means that the job is completed"

    [ ]
```

Printing

```
brightness [⇑255]
hardcopy: p |  i t
    [p box: stationRect hue: self hue sat: self saturation
        bright: self brightness containing:
        [label asParagraph presson: p                    "label"
            in: (p transrect: (stationRect inset: 2)).
        t ← 0.
        for: i from: clerklist do:
            [t← t max: i height].
        p bitmap: (Rectangle new                "copy gray stripe for now"
            origin: (stationRect origin + (4 ⊙(t+24)))
            extent: ((stationRect width-8) ⊙ 8)) bits: false.
        for: i from: clerklist do: [i presson: p].
        for: i from: doorQ contents do: [i presson: p].
        for: i from: custQ contents do: [i presson: p].]]
hue [⇑40]
saturation [⇑255]
┘
SystemOrganization classify: ↱Station under: 'Simulator'.┘
Station classInit┘
```

"Traveller"

```
Class new title: 'Traveller'
   subclassof: Form
   fields: 'screen          "saved screen image before places the traveller"
            loc            "last location of the traveller"
            opaque'
   declare: '';
   asFollows_]
```

A Traveller keeps an editable image that can be animated
Initialize with (Traveller new size: width@height) at: <Point>

Initialization

```
assign: other | "other must be another Traveller"
   [super replaceBy: other.
   opaque← other hull.
   screen← String new: data length]
at: loc [self pickupScreen: loc]
copy
   [↑Traveller new
      image: data copy hull: opaque copy bitsPerLine: extent x]
fullprinton: strm
   [strm append: '(Traveller new image: ↗';
      print: (data copyto: (Vector new: data length));
      append: ' hull: ↗';
      print: (opaque copyto: (Vector new: opaque length));
      append: ' bitsPerLine: '; print: extent x;
      append: ')']
hull: opaque screen: screen
image: i hull: h bitsPerLine: b
   [super data: i bitsPerLine: b.
   h length≠data length⊃[user notify: 'inconsistent hull']
   screen← String new: data length.
   opaque← [h is: String⊃[h] h copyto: (String new: h length)]]
size: ext
   [super size: ext.
   opaque← String new: data length.
   screen← String new: data length]
```

Movement

```
hide |
   [self putdownScreen: loc. screen ← nil.]
moveby: inc [self moveto: loc+inc]
moveto: dest          "restore background, move, save and reshow"
   [(Rectangle new origin: loc extent: extent)                "self asRectangle"
      bitsFromString: screen mode: storing.
   loc← dest.
   (Rectangle new origin: loc extent: extent)
      bitsintoString: screen mode: storing;
      bitsFromString: opaque mode: erasing;
      bitsFromString: data mode: oring]
moveto: dest mode: mode                "restore background, move, save and reshow"
   [ [screen=nil⊃[self pickupScreen: dest]].
   (Rectangle new origin: loc extent: extent)                "self asRectangle"
      bitsFromString: screen mode: storing.
```

```
    loc← dest.
    (Rectangle new origin: loc extent: extent)
        bitsIntoString: screen mode: storing;
        bitsFromString: data mode: mode]
place: loc [self pickupScreen: loc; moveto: loc]
show          "display without pickup"
    [self moveto: loc]
stepby: inc wrt: list | walker
    [for$ walker from: list do$
        [self≡walker ⊃ []
         self intersect: walker after: inc ⊃ [⍝false]].
     self moveby: inc]
```

Editing

```
o pt | i j mask
    [j ← origin x+pt x.
    i ← origin y+pt y*raster*2+(j/8)+1.
    mask ← ↗(0200 0100 040 020 010 4 2 1)∘(j\8+1).
    ⍝[(opaqueoi land: mask)=0⊃[2] "trans"
        (dataoi land: mask)=0⊃[0] "white"
        1]] "black"
o pt ← val | i j mask
    [j ← origin x+pt x.
    i ← origin y+pt y*raster*2+(j/8)+1.
    mask ← ↗(0200 0100 040 020 010 4 2 1)∘(j\8+1).
    dataoi ← [val=1⊃[dataoi lor: mask] dataoi land: ¯1-mask].
    opaqueoi ← [val=2⊃[opaqueoi lor: mask] opaqueoi land: ¯1-mask]]
clear: color [self sub: (zero rect: extent) clear: color]
comp
    [self show.          "show on screen"
    (Rectangle new origin: loc extent: extent)
        bitsFromString: opaque mode: xoring;                    "comp the hull"
        bitsIntoString: data mode: storing]                  " and store back in"
copy: trav
    [super copy: trav.
    self withHull$ [trav withHull$
        [super copy: trav]]]
showHullAt: pt magnified: scale
    [self sub: (0⊙0 rect: extent) showHullAt: pt magnified: scale]
showMagnified: scale at: pt | b m x y dot z xinc
    [dot← Rectangle new origin: 0⊙0 extent: scale⊙scale.
    b← 0. xinc← scale⊙0.
    for$ y to: self extent y do$
        [dot moveto: pt x⊙(pt y+(y-1*scale)).
        b← b+1|2+1. m← 0200.
        for$ x to: self extent x do$
            [ [m=0⊃[m← 0200. b← b+1]].
            z← dataob land: m.
            [z>0⊃[dot clear: black]].
            dot moveby: xinc.
            m← m/2]]]
"
Traveller allInstances notNil∘1 showMagnified: 5 at: user mp.
"
sub: rect | t
    [data⊃[t← Traveller new data: data raster: raster
            origin: origin+rect origin extent: rect extent.
        t hull: opaque screen: screen. ⍝t]
```

```
    ↑super sub: rect]
sub: rect clear: color
    [(super sub: rect)
        clear: [color=black⊃[black] white];
        data← opaque;                "kluge lets it also store into the hull"
        clear: [color≠gray⊃[black] white]]
sub: r showHullAt: pt magnified: scale | b m x y dot z xinc
    [dot← Rectangle new origin: 0⊙0 extent: (scale-1)⊙(scale-1).
    xinc← scale⊙0.
    for: y from: r minY to: r maxY-1 do:
        [x← r minX.
        b← y*raster*2+(x/8)+1. m← 1 lshift: 7-(x\8).
        dot moveto: (x⊙y)*scale + pt.
        for: x from: r minX to: r maxX-1 do:
            [[m=0⊃[m← 0200. b← b+1]].
            m=0200 and: opaque@b=0377⊃
                [dot moveby: xinc*8. x← x+7. b← b+1]
            z← opaque@b land: m.
            [z=0⊃[dot clear: gray]].
            dot moveby: xinc.
            m← m/2]]]
withHull: expr | v d
    [d← data. data← opaque. v← expr eval. data← d. ↑v]
```

Query

```
height |
    [↑ extent y]
intersect: walker after: inc
    [↑(((loc + inc) max: walker where) < ((loc + extent + inc) min: (walker
where+walker extent)))]
probe: p
    [loc≤p⊃
        [p-loc<extent⊃[↑self] ↑false]
    ↑false]
where |
    [↑ loc]
width |
    [↑ extent x]
```

Private

```
hull [opaque≠nil⊃[↑String new: data length]
    ↑opaque]
pickupScreen: at |
    [screen ← (Rectangle new origin: at extent: extent) bitsIntoString]
putdownScreen: at |
    [(Rectangle new origin: at extent: extent) bitsFromString: screen]
```

Printing

```
hue [↑0]
presson: p | unused mask i
    [unused← raster*16-extent x.                "have to clip for Dover"
    mask← ↑(0377 0376 0374 0370 0360 0340 0300 0200)@(unused\8+1).
    for: i from: raster*2 to: raster*2*extent y by: raster*2 do:
        [unused<8⊃[data@i← data@i land: mask.
                    opaque@i← opaque@i land: mask]
        data@i←0. data@(i-1)← data@(i-1) land: mask.
        opaque@i←0. opaque@(i-1)← opaque@(i-1) land: mask].
```

```
[ColorPrint⊃[p brightness: 255; saturation: 255; hue: self hue.
    p bitmap: (Rectangle new origin: loc extent: extent)
        bits: opaque]].
p brightness: 0.
p bitmap: (Rectangle new origin: loc extent: extent) bits: data]
┘
SystemOrganization classify: ↑Traveller under: 'Simulator'.┘
```

"Job"

```
Class new title: 'Job'
   subclassof: Traveller
   fields: 'label          "an identification <String>"
            agenda         "<Queue> of job tasks"
            counter        "taskmaster idcounter"
            enterTime      "time entered current station"
            timespent      "some statistics--total time spent so far
and"
            tasks          "<Set> of tasks already completed"
            feature        "to be used by the subclass if needed"
            tasksToDo
            tasksDone
            station
            layout
            inputSched     '
   declare: 'JZ JX JY ';
   asFollows_]
```

Job-View
label ✓
counter
enterTime
Timespent
Path

a Traveller moving from place to place under direction of an agenda.
Create with (Job init ia: <number> in: <Simulation>).
Can assign an agenda or let the simulation assign it.

Initialization

```
classInit
   [JX←JY←JZ←nil]
id: label in: layout | idform subform pic p
   [[label is: String ⊃ [] label ← label asString].
    p←self picture.
    pic← PictureOrganization lookup: p.
    pic ⊃
       [super assign: pic copy.
       self string: label font: 1.
       super at: (0@0).]
   user notify: p asString + ' is not a known picture'.]
in: layout
init |
   [timespent ← 0.0.
    tasksToDo ← Set default.
    tasksDone ← Set default.]
release
   [tasks←agenda←tasksToDo←tasksDone ←nil.
    layout ← nil. station←inputSched←nil.]
```

Management of the Agenda

```
addTask: task | t
   [[task is: UniqueString ⊃ [t← task asString]
     task is: String ⊃ [t←task]
     user notify: 'task must be name of a station'. ^false].
    t ← layout whichStation: t.
    t⊃[tasksToDo next ← t]
    user notify: task asString+' is not the name of a station in this
simulation'. ^ false]
appendTasks: tasklist | i
```

```
    [for$ i to: tasklist length do$
        [self addTask: (tasklist ∘i).]]
insertTask: task | l t
    [l←tasksToDo length.
     t ← tasksToDo asStream.
     tasksToDo ← Set new vector: l+1.
     self addTask: task.
     for$ l from: t do$ [tasksToDo next ← l].
    ]
nextTask | t
    [t ← tasksToDo ∘1.
     tasksDone next ← t.
     tasksToDo deletel: 1.
     ↑t.]
setTask: task  "in case of misspelling"
    [tasksToDo ← Set default.
     task is: Vector ⊃ [self appendTasks: task]
     self addTask: task]
setTasks: task
    [tasksToDo ← Set default.
     task is: Vector ⊃ [self appendTasks: task]
     self addTask: task]
```

Scheduling

```
disappear
    [self hide; release]
enter |
    [station enter: self]
firsttime |        "firsttime job enters the layout, see if anything to do"
    [enterTime ← layout time.
     layout enter.
     (station ← self nextTask) ⊃ [self enter] "ok get started"
     layout exit: self ]          "no, just loitering"
getNewTask | s              "see if have another task to do"
    [tasksToDo empty ⊃
        [timespent ← layout time - enterTime.
         layout exit: self].
     layout enterCorridor: self.
     s ← Vector new: 2.
     s ∘ 1 ← ↗goto:.
     s ∘ 2 ← ((tasksToDo∘1) nextPlace: self)-((extent x)⊙0).
     layout schedule: self todo: s after: self travelTime ]
goto: there | d            "move the Traveller toward location of next task"
    [[(d←there x-loc x)=0 ⊃
        [(d←there y - loc y)= 0 ⊃
            [layout leaveCorridor: self.
             station ← self nextTask.
             self enter.
             ↑ self]
        self stepby: (0⊙([d>0⊃ [(self speed min: d) max: 1] ((0 - self speed)
max: d) min: ¯1])) wrt: layout corridor]
        self stepby: (([d>0⊃[(self speed min: d) max: 1] ((0 - self speed) max:
d) min: ¯1])⊙0) wrt: layout corridor].
     layout schedule: self todo: (↗goto:, there) after: self travelTime. ]
leave | "has been getting service and can now leave"
    [station leave: self.                "gives station a chance to change the job agenda"
     self getNewTask.]
startInMiddle: task
```

```
    [self setTasks: task.
     enterTime ← layout time.
     layout enter.
     self getNewTask.]

TaskMaster
feature: feature
getJob | n
    [counter ← counter+1 \100.
     n← self class init.
     n in: layout.
     n feature: self setFeature.
     n setTasks: [agenda is: inputschedule ⇒ [agenda next] agenda].
     ↑n]
getNewArrival | n
    [n←self getJob.
     layout newJob: n label: (counter asString) after: 0.0.
     layout schedule: self todo: ↗getNewArrival after: inputSched next
asFloat.]
readAll | i n
    [while: (i←inputSched next) do:
        [n ← self getJob.
         layout schedule: n todo: ↗firsttime after: i] ]
taskMaster: layout startAt: t schedule: inputSched assignments: agenda
    [counter ← 0.
     inputSched set ⇒ [self readAll]
     layout schedule: self todo: ↗getNewArrival after: t.]

Query
completedTasks |
    [↑self tasksAsString: tasksDone asStream.]
enterTime |
    [↑enterTime]
feature
    [↑feature]
name |
    [↑label]
numberOfTasks |
    [↑(tasksDone position + tasksToDo position)]
property
    [↑feature]
remainingTasks |
    [↑self tasksAsString: tasksToDo asStream.]
tasksAsString: strm | outstrm i
    [outstrm ← Stream default.
     for: i from: strm do:
        [outstrm append: (i name); append: ' '.]
    ↑(outstrm contents)]
time
    [↑timespent]

Report
finalReportOn: strm | str cr
    [cr← '
'.
    strm append:
    (self class title + ' ' + label + ' Duration ' +
```

```
        (([layout≡nil⊃[timespent] layout time-enterTime])
            roundTo: 0.01) asString)
      asParagraph allBold +
    (cr + 'Visited: ' + self completedTasks +
        [str ← self remainingTasks.
        str length=0⊃[cr] cr + 'Will visit: ' + str + cr])+ ([feature≡nil ⊃ ['']
'Feature: '+ feature asString+'
'])
  ]
reportOn: strm| str
  [self finalReportOn: strm]
```

View

Models

```
picture
    "The name of the picture for the
    job. The default is a square
    shape."

  [ ⍝ 'default']
setFeature
    "Returns a descriptive property
    of the job"

  [ ⍝ nil]
speed
    "The number of display bits per
    travel time"

  [ ⍝ 4]
travelTime
    "The amount of time it takes the
    job to travel one display bit if it
    travel speed=1."

  [ ⍝ 0.1]
⌟
SystemOrganization classify: ↗job under: 'Simulator'.⌟
job classInit⌟
```

View

View

View

"Worker"

```
Class new title: 'Worker'
   subclassof: Traveller
   fields: 'label                "number identification"
           timebusy              "some statistics:  total time busy,"
           startTime             "starting at this time and"
           served              "<Number> of jobs served"
           busy                  "<Boolean>--true if busy, false if idle"
           station
           jobs
           tasksToDo'
   declare: 'WZ WX WY ';
   asFollows_!
```

a Traveller acting as a worker servicing jobs at a station.
Create with (Worker init id: <Number> in: <Station>).

Initialization

```
classInit
   [WX ← WY←WZ←nil]
id: label
id: label in: stat
   [[station≡nil ⊃ []
     station name = 'Corridor' ⊃
                        [station release]].
     station ← stat.]
in: stat
   [[station≡nil ⊃ []
     station name = 'Corridor' ⊃
                        [station release]].
     station ← stat.]
init | p pic
   [timebusy ← 0.0.
    served ← 0.
    busy ← false.
    tasksToDo ← Set default.
    p←self picture.
    pic← PictureOrganization lookup: p.
    pic ⊃ [super assign: pic copy.]
    user notify: p asString + ' is not a known picture'.]
labelPicture | slabel
   [[label is: String ⊃ [slabel←label.] slabel ← label asString].
    self string: slabel font: 1.]
release | i
   [tasksToDo ≡nil ⊃ [↑self]
    station ← nil.
    tasksToDo ← nil.
    [busy ⊃ [for: i from: jobs do:
                  [i release]]].
    jobs←nil.].
]
```

Scheduling

```
callNextJob|
   [station giveServiceBy: self]
```

```
giveupJob | i j k
   [jobs=nil ⊃ []
    "compute statistics and make worker idle"
    served ← served + (k ← jobs length).
    timebusy ← timebusy+(station time-startTime).
    busy ← false.
    self comp.
    j ← jobs asStream.                "remove multiple jobs in reverse order"
    jobs←Set default.
    for: i from: (k to: 1 by: ¯1) do:
      [(j oi) comp; moveto: (station exit).
       (j oi) leave.]
]
goto: newStation | there d
   [there ← (newStation workerDoor - ((extent x/2)⊙extent y)).

      [(d←there x - loc x)=0⊃
         [(d←there y - loc y) = 0 ⊃
            [busy ← false.
             newStation addWorker: self. ⇑self]
      self moveby: 0⊙([d>0⊃[(self speed min: d) max: 1] (0 - self speed
max: d) min: ¯1])]
   self moveby: ([d>0⊃[(self speed min: d) max: 1] (0 - self speed max: d)
min: ¯1])⊙0].
    (station layout) schedule: self todo: (↱goto:, newStation) after: self
travelTime. ]
putBackJobs
   [station insert: jobs.
    jobs←nil.
    busy←false.]
takeJob: job |          "if a job is being served, then it is
                          the worker, not the station directly, that knows"
   [startTime ← station time.
    station workersMoveOverFrom: (label+1)
            to: (loc x + 4 +(extent x max: job width)).
    busy ← true.
    self comp.
    job moveto: (loc+(0⊙32)); comp.
    jobs ← Set default.
    jobs next ← job.]
takeNoJobs: jobs |
   [busy ← true.
    station workersMoveOverFrom: (label+1)
            to: (loc x + 4 +(extent x max: self jobwidth)).
    self comp.
    startTime ← station time.]
travelfrom: oldStation to: newStation |
   [station waiting ⊃ [⇑false]
    oldStation removeWorker: self.
    busy ← true.
    (station layout) schedule: self todo: (↱goto:, newStation) after: self
travelTime.
    ⇑true]
travelto: newStation | c
   [[busy ⊃
         [self putBackJobs]].
   c← [newStation is: Station⊃ [newStation]
   (station layout) whichStation: newStation].
```

```
    c ⊃ [self travelfrom: station to: c]
    user notify: 'worker tasks list must specify existing stations only.']
```

Management of the Agenda

```
addTask: task | t
   [[task is: UniqueString ⊃ [t← task asString]
     task is: String ⊃ [t←task]
     user notify: 'task must be name of a station'. ↑false].
    t ← station layout whichStation: t.
    t⊃[tasksToDo next ← t]
    user notify: task asString+' is not the name of a station in this
simulation'. ↑ false]
appendTasks: tasklist | i
   [for: i to: tasklist length do:
      [self addTask: (tasklist oi).]]
nextTask | t
   [t ← tasksToDoo1.
    tasksToDo deletel: 1. ↑t]
setTasks: task [
    task is: Vector ⊃ [
      tasksToDo ← Stream new of:
          (Vector new: task length).
      self appendTasks: task]
   tasksToDo ← Stream new of: (Vector new: 1).
   self addTask: task]
```

Query

```
at: stationName
   [↑ [station≡nil ⊃ [false]
       stationName = station name]]
busy |
   [↑busy]
job
   [↑jobs]
jobwidth | k i
   [k←0.
    for: i from: jobs do:
       [i width > k ⊃ [k←i width]].
    ↑k]
station
   [↑ station]
```

Report

```
probe: p | x
   [super probe: p⊃[↑self]
   for: x from: jobs do:
       [x probe: p⊃[↑x]].
   ↑false]
reporton: strm | t
   [t ← [station times0⊃[100.0]
          ((timebusy *100)/station time) roundTo: 0.1].
   strm append: (
       (self class title + ' ' + label asString) +
       [t<10.0⊃['   ']; <100.0⊃['  ' ] ' '] + t asString +
       [busy ⊃['    yes  ' ] '      no  ' ] +
       [served<10⊃['  ']; <100⊃[' ' ] ''] + served asString +'
'      ) asParagraph]
```

Image Display

```
hide | i
   [busy ⊃ [super hide.
            for: i from: jobs asStream do: [i hide].]
    super hide]
moveby: inc | i
   [busy ⊃ [ super moveby: inc.
             for: i from: jobs asStream do: [i moveby: inc.]]
    super moveby: inc.]
place: at | i
   [busy ⊃ [super place: at.
            for: i from: jobs asStream do: [i place: at+(0⊙32)].]
    super place: at]
show: at | i
   [busy ⊃ [ super show: at.
            for: i from: jobs asStream do: [i show: at+(0⊙32)].]
    super show: at]
```

Models

```
batchSize
   "Exact number of jobs that the
    worker must handle at one time"

   [ ⇑ 1]
picture
   "Name of the picture representing
    the worker. The default is a small
    rectangular shape."

   [ ⇑ 'default']
serviceTime: job
    "The time the worker spends
    giving service to the job.  It is
    possible that job is a set of jobs.
    Time might a function of the
    job feature."

   [ ⇑ 1.0]
setTaskSchedule
    "Initialize the worker's travelling
     schedule. This example has the
     worker move every hour. "

   [ (station layout) schedule: self
                      todo: ↗taskSchedule
                      after: 60. ]
speed
    "Number of display bits per travel
    time"

   [ ⇑ 4]
travelTime
   "The amount of time it takes the
    worker to travel one display bit if its
    travel speed=1."
```

```
[ ⇑ 0.1]
```

Printing

```
hue [⇑80]
presson: p | j
    [super presson: p.
    fors j from: jobs do$
        [j presson: p]]
```

Parts

```
taskSchedule | task
 "For the worker to service more than one station, the worker is first sent
the message setTaskSchedule. Suppose worker changes stations each hour. "
    [[tasksToDo empty ⊃
        [self setTasks: 'aStation', 'etc']].
    [tasksToDo end⊃ [tasksToDo reset]].

        task ← tasksToDo next.
        (station layout) schedule: self
                    todo: ↗travelto:, task
                    after: 0.
        self setTaskSchedule. ]
┘
SystemOrganization classify: ↗Worker under: 'Simulator'.┘
Worker classInit┘
```

XEROX XEROX

E A R S

Filename: simulationwindows.st

Creation Date: April 25, 1978 2:35 PM

Printed by: Weyer, Steve

XEROX XEROX

```
'From Smalltalk 5.3b/xm on 24 April 1978 at 9:50:29 am.'┘

                                                        "Errorframe"
Class new title: 'Errorframe'
   subclassof: Dispframe
   fields: 'saveContext wind saveFlag'
   declare: '';
   asFollows┘

This is intended to frame an error message within the context of the error

Window Protocol
eachtime | t
   [text window has: user mp⊃
      [user kbck⊃[t← self kbd⊃
         [ [t≡nil⊃[] self space; print: (nil run: t in: false)].
            self prompt]]]
   ↑false]
init
windowenter
   [NormalCursor topage1]
windowleave

Simulation Protocol
context: saveContext
done
   [wind done. wind←saveContext ← nil.]
flag: saveFlag
frame ← f
   [self rect: f]
from: wind
open | notifyWindow
   [wind leave.
    notifyWindow ← user notifier: '' stack: saveContext interrupt: saveFlag.
    notifyWindow ⊃ [user restartup: notifyWindow]]
proceed
   [NormalCursor topage1.
    wind done.
    saveContext push: nil.
    Top run: saveContext at: 1]
┘
SystemOrganization classify: ↑Errorframe under: 'SimulationWindows'.┘
```

"ErrorWindow"

```
Class new title: 'ErrorWindow'
   subclassof: PanedWindow
   fields: 'errorString saveContext playground'
   declare: '';
   asFollows_]
```

Used initially instead of notify window,
provides panic button to see notify window

Initialization

```
frame: frame
   ["textframe"
    (panes∘1) frame ← (Rectangle new origin: (frame origin) extent: (frame
extent - (51⊙0))).
   "menupane"
    (panes∘2) frame ← (Rectangle new origin: (frame origin + ((frame
extent x - 50) ⊙ 0)) extent: (50 ⊙ frame extent y)).]
from: playground
of: errorString from: context with: flag
   [(panes∘1) context:  context; flag: flag.
    (panes∘1) append: errorString; cr;
          tab; print: context; cr;
          append: 'It was sent from:'; cr;
          tab; print: context sender; outline.]
title: title frame: f | textpane menupane
   [user restoredisplay.
    textpane ← Errorframe new .
    menupane ← NoScrollMenuPane new.
    self title: title with: (textpane, menupane) at: nil.
    self frame: (self fixframe: f).
    textpane from: self.
    menupane for: textpane.
    menupane of: ↱(proceed done open).]
```

Event Responses

```
bluebug

done
   [(panes∘2) compselection; select: 0.
    self erase.
    panes ← nil.
    playground hideErrorWindow.
    playground ← nil.
    saveContext←nil.]
eachtime | pane
   [frame has: user mp ⊃
      [for: pane from: panes do:
         [pane startup]]
    self outside ⊃ []
    ⇑ false]
has: pt
   [⇑frame has: pt]
keyset

printon: strm
```

self close

```
    [strm append: 'An Error Window']
refresh
    [self enter; leave]
takeCursor
    [user cursorloc ← frame origin + ((frame extent x - 60)⊙ 20)]
┘
SystemOrganization classify: ↑ErrorWindow under:
'SimulationWindows'.┘
```

```
Class new title: 'ModelWindow'
   subclassof: PanedWindow
   fields: 'layoutpane'
   declare: 'titleframes stdTemplates ';
   asFollows_]
```

This is a Five-Paned Simulation Window consisting of:
super classes
sub classes
messages
methods or picture editors
menus

Initialization

```
classInit |
   [stdTemplates ← (0@0 rect: 12@11), (12@0 rect: 24@11), (24@0 rect:
36@11), (0@11 rect: 29@36), (29@11 rect: 36@36)]
from: layoutpane
   [(panes∘1) updateTo: layoutpane]
release
   [layoutpane release.
    layoutpane← nil.
    panes← nil.]
title: title frame: f| superclassPane subclassPane messagePane editPane
menuPane
   [superclassPane ← SuperClassPane new.
    subclassPane ← SubClassPane new.
    messagePane ← MessagePane new.
    editPane ← EditPane new.
    menuPane ← NoScrollMenuPane new.
    self title: title
          with: (superclassPane, subclassPane, messagePane, editPane,
menuPane)
          at: stdTemplates.
    self frame: (self fixframe: f); show.
    superclassPane in: title of: subclassPane with: messagePane.
    subclassPane from: superclassPane to: messagePane and: editPane.
    messagePane from: subclassPane to: editPane.
    editPane from: subclassPane and: messagePane to: menuPane.
    menuPane for: editPane.
    superclassPane update.]
```

Window Protocol

```
bluebug
close| pane
   [fors pane from: panes dos [pane close].
    (frame inset: 0@0-border) clear.
    ((0@(frame origin y- 20)) rect: (frame corner+ (15@15))) clear.]
eachtime| pane
   [((frame origin - (20@0)) rect: (frame corner + (20@0))) has: user mp ⊃
      [fors pane from: panes dos [pane startup]]
    self outside ⊃ []
    user kbck ⊃ [user kbd. frame flash.]
    ⇑false]
enter
```

```
    []
firsttime
    [((frame origin - (20⊙0)) rect: (frame corner + (20⊙0)))  has: user mp ⊃
        [self reset. ⇑self enter]
        ⇑false]
hardcopy: p | pane w
    [self showtitle.
    for: w from: titleframes do:
        [w hardcopy: p].
    for: pane from: panes do: [pane hardcopy: p]]
has: pt
    [⇑ frame has: pt]
keyset
leave
    []
printon: strm
    [strm append: 'A ModelWindow']
refresh | pane
    [(panes∘4) showscrollbar.
     self reset; showtitle.
     for: pane from: panes do: [pane windowenter].
    ]
showtitle | i w tf
    [titleframes← Vector new: 3.
    for: i to: 3 do:
        [w← (panes∘i) window.
        w← Rectangle new
            origin: w origin-(0⊙20)
            extent: w width⊙18.
        w clear: 0.
        tf← Textframe new
            para: (↱('Category' 'Component' 'Interface')∘i) asParagraph allBold
center
            frame: w.
        tf inset: 0⊙2; outline; show.
        titleframes∘i← tf]]
yellowbug
⌟
SystemOrganization classify: ↱ModelWindow under:
'SimulationWindows'.⌟
ModelWindow classInit⌟
```

"Playground"

```
Class new title: 'Playground'
   subclassof: PanedWindow
   fields: 'currentSim                "one of the <Layouts>, currently being
run"
           runflag              "while true, continuously runs the
currentSim"
           menulist             "for the menuPane"
           EXITFLAG
           fileCount
,
   declare: 'menutitles fontToggle ';
   asFollows_J
```

This paned window becomes the top level context.
Tell a playground to startup.
pane 1 categorypane
pane 2 versionpane
pane 3 command menu
pane 4 the current simulation statistics window
pane 5 the model window or the simulation window dummy.

```
Initialization
category: category
   [self clear: false.
    (panes∘2) update.
    runflag ← false.
    (panes∘5) close.
    panes∘ 5 ← SimulationWindow new.
    [currentSim ≡ nil ⊃ [] currentSim release].
    currentSim ← nil.
    menulist ← ↱(define printBW printC font quit leave).
    (panes∘3) of: menulist.
    (panes∘4) closefile.
    panes∘4 ← SimulationWindow new.
    user waitnobug.]
classInit
   [fontToggle←true.]
frame: frame |
   ["layoutPane"
    (panes∘2) frame ←
      (Rectangle new origin: (520⊙50) extent: (85⊙150)).
    "menuPane"
    (panes∘3) frame ←
      (Rectangle new origin: (520⊙ 230)
                          extent: 85⊙105) ]
init | t layoutPane menuPane
   [fileCount ← 0.
    t←Vector new: 6.
    t∘ 1 ← SimulationWindow new.
    t∘ 2 ← (layoutPane ← LayoutPane new).
    t∘ 3 ← (menuPane ← MenuPane new).
    t∘ 4 ← SimulationWindow new.
         "dummy editing window"
    t∘ 5 ← SimulationWindow new.
         "dummy workspace window"
```

```
    to 6 ← SimulationWindow new.
          "dummy error window"
    self title: 'Simulation View' with: t at: nil.
    self frame: (self fixframe: user screenrect); show.
    frame clear: gray.
    menulist ← ↱(define printBW printC font quit leave).
    menuPane for: self; of: menulist.
    menuPane outline; fill.
    layoutPane for: self; update; outline; fill.
    self label.
    to 4 ← ReportWindow new title: 'Simulation Description' frame:
(Rectangle new origin: (380⊙ 418) corner: (610⊙670)).
    to4 from: (dp0 file: 'Simulation.Description').
    (to4) refresh.
    currentSim ← nil.
    runflag ← EXITFLAG ← false.]
label | t i            "P label"
   [menutitles← Vector new: 2.
    for$ i to: 2 do$
      [t ← Textframe new
          para: (↱('Simulations' 'Commands')oi) asParagraph allBold center
          frame: (Rectangle new origin: 520⊙(↱(29 209)oi)
                extent: 85⊙19).
      t frame outline: 1.  t inset: 0⊙4; show.
      menutitlesoi← t].
   ]
reenter
   [(panesoo3) fill.
   EXITFLAG←false]
release | i
   [[currentSim ≡ nil ⊃ [] currentSim release].
    panes≡nil⊃[]
    for$ i to: 3 do$ [(panesoi) release].
    for$ i from: (4 to: 5) do$ [(panesoi) closefile].
    (panesoo6) done.
    panes←nil.]
resetFile
   [fileCount ← 0]
showError: errorString stack: context interrupt: flag
   [user restoredisplay.
    [runflag ⊃ [self stop]].
    panesoo6 ← ErrorWindow new title: 'Error' frame: (Rectangle new origin:
(200⊙200) extent: (200⊙150)).
   (panesoo6) of: errorString from: context with: flag.
   (panesoo6) from: self; takeCursor.
    (panesoo3) compselection; leave.
    thisContext sender ← nil.
    user run: true]
simulateWith: holdSim
   [[currentSim is: Simulation ⊃ [currentSim release]].
    currentSim ← holdSim.
    menulist←
      [(panesoo5) is: ModelWindow ⊃
            [↱(start done printBW printC font  quit leave)]
          ↱(start define printBW printC font  quit leave)].
    (panesoo4) closefile.
    panesoo4←SimulationWindow new.
    runflag ← false.
```

```
    self refresh.
    user waitnobug.]
```

Scheduling

```
bluebug
eachtime | pane p
    [until: EXITFLAG do:
        [[runflag ⊃
            [currentSim cycle.
            user anybug⊃[]
            ↑true]].          "Ignore other panes during normal running"
        for: pane from: panes do: [pane startup⊃[↑true]].
        currentSim is: Simulation ⊃ [
        user redbug⊃
            [XeqCursor showwhile: (currentSim probe: user mp)]]]
    ↑false]
firsttime
    [↑true]
lasttime |
    []
```

Simulation Protocol

```
clear: flag        | i
    [user screenrect clear.
     for: i to: 3 do: [(panes∘i) refresh].
     self label.
     flag⊃[(panes∘5) enter; leave.]]
cycle |
    [[runflag ⊃
        [self stop]].
     currentSim cycle.]
define | n
    ["This browser provides interface for redefining models for the
currentSim."
     panes∘5 ← ModelWindow new
        title: 'Category        | Component    |    Interface    '
        frame:  (30@418 rect: 340@670).
    (panes∘5) from: (panes∘2).
     currentSim≠nil⊃
        [menulist ← ↗(done printBW printC font quit leave).
         (panes∘3) revise: menulist with: nil]
     n ← menulist find: ↗define.
     self revisemenu: menulist∘(1 to: n-1)
            with: ↗(done)
                and: [n≥menulist length ⊃ [nil]  menulist∘((n+1) to: menulist
length)].]
done| t n
    [(panes∘5) close.
     panes∘5 ← SimulationWindow new.
     currentSim≠nil⊃
        [menulist ← ↗(define printBW printC font quit leave).
         (panes∘3) revise: menulist with: nil]
     n ← menulist find: ↗done.
     self revisemenu: menulist∘(1 to: n-1)
            with: ↗(define )
                and: [n≥menulist length ⊃ [nil]  menulist∘((n+1) to: menulist
length)].]
```

```
font
    [[fontToggle ⊃
          [fontToggle←false.
           DefaultTextStyle setfont: 0 name: 'SANS-SERIF12'.]
    fontToggle←true.
    DefaultTextStyle setfont: 0 name: 'SERIF12'.].
   self refresh.]
hideErrorWindow
   [panes∘6←SimulationWindow new.
    self refresh.]
leave
   [user unschedule: self.
    EXITFLAG ← true]
log | n
   [currentSim log.
   n← menulist find: ↗log.
   self revisemenu: menulist∘(1 to: n-1)
      with: ↗(nolog)
      and: menulist∘(n+1 to: menulist length)]
nolog | n
   [currentSim nolog.
   n← menulist find: ↗nolog.
   self revisemenu: menulist∘(1 to: n-1)
      with: ↗(log)
      and: menulist∘(n+1 to: menulist length)]
print | p t x
   [user displayoffwhile:
      [p ← dp0 pressfile: (t← (self title + '.press.') asFileName).
      p box: (frame inset: 1) hue: 0 sat: 0 bright: 140 containing: nil.
      super hardcopy: p.
      for: x from: menutitles do:
         [x inset: 2⊙2; hardcopy: p].
      [currentSim is: Simulation⊃[currentSim hardcopy: p]].
      p close.
      user quitThen:
'Empress ' + [ColorPrint⊃['VIOLA/H '+t] t] + '
Resume small.boot.
']]
printBW
   [Smalltalk define: ↗ColorPrint as: false.
    self print.]
printC
    [Smalltalk define: ↗ColorPrint as: true.
    self print.]
quit |
   [user quit.]
refresh | i
   [user screenrect clear.
    for: i to: 2 do: [(panes∘i) refresh].
    (panes∘3) outline; of: menulist.
    self label.
    [currentSim is: Simulation ⊃ [currentSim show] (panes∘4) refresh].
    for: i from: (5 to: 6) do: [(panes∘i) refresh].
    runflag ⊃ [user cursorloc ← ((panes∘3) frame origin + (¯30⊙¯30)).]]
report |
   [currentSim report.
    runflag ⊃ [user cursorloc ← ((panes∘3) frame origin + (¯30⊙¯30)).]]
restart | t str holdSim
```

```
    [(panes∘4) closefile.
     panes∘4 ← SimulationWindow new.
     Job classInit.
     Worker classInit.
     Station classInit.
     currentSim release.
     holdSim ← currentSim.  currentSim ← nil.
     self refresh.
     currentSim ← holdSim init.
     (panes∘3) fill; select: (panes∘3) selection.
     panes∘4 ← currentSim document.
     fileCount ← fileCount+1.
     t ← (panes∘2) selectionTitle+fileCount asString +'.perf'.
     (panes∘4) from: (dp0 file: t).
     str← 'REPORT '+fileCount asString+': '+ currentSim class title.
     str ← (str asParagraph) maskrunsunder: 4 to: 4.
     (panes∘4) cr; append: str; append: (('' asParagraph) maskrunsunder: 0
to: 0).
    (panes∘4) cr; display.
     (panes∘4) enter; leave.]
revisemenu: begin with: middle and: end | t n e
    [e← [end≠nil ⊃[end length] 0].
     n ← begin length + middle length + e.
     t ← Vector new: n.
     to(1 to: begin length) ← begin.
     to((begin length+1) to: (begin length+middle length)) ← middle.
     [e= 0 ⊃[] to (((n-e)+1) to: n) ← end].
     menuList ← t.
     (panes∘3) revise: menuList with: nil.
     ]
run |
    [runFlag ← true.
      self reviseMenu: ↱(stop)
              with: menuList∘(2 to: menuList length)
                 and: nil.
     (panes∘3) frame has: user mp ⊃
       [user cursorLoc ← ((panes∘3) frame origin + (¯50@10))].]
start | t str
    [Job classInit.
     Worker classInit.
     Station classInit.
     currentSim ← currentSim init.
     currentSim log.
     panes∘4 ← currentSim document.
     fileCount ← fileCount+1.
     t ← (panes∘2) selectionTitle+fileCount asString +'.perf'.
     (panes∘4) from: (dp0 file: t).
     str← 'REPORT '+fileCount asString+': '+ currentSim class title.
     str ← (str asParagraph) maskrunsunder: 4 to: 4.
     (panes∘4) cr; append: str; append: (('' asParagraph) maskrunsunder: 0
to: 0).
    (panes∘4) cr; display.
     self reviseMenu: ↱(run tick cycle restart refresh nolog report)
           with: menuList∘(2 to: menuList length)
          and: nil.
    user waitnobug.]
stop |
    [runFlag ← false.
```

```
    self reviseMenu: ↑(run )
                with: menuList ∘ (2 to: menuList length)
                and: nil .]
tick |
    [[runFlag ⊃ [self stop]].
    currentSim tick.]
┘
SystemOrganization classify: ↑Playground under: 'SimulationWindows'.┘
Playground classInit┘
```

12

Reset Windows / Paragraph Editor / Sweeper

how Many

$\prod_i \left[\text{for } i \text{ from:} \right.$

Repeat under certain times $\log_2 \varepsilon$

$\left. \sum_{i=n, l=T} i \text{ dist} \right]$

"ReportWindow"

```
Class new title: 'ReportWindow'
   subclassof: CodeWindow
   fields: ''
   declare: '';
   asFollows_⌋
```

Performance measurements are printed in this fixed size, editable window

Initialization

```
frame: frame | "fixes the menupane size to width of 50 and lets the
codepane be everything else"
   ["codepane"
    (panes@1) frame ← (Rectangle new origin: (frame origin+(1@1)) extent:
(frame extent - (52@1))).
    "menupane"
    (panes@2) frame ←  (Rectangle new origin: (frame origin + ((frame
extent x - 49) @ 1)) extent: (49@frame extent y-1)).]
from: file
   [(panes@1) from: file]
init |
   ["fool initialization for the playground"]
title: title frame: f | reportPane menuPane
   [reportPane ← ReportPane new.
    menuPane ← NoScrollMenuPane new.
    self title: title with: (reportPane, menuPane) at: nil.
    self frame: (self fixframe: f); show.
    menuPane for: reportPane.
    reportPane showing: (('' asParagraph) maskrunsunder: 0 to: 0); from:
reportPane.
   menuPane of: ↱(cut paste copy undo doit print).
      ]
```

Event Responses

```
bluebug |
   []
closefile
   [(panes@1) closefile]
eachtime | pane
   [((frame origin - (20@0)) rect: (frame corner + (20@0))) has: user mp ⊃
         [user bluebug ⊃ [Ωself bluebug]
          for$ pane from: panes do$ [pane startup]]
    self outside ⊃ []
    user kbd$ ⊃ [user kbd. frame flash] "flush typing outside"
    Ωfalse]
enter |
   []
firsttime
   [((frame origin - (20@0)) rect: (frame corner + (20@0)))  has: user mp ⊃
      [self reset. Ωself enter]
      Ωfalse]
has: pt |
   [Ωframe has: pt]
keyset |
   []
leave []
```

self close ?

report pane close file.

Report Pane

```
printon: strm |
    [strm append: 'A ReportWindow']
refresh | i
    [self reset; showtitle.
     for: i from: panes do: [i windowenter].
     (panes@1) showscrollbar.]
┘
SystemOrganization classify: ↑ReportWindow under:
'SimulationWindows'.┘
```

```
"SimulationWindow
"

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

False workpane for the playground

```
Initialization
close
closefile
done []
enter
firsttime |
   [↑false]
hardcopy: ignored
init
leave
outline
refresh
↵
SystemOrganization classify: ↗SimulationWindow under:
'SimulationWindows'.↵
```

E A R S

Filename: simulationpanes.st

Creation Date: April 25, 1978 2:33 PM

Printed by: Weyer, Steve

```
'From Smalltalk 5.3b/xm on 24 April 1978 at 9:45:28 am.'_!
```

"MenuPane"

```
Class new title: 'MenuPane'
    subclassof: ListPane
    fields: 'reportpane'
    declare: 'commandmenu ';
    asFollows_!
```

I am a ListPane with new interface serving as a menu for a reportwindow

Initialization

```
for: reportpane | "I send my selections to reportpane"
    []
init
    [self para: nil frame: nil style: MenuPaneStyle]
release
    [reportpane ← nil]
```

Window Protocol

```
bluebug |
    []
hue [↑80]
redbug | newSel pt "provides interface similar to class Menu"
    [pt ← self locked ⊃ [pt flash].
     newSel ← 0.
     while: (window has: (pt ← user mp)) do:
       [user redbug ⊃
           [newSel ← (pt y - window origin y)/self lineheight+firstShown.
            selection=newSel⊃ []
            [(1 max: firstShown) ≤ newSel and: newSel ≤ (list length min:
lastShown)⊃ [] newSel←0].
            self compselection.
            selection ← newSel.
            self compselection]
          newSel =0 ⊃[]
          selection ≠0⊃
             [XeqCursor showwhile:
                 [newSel ← 0. self selected]. ↑self]]
     self compselection; select: 0; deselected.
     ↑false]
refresh
    [self outline; fill; grayselection]
selected |
    [scrollBar hide.
     reportpane perform: list∘selection.
     scrollBar show.
     self compselection; select: 0.]
selection
    [↑selection]
_!
SystemOrganization classify: ↗MenuPane under: 'SimulationPanes'._!
```

"CategoryPane"

```
Class new title: 'CategoryPane'
   subclassof: MenuPane
   fields: ''
   declare: '';
   asFollows_┘
```

This class displays the fixed categories available in the curriculum

```
ListPane Protocol
enter
   [scrollBar show.
    self outline; fill; select: selection]
leave
   [scrollBar hide.
    self compselection; grayselection.]
selected
   [reportpane category: (Smalltalk lookup: listoselection).
    self select: selection.]
selectiontitle
   [↑((Smalltalk lookup: listoselection) title)]
┘
SystemOrganization classify: ↗CategoryPane under: 'SimulationPanes'._┘
```

"LayoutPane"

```
Class new title: 'LayoutPane'
    subclassof: MenuPane
    fields: 'categorypane'
    declare: '';
    asFollows↵
```

This class displays the layouts currently available within the curriculum category

Initialization

```
from: categorypane
release
    [categorypane ← nil. super release]
selectiontitle
    [↑(listoselection) asString]
update |
    [self of: (SystemOrganization category: 'Simulation').]
```

ListPane Protocol

```
enter
    [scrollBar show.
     self outline; fill; select: selection]
leave
    [scrollBar hide.
     self compselection; grayselection.]
newUpdate | n
    [selection=0 ⊃[self update. self select: 0]
     n ← listo selection.
     self update.
     selection←list find: n.
     self grayselection]
selected
    [scrollBar hide.
     reportpane simulateWith: (Smalltalk lookup: listoselection).
     scrollBar show.
     self select: selection.]
↵
SystemOrganization classify: ↗LayoutPane under: 'SimulationPanes'.↵
```

"MessagePane"

```
Class new title: 'MessagePane'
   subclassof: MenuPane
   fields: 'methodpane subclasspane superclasschoice
subclasschoice'
   declare: '';
   asFollows_!
```

This class displays the fixed selectors available for each superclass

Initialization

```
from: subclasspane to: methodpane
selectors: superclasschoice
```

Window Protocol

```
close
   [methodpane ← subclasspane ← nil.
    super close.]
hardcopy: p
   [superclasschoice=nil ⊃ [super hardcopy: p]
    (superclasschoice∘1) = ↗Pictures ⊃
       [p box: window hue: self hue sat: self saturation
          bright: self brightness containing:
          [p hue: 0; bitmap: window bits: false]]
   super hardcopy: p]
redbug
   [superclasschoice=nil ⊃ []
    subclasschoice=0⊃[]
    subclasschoice=↗new ⊃ []
    (superclasschoice∘1) = ↗Pictures ⊃
       [methodpane editpicture: subclasschoice form: ((PictureOrganization
lookup: subclasschoice) copy).
          selection←0.]
    ↑super redbug]
windowenter | icon
   [superclasschoice=nil ⊃ [super windowenter]
    subclasschoice=0 ⊃ [super windowenter]
    subclasschoice=↗new ⊃ [super windowenter]
    (superclasschoice∘1) = ↗Pictures ⊃
          [self outline.
           icon ← PictureOrganization lookup: subclasschoice.
           icon place: (window center-(icon extent/2))]
    super windowenter.]
yellowbug
```

ListPane Protocol

```
deselected
   [subclasschoice≠↗new ⊃[methodpane notshowing]]
dirty
   [↑methodpane dirty]
selected | str
   [subclasschoice canunderstand: list∘selection ⊃
      [methodpane showing: (subclasschoice code: list∘selection)]
    str ← Smalltalk lookup: (superclasschoice∘1).
    methodpane showing: (str code: list∘selection)]
```

Model Protocol

```
acceptpicturename: iconname form: icon
   [PictureOrganization insert: iconname with: icon.
    subclasspane update.]
choice: subclasschoice | strm c
   [subclasschoice = 0 ⇒
         [self of: (Vector new: 0).]
    subclasschoice=↱new ⇒
         [self of: (Vector new: 0).
          methodpane newclass: (superclasschoice∘1)]
   (c ← Smalltalk lookup: subclasschoice) ⇒
              [self of:
              (↱((layout arrivalSchedule reportSchedule )
                 (atEntrance: atExit:)
                 (picture batchSize serviceTime: speed travelTime
setTaskSchedule taskSchedule)
                 (picture setFeature speed travelTime ))
                 ∘ (superclasschoice ∘2)).
     subclasschoice ← c.]]
compile: parag | str
   [subclasschoice=↱new ⇒
      [str ← parag text delete: 32.
       str ← str delete: 13.
       str←str delete: 9.
      user displayoffwhile: [Class new title: str subclassof: (Smalltalk
lookup: (superclasschoice∘1)) fields: '' declare: ''.
       subclasspane classify: str unique.
       subclasspane update.]]
    ⇑methodpane compile: parag in: subclasschoice under: 'Parts'.]
displaypicture: icon name: subclasschoice
   ["show the traveller in the pane"
     selection ← 0.
     window clear: white.
     icon place: (window center − (icon extent/2)).]
requestpicturename: form formerly: iconname
   [methodpane copypicture: form name: iconname]
setchoice: subclasschoice
⌟
SystemOrganization classify: ↱MessagePane under: 'SimulationPanes'.⌟
```

"NoScrollMenuPane"

```
Class new title: 'NoScrollMenuPane'
   subclassof: MenuPane
   fields: ''
   declare: '';
   asFollows_┘
```

A menuPane that does not have a scroll bar

```
ListPane Protocol
enter
hue [↑200]
leave
outside
   [↑false]
selected |
   [reportpane perform: list∘selection.
    self compselection; select: 0.]
_┘
SystemOrganization classify: ↑NoScrollMenuPane under:
'SimulationPanes'._┘
```

"ReportPane"

```
Class new title: 'ReportPane'
    subclassof: CodePane
    fields: ''
    declare: '';
    asFollows_]
```

I am a Window for editing a paragragh or a picture with redbug and kbd, ignores bluebug and yellowbug

```
Initialization
from: selectorPane | t
    [selectorPane is: File ⊃
        [t←selectorPane contents.
         t length < 2 ⊃ []
         pared contents unpackFromString: t.]]
showscrollbar
    [scrollBar show]

Window Protocol
bluebug |
    []
enter
    []
firsttime
    [((frame origin - (20@0)) rect: (frame corner + (20@0))) has: user mp ⊃
        [self reset. ∩self enter]
        ∩false]
frame
    [∩frame]
keyset |
    []
leave
    []
redbug
    [pared selecting]
showing: paragraph
    [super showing: paragraph.
     scrollBar show.]
value
    [∩self]
yellowbug |
    []

Text Editing
accept
    [pared formerly⊃ [selectorPane compile: pared contents⊃
        [pared formerly: false]]
     frame flash.]
cancel
    [pared formerly ⊃ [self showing: pared formerly]
     frame flash]
closefile
    [selectorPane reset; append: pared contents packIntoString; shorten.]
copy |
    [pared copy]
```

Shrink – new Shrink

close w/ pared ← n.l ?

```
cut |
    [pared cut]
doit |
    [pared Scrap ←
        (self execute: pared selectionAsStream for: self)
     asString asParagraph]
paste |
    [pared paste]
print | PP
    [user displayoffwhile: [
    PP←PressPrinter init of: (dpO file: 'temp.press').
     PP print: pared contents.
     PP close.
     user quitThen: 'Empress temp.press; res small.boot
'.]]
rectangle | rect
    [rect ← Rectangle new fromuser.
     rect outline.
     pared Scrap ← rect asString asParagraph.
     pared paste]
undo |
    [pared undo]
⌟
SystemOrganization classify: ↑ReportPane under: 'SimulationPanes'.⌟
```

"ReportwithPicPane"

```
Class new title: 'ReportwithPicPane'
   subclassof: ReportPane
   fields: 'form small large scale picture'
   declare: '';
   asFollows_⌋
```

I am a Window for editing a paragragh or a picture with redbug and kbd, ignores bluebug and yellowbug

```
Initialization
from: selectorPane "ignore Citation"
   [picture ← false]
of: form
   [growing←false. scale ← 9.]

Window Protocol
hardcopy: p | t
   [picture⊃
      [t← Textframe new.          "borrow color"
      p box: frame hue: t hue sat: t saturation
         bright: t brightness containing:
         [p hue: 0; bitmap: frame bits: false]]
   super hardcopy: p]
redbug | pt lrect val
   [picture ⊃[pt ← user mp. lrect ← large asRectangle.
   lrect has: pt⊃
      [pt ← pt-large origin/scale.
      val ← (form∘pt)+1 \3.
      while: user redbug do:
         [pt ← user mp.
         lrect has: pt⊃ [self∘(pt-large origin/scale) ← val]]]]
    pared selecting]
show
   [picture ⊃ [self pictureshow] super show]
windowenter
   [picture⊃ [self pictureshow]
    super windowenter]

Form Editing
∘ pt ← val | color
   [form∘pt ← val.
   color← [val=0⊃[white]; =1⊃[black] gray].
   (small sub: (pt rect: pt+1)) clear: [color=black⊃[black] white].
   (large sub: (pt*scale rect: pt+1*scale-1)) clear: color]
black
   [self paint: black]
done
   [selectorPane requestpicturename: form formerly: picture.
    picture ← false.]
gray
   [self paint: gray]
paint: color | r x y
   [r ← (Rectangle new fromuser + (scale/2) intersect: large asRectangle) -
      large origin / scale.
   r empty⊃ [frame flash]
```

```
    form sub: r clear: color.
    self showsmall.
    color=gray⊃[form sub: r showHullAt: large origin magnified: scale]
    r←small origin blowup: r origin+scale+large origin by: scale spacing: 1]
picture: picture
pictureshow | se lo le
    [super outline.
    growing⊃ []
    se ← frame extent min: form extent.
    small ← Form new at: (Rectangle new origin: frame origin extent: se).
    lo ← frame origin x⊙small corner y.
    le ← (frame corner-lo/scale min: form extent)*scale.
    large ← Form new at: (Rectangle new origin: lo extent: le).
    self showsmall.
    (Rectangle new origin: small origin extent: le/scale) blowup: large origin
by: scale spacing: 1.
    form showHullAt: lo magnified: scale]
scale| r
    [r ← Rectangle new fromuser extent/form extent.
    scale ← (r x max: r y) max: 2.
    self pictureshow]
showsmall
    [small copy: (form sub: (0⊙0 rect: small extent))]
size | r newform overlap
    [r ← Rectangle new fromuser-large origin/scale.
    r empty⊃ [frame flash]
    newform ← Traveller new size: r extent.  newform clear: white.
    overlap ← r intersect: (0⊙0 rect: form extent).
    (newform sub: overlap-r origin) copy: (form sub: overlap).
    form assign: newform.  self pictureshow]
store
    [selectorPane acceptpicturename: ((pared contents text) delete: 32) form:
form]
white
    [self paint: white]
⌟
SystemOrganization classify: ↗ReportwithPicPane under:
'SimulationPanes'.⌟
```

"EditPane"

```
Class new title: 'EditPane'
   subclassof: ReportwithFicPane
   fields: 'subclasspane messagepane menupane'
   declare: 'iconmenu codemenu icon iconname ';
   asFollows_┘
```

This class provides the area for displaying the method code or a form editor. Are editing a form when the iconname is not nil.

Initialization

```
classInit
   [codemenu ← ↱(cut paste copy undo cancel doit rectangle accept).
    iconmenu ← ↱(black white gray done)]
from: subclasspane and: messagepane to: menupane
   ["and the report pane has a selectorpane=messagepane"
    self from: messagepane]
```

Text Editing

```
newclass: superclassname
   [picture←false.
    self showing: '      replace with '+ superclassname asString + ' name'.
    menupane of: codemenu.]
notshowing
   [picture←false. super showing: ''.
    menupane of: (Vector new: 0).]
showing: code
   [picture ← false.
    super showing: code.
    menupane of: codemenu.]
windowleave
```

Form Editing

```
copypicture: form name: iconname
   [picture ← false.
    super showing: iconname.
    menupane of: ↱(cut paste cancel store)]
editpicture: iconname form: icon
   [picture ← iconname.
    super of: icon; pictureshow.
    menupane of: iconmenu]
newpicture
   [picture ← false.
    super showing: '      replace with name of picture'; of: (Traveller new
size: (16@16)).
    menupane of: ↱(cut paste store).]
┘
SystemOrganization classify: ↱EditPane under: 'SimulationPanes'._┘
EditPane classInit_┘
```

clrl
[subclasspane ← nil, menupane ← nil, messagepane ← nil,
not clrl]

"SubClassPane"

```
Class new title: 'SubClassPane'
   subclassof: MenuPane
   fields: 'messagepane methodpane superclasspane pictureflag'
   declare: '';
   asFollows↘
```

This is a list of the subclasses for the current simulation category

```
Initialization
from: superclasspane to: messagepane and: methodpane
   [pictureflag ← false]

Window Protocol
close
   [superclasspane ← messagepane ← methodpane ← nil. super close.]
yellowbug

ListPane Protocol
deselected
   [messagepane choice: 0]
dirty
   [↑methodpane dirty]
selected | icon
   [pictureflag ⊃
      [(list∘selection) = ↗new ⊃
         [messagepane of: (Vector new: 0).
          methodpane newpicture.
          messagepane setchoice: ↗new.]
       icon ← Traveller new
             assign: ((PictureOrganization lookup: (list∘selection)) copy).
       messagepane displaypicture: icon name: list∘selection.
       methodpane notshowing.]
    messagepane choice: (list∘selection)]

Model Protocol
classify: classname
   [superclasspane classify: classname]
notpictures
   [pictureflag←false.
    methodpane picture: false.]
showpictures
   [pictureflag ← true.
    messagepane choice: 0.]
update
   [superclasspane selected]
↘
SystemOrganization classify: ↗SubClassPane under: 'SimulationPanes'.↘
```

"SuperClassPane"

```
Class new title: 'SuperClassPane'
   subclassof: MenuPane
   fields: 'category subclasspane messagepane layoutpane'
   declare: '';
   asFollows_⌋
```

This is a selection of simulation components

```
Initialization
in: category of: subclasspane with: messagepane
update
   [self of: ↱(Simulation Station Worker Job Pictures).]
updateTo: layoutpane


Window Protocol
close
   [subclasspane ← messagepane ← layoutpane ← nil.
    super close.]
yellowbug

ListPane Protocol
deselected
   [subclasspane of: (Vector new: 0).
    messagepane selectors: nil; choice: 0.]
dirty
   [↑subclasspane dirty]
selected
   [subclasspane of: self subclasses.
    messagepane selectors: ((listo selection), selection) .]

Model Protocol
classify: classname
   [SystemOrganization classify: classname under: (listoselection).
    SystemOrganization classify: classname alsounder: category.
    selection=1⊃ [layoutpane newUpdate].
]
subclasses |
   [(listo selection) = ↱Pictures ⊃
         [subclasspane showpictures.
          ↑ (↱(new ) concat: PictureOrganization names sort)]
    subclasspane notpictures.
    ↑(↱(new) concat: (SystemOrganization category: listoselection) sort)]
⌋
SystemOrganization classify: ↱SuperClassPane under: 'SimulationPanes'._⌋
```