RP-131

UserView understands 'print: printee
[printee printon: disp]

"

The New Improved SMALLTALK System Definition

June 28, 1977

This is the new SYSDEFS

This is <INGALLS>bootstrap.st. The read routine, compiler and file system(draft) appear in a separate file called bootstrap2.st. The user interface(draft) is in userface.st. The objects which are initially needed get created in the file launch.ft. A complete simulator exists in simulate.ft, and old-system defs of the new classes are in genclass.ft"

"RAW MATERIALS"

Class new title: 'Object';
subclassof: nil;
abstract~

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

"primitives"
Object understands: '@ x [] primitive: 4'! *"test for identical pointers"*
"Font will be edited so this looks like ≡, meaning eq"
Object understands: 'hash [] primitive: 46'! *"pointer as an Integer"*
Object understands: 'asOop [] primitive: 46'! *"Dont override this"*
Object understands: 'refct [] primitive: 45'! *"current reference count"*
Object understands: 'class [] primitive: 27'! *"class of this object"*
Object understands: 'instfield: n [] primitive: 38'! *"subscript any objct"*
Object understands: 'instfield: n ← val [] primitive: 39'!

"boolean connectives"
Object understands: 'or: x [self⇒[⇑true] ⇑x]'!
Object understands: 'and: x [self⇒[⇑x] ⇑false]'!
Object understands: 'xor: x [x⇒[⇑self@false] ⇑self]'!
Object understands: 'eqv: x [x⇒[⇑self] ⇑self@false]'!

"following don't evaluate their arg unless necessary.
They are built for comfort, not for speed."
Object understands: 'or: x [self⇒[⇑true] ⇑x eval]'!
Object understands: 'and: x [self⇒[⇑x eval] ⇑false]'!

"default protocol"
Object understands: 'printon: strm
[self@nil⇒ [strm append: "nil"]
self@false⇒ [strm append: "false"]
self@true⇒ [strm append: "true"]
self class print: self on: strm]'!
Object understands: 'asString | strm
[strm ← Stream default.
self printon: strm. ⇑strm contents]'!
Object understands: 'print

```
    [user show: self asString]'!
Object understands: '= x [↑self@x]'!
Object understands: '≠ x [↑self=x@false]'!
Object understands: 'is: x [↑self class@x]'!
Object understands: '_ x | v
    [v ← Vector new: 2.
    v˙1 ← self. v˙2 ← x.  ↑v]'!
Object understands: 'startup  "loopless scheduling"
    [self firsttime⇒
        [while▫ self eachtime do▫ [].
        ↑self lasttime]
    ↑false]'!
Object understands: 'canunderstand: selector
    [↑self class canunderstand: selector]'!
Object understands: 'copy      "create new copy of self"
    [↑self class copy: self]'!
Object understands: 'recopy"recursively copy whole structure"
    [↑self class recopy: self]'!
Object understands: 'error
    [user notify: "Message not understood."]'!
Object understands: 's code
    [self class understands: "doit [↑[" + code + "]]".
    ↑self doit]'!
```

"FUNDAMENTAL ORGANIZATION"

```
Class new title: 'Class';
    fields: 'title "<String> for identification, printing"
            myinstvars "<String> partnames for compiling, printing (includes comments)"
            instsize "<Integer> for storage management"
            messagedict "<MessageDict> for communication, compiling"
            monitors "<Dictionary/nil> compiler checks here"
            superclass "<Class> for execution of inherited behavior"
            environment "<Vector of SymbolTables> for external references"
            fieldtype'; "<Integer> encodes field size, if bits".
    veryspecial: 1~

"order of messages, just to make things simpler:
    title insystem subclassof fields/abstract (required)
    sharing bytesize veryspecial (any order)"
Class understands: 'title: title
    [self title: title insystem: Smalltalk]'!
Class understands: 'title: title insystem: system
    [system define: title unique as: self.
    superclass ← Object]'!
Class understands: 'subclassof: superclass'!
Class understands: 'abstract
    [self fields: nullString]'!
Class understands: 'fields: myinstvars    "list of instance variables"
    [fieldtype ← 16.
    instsize ← self instvars length.
    instsize>16⇒
        [user notify: "too many instance variables"]
    messagedict ← MessageDict default.
    environment ← Vector new: 1. environment 1 ← Smalltalk]'!
Class understands: 'instvars
    [superclass@nil ⇒ [⇑myinstvars asVector]
    ⇑superclass instvars concat: myinstvars asVector]'!
Class understands: 'understands: code    "install method"
    [user displayoffwhile° [Compiler new compile: code in: self]]'!
Class understands: 'canunderstand: selector
    [⇑messagedict has: selector]'!
Class understands: 'derstands: selector
    [messagedict ← messagedict delete: selector]'!
Class understands: 'install: name method: method literals: literals
        code: code backpointers: backpointers
    [   [messagedict has: name⇒ [CodeKeeper next← messagedict literals: name]].
    messagedict ← messagedict insert: name method: method
        literals: literals code: code backpointers: backpointers]'!
Class understands: 'code: selector
    [⇑messagedict code: selector]'!
Class understands: 'new [] primitive: 28'!"creation of instances"
Class understands: 'printon: strm
    [strm append: title]'!
Class understands: 'allinstances [] primitive: 60'!"enumeration"
    "If Ted cant do it, nobody can..."
Class understands: 'bytesize: n"non-pointer declaration"
    [fieldtype ← 32+ [n=8⇒ [8] 16]]'!"vanilla or chocolate only"
Class understands: 'veryspecial: n    "for ClassClasses"
    [instsize ← instsize+n]'!    "secret freelist fields"
Class understands: 'superclass [⇑superclass]'!
Class understands: 'environment [⇑environment]'!
Class understands: 'print: inst on: strm | ivars i
    [ivars ← myinstvars asVector.
    strm append: "("; append: title; append: " new ".
    for° i to: instsize do°
```

```
        [strm append: ivars'i; append: ": ";
            print: (inst instfield: i); space]
    strm append: ")"]'!
Class understands: 'init    "init and default get propagated to instances"
    [⇑self new init]'!
Class understands: 'default
    [⇑self new default]'!
Class understands: 'copy: inst | t i
    [t ← self new.
    for: i to: instsize do:
        [t instfield: i ← inst instfield: i]
    ⇑t]'!
Class understands: 'recopy: inst | t i
    [t ← self new.
    for: i to: instsize do:
        [t instfield: i ← (inst instfield: i) recopy]
    ⇑t]'!
Class understands: 'ed: selector | c s
    [c← self code: selector. user clearshow: c.
    while: (s← user request: "substitute: ") do:
        [c ← c subst: s for: (user request: "for: ").
        user clearshow: c]
    self understands: c]'!
```

Class new title: 'VariableLengthClass';
subclassof: Class;
fields: '';
veryspecial: 20~

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

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

```
Context understands: 'eval [] primitive: 30'!
Context understands: 'sender: sender receiver: receiver mclass: mclass
        method: method tempframe: tempframe pc: pc stackptr: stackptr'!
Context understands: 'remoteCopy
    [⇑Context new sender: sender receiver: receiver mclass: mclass
        method: method tempframe: tempframe pc: pc+2 stackptr: stackptr]'!
Context understands: 'sender [⇑sender]'!
Context understands: 'sender← sender []'!
Context understands: 'printon: strm
```

```
    [receiver class printon: strm. sender@nil⇒ []
        strm append: "'"; print: sender thisop]'!
Context understands: 'trace | strm a
    [strm ← Stream default. self printon: strm.
    a ← sender. until° a@nil do°
        [strm cr. a printon: strm. a ← a sender]
    ⇑strm contents]'!
Context understands: 'thisop | a
    [a ← method·pc.
    a≥0320⇒ [⇑self litof: a-0320]
    a≥0260⇒ [⇑self specialops·(1+a-0260)]
    ⇑☞something]'!
Context understands: 'litof: a
    [⇑(method word: a+4) asObject]'!
Context understands: 'specialops
    [⇑☞(+ - < > ≤ ≥ = ≠
        * / \ | min: max: land: lor:
        · ⇒("'←" atomize) next ⇒("next←" atomize) length @ nil nil
        class and: or: new new: to: oneToMeAsStream asStream)]'!
Context understands: 'debug | t
    [user cr. self print.
    while° [t ← user request: "
*"] do° [user show: (self'st) asString]]'!
```

"NUMBERS"

```
Class new title: 'Number';
    abstract~    "Numbers in general"
Number understands: 'min: arg
    [self>arg⇒[↑arg]]'!
Number understands: 'max: arg
    [self<arg⇒[↑arg]]'!
Number understands: '@ y
    [↑Point new x: self y: y]'!
Number understands: 'to: x
    [↑Interval new from: self to: x by: 1]'!
Number understands: 'to: x by: y
    [↑Interval new from: self to: x by: y]'!
Number understands: 'subscripts: a
    [↑a·self asInteger]'!
Number understands: 'subscripts: a ← val
    [↑a·self asInteger ← val]'!

Class new title: 'Integer';    "16-bit integers"
    subclassof: Number;
    fields: '';
    bytesize: 16;
    veryspecial: 1~ "instance state not currently accessible"
Integer understands: '+ arg
    [↑self + arg asInteger] primitive: 6'!
Integer understands: '- arg
    [↑self - arg asInteger] primitive: 7'!
Integer understands: '* arg
    [↑self * arg asInteger] primitive: 21'!
Integer understands: '/ arg
    [↑self / arg asInteger] primitive: 22'!
Integer understands: '< arg
    [↑self < arg asInteger] primitive: 8'!
Integer understands: '= arg
    [↑self = arg asInteger] primitive: 9'!
Integer understands: '> arg
    [↑self > arg asInteger] primitive: 10'!
Integer understands: '≤ arg
    [↑self ≤ arg asInteger] primitive: 11'!
Integer understands: '≠ arg
    [↑self ≠ arg asInteger] primitive: 12'!
Integer understands: '≥ arg
    [↑self ≥ arg asInteger] primitive: 13'!
Integer understands: 'lshift: arg
    [↑self lshift: arg asInteger] primitive: 25'!
Integer understands: 'land: arg
    [↑self land: arg asInteger] primitive: 23'!
Integer understands: 'lor: arg
    [↑self lor: arg asInteger] primitive: 24'!
Integer understands: 'xor: arg
    [↑self xor: arg asInteger] primitive: 35'!
Integer understands: 'field: fld
    [↑self field: fld asInteger] primitive: 36'!
Integer understands: 'field: fld ← val
    [↑self field: fld asInteger ← val asInteger] primitive: 37'!
Integer understands: '\ arg"mod"
    [↑self \ arg asInteger] primitive: 26'!
Integer understands: '| arg"truncate"
    [↑self/arg*arg]'!
Integer understands: 'printon: strm
    [self<0⇒[strm append: "-". (0-self) absprinton: strm]
    self absprinton: strm]'!
Integer understands: 'absprinton: strm | rem
    [rem ← self\10.
    [self>9⇒ [self/10 absprinton: strm]].
    strm next ← rem+060]'!
```

```
Integer understands: 'asInteger [↑self]'!
Integer understands: 'asFloat [] primitive: 34'!
Integer understands: 'oneToMeAsStream      "used by for-loops"
    [↑Stream new of: (Interval new from: 1 to: self by: 1)]'!
Integer understands: 'copy [↑self]'!
Integer understands: 'recopy [↑self]'!
Integer understands: 'isletter
    [self ≥ 0141⇒  " a "
        [↑self ≤ 0172] " z "
    self ≥ 0101⇒   " A "
        [↑self ≤ 0132] " Z "
    ↑false]'!
Integer understands: 'isdigit
    [self ≥ 060⇒   " 0 "
        [↑self ≤ 071]  " 9 "
    ↑false]'!

"Following two must be failures from Array subscripting: "
Integer understands: 'subscripts: a
    [user notify: "Subscript out of bounds: " + self asString]'!
Integer understands: 'subscripts: a ← val
    [user notify: "Subscript out of bounds: " + self asString]'!

Integer understands: 'purge [] primitive: 44'!"write this oop to disk"
    "Warning: The Surgeon General has determined that the following message
    may be hazardous to the health of your system."
Integer understands: 'asObject [] primitive: 81'!"makes a pointer"
```

```
Class new title: 'Float';        "Floating-Point"
    subclassof: Number;
    fields: '';
    bytesize: 16;
    veryspecial: 3~ "instance state not currently accessible"
Float understands: '+ arg
    [↑self+arg asFloat] primitive: 67'!
Float understands: '- arg
    [↑self-arg asFloat] primitive: 68'!
Float understands: '* arg
    [↑self*arg asFloat] primitive: 69'!
Float understands: '/ arg
    [↑self/arg asFloat] primitive: 70'!
Float understands: '< arg
    [↑self<arg asFloat] primitive: 71'!
Float understands: '= arg
    [↑self=arg asFloat] primitive: 72'!
Float understands: '≤ arg
    [↑self≤arg asFloat] primitive: 73'!
Float understands: '> arg
    [↑self>arg asFloat] primitive: 74'!
Float understands: '≥ arg
    [↑self≥arg asFloat] primitive: 75'!
Float understands: '≠ arg
    [↑self≠arg asFloat] primitive: 76'!
Float understands: 'fpart [] primitive: 77'!
Float understands: 'ipart
    [↑self-self fpart]'!    "NOTE this isnt an Integer"
Float understands: 'asInteger [] primitive: 78'! "this IS an Integer"
Float understands: 'sqrt [] primitive: 79'!
Float understands: 'ipow: x    "fixed powers in log n steps"
    [x=0⇒ [↑1.0]
    x=1⇒ [↑self]
    x>1⇒ [↑((self*self) ipow: x/2)*(self ipow: x\2)]
    ↑1.0/(self ipow: 0-x)]'!
Float understands: 'epart: base | x"gives floor log.base self"
    [self<base⇒ [↑0]       "self assumed positive"
```

```
    self<(base*base)⇒ [⇑1]
    x ← 2*(self epart: base*base). "binary recursion like ipow"
    ⇑x + ((self/(base ipow: x)) epart: base)]'!
Float understands: 'printon: strm
    [self<0.0⇒ [strm append: "-". (0.0-self) absprinton: strm]
    self absprinton: strm]'!
Float understands: 'absprinton: strm | x y q i fuzz
    [fuzz ← 5.0e-9.          "fuzz tracks significance"
    y ← [self<1.0⇒ [0-(10.0/self epart: 10.0)] self epart: 10.0].
    x ← self/(10.0 ipow: y)+fuzz.          "normalize x"
    [x≥10.0⇒ [y ← y+1. x ← x/10.0]]. "y = exponent"
    [y<6 and: y>-4⇒
        [q ← 0.          "decimal notation"
        y<0⇒ [strm append: "0.0000"·(1 to: 1-y)]
        fuzz ← fuzz * 10.0 ipow: y].
    q ← y. y ← 0].          "scientific notation"
    for¤ i to: 9 do¤
        [strm next ← 060+x ipart.
        x ← 10.0 * x fpart.
        0>(y ← y-1)⇒
            [x<(fuzz ← fuzz*10.0)⇒["done - fix"]
            y=-1⇒ [strm append: "."]]]
    [y=-1⇒[strm append: "-0"]].
    q≠0⇒[strm append: "e"; print: q]]'!
Float understands: 'asFloat'~
Float understands: 'copy [⇑self]'!
Float understands: 'recopy [⇑self]'!
```

"ARRAYS"

```
Class new title: 'Array';
    abstract~                "arrays in general"
Array understands: 'length [] primitive: 16'!
Array understands: '_ x
    [↑x subscripts: self] primitive: 38'!
Array understands: '_ x ← val
    [↑x subscripts: self ← val] primitive: 39'!
"Note that subscripting by an integer is primitive.  A subscript of another class will be
called with the message subscripts: array, in the hopes that it knows how to behave as
a subscript."
Array understands: 'subscripts: x          "subarrays"
    [↑Substring new data: x map: self]'!
Array understands: 'subscripts: x ← val    "subrange replacement"
    [self length≠val length⇒
        [user notify: "lengths not commensurate"]
    val copyto: (Substring new data: x map: self).
    ↑val]'!
Array understands: 'all ← val | i
    [for: i to: self length do:
        [self_i ← val]]'!
Array understands: '= arg | x
    [self length ≠ arg length⇒ [↑false]
    for: x to: self length do:
        [(self_x) = (arg_x)⇒ [] ↑false]
    ↑true]'!
Array understands: 'find: x | i
    [for: i to: self length do:
        [self_i=x⇒ [↑i]].
    ↑0]'!
Array understands: 'findnon: x | i
    [for: i to: self length do:
        [self_i≠x⇒ [↑i]].
    ↑0]'!
Array understands: 'has: x
    [↑0≠(self find: x)]'!
Array understands: 'reverse
    [↑Substring new data: self map: (self length to: 1 by: -1)]'!
Array understands: 'concat: arg | x
    [x ← self species new: self length + arg length.
    x_(1 to: self length) ← self.
    x_(self length+1 to: x length) ← arg.  ↑x]'!
Array understands: 'copy
    [↑self copyto: (self species new: self length)]'!
Array understands: 'copyto: t | i s
    [s ← t asStream.
    for: i from: self do:
        [s next← i]
    ↑t]'!
Array understands: 'replace: a to: b by: s | x
    [x ← self species new: self length+s length -(1+b-a).
    x_(1 to: a-1) ← self_(1 to: a-1).  t←a+s length-1.
    x_(a to: t) ← s.
    x_(t+1 to: x length) ← self_(b+1 to: self length).
    ↑x]'!
Array understands: 'growby: n
    [↑self copyto: (self species new: self length+n)]'!
Array understands: 'grow
    [↑self copyto: (self species new: (4 max: self length*2))]'!
Array understands: 'last
    [↑self_self length]'!
Array understands: 'last ← val
    [↑self_self length ← val]'!
Array understands: 'species
    [↑Vector]'!
Array understands: 'read
    [↑self new asStream read]'!
Array understands: 'asStream
    [↑Stream new of: self]'!
```

```
Array understands: 'isIntervalBy1
    [↑false]'!
Array understands: 'swap: i with: j | t
    [t ← self·i. self·i ← self·j. self·j ← t]'!
```

```
VariableLengthClass new title: 'Vector';      "Array of objects"
    subclassof: Array~
Vector understands: 'x | v
    [v ← self growby: 1."use a stream if youre in a hurry"
    v last ← x. ↑v]'!
Vector understands: 'printon: strm | i
    [strm append: "(".
    for i to: self length-1 do
        [strm print: self·i; append: ", "].
    strm print: self last; append: ")" ]'!
```

```
VariableLengthClass new title: 'String';      "Array of 8-bit bytes"
    subclassof: Array;
    bytesize: 8~
String understands: 'word: x  "read word in String"
    [↑self·(2*x) + (self·(2*x-1) lshift: 8)]'!
String understands: 'word: x ← y  "write word in String"
    [self·(2*x) ← y land: 0377.
    self·(2*x-1) ← y lshift: ¯8. ↑y]'!
String understands: 'printon: strm | x"print inside string quotes"
    [strm next← 047.
    for x from: self do
        [strm next← x.
        x=047⇒[strm next← x]]    "imbedded quotes get doubled"
    strm next← 047]'!
String understands: 'species
    [↑String]'!
String understands: 'asVector
    [↑self asStream asVector]'!
String understands: 'asParagraph
    [↑Paragraph new text: self alignment: 0]'!
String understands: 'recopy
    [↑self copy]'!
String understands: 'subst: repl for: key | key1 i nskip result
    [nskip ← 0. key1 ← key·1. result ← Stream default.
    for i to: self length do        " the Boyer Slow string replacement "
        [nskip>0⇒ [nskip ← nskip-1]
        self·i = key1⇒
            [self·(i to: (self length min: i+key length-1)) = key⇒
                [result append: repl. nskip ← key length-1]
            result next← self·i]
        result next← self·i]
    ↑result contents]'!
String understands: 'unique | u    "copy and intern"
    [u ← UniqueString new: self length.
    ↑u of: self]'!
String understands: 'hash | x h    "not great, but compatible with FT atom hashing"
    [h ← 13131.
    for x from: self do
        [h ← x * h.
        h ← (h lshift: ¯1)+(h lshift: 15)]
    ↑h]'!
```

```
VariableLengthClass new title: 'UniqueString';      "allows fast compare (eq)
for tables"
    subclassof: String;
```

bytesize: 8~

```
UniqueString understands: 'of: s | i a v
    [a ← self intern: s hash: (i←s hash)⇒ [⇑a]
    i ← 1+(i lshift: ¯8).
    v ← USTable·i.
    USTable·i ← Vector new: 2*v length.      "grow that hash bucket"
    for§ a from: v do§      "copy all its contents"
        [a@nil⇒ []
        self intern: a hash: a stringhash]
    ⇑self of: s]'!      "and try again... "
UniqueString understands: 'intern: s hash: h| i j v n
    [v ← USTable·(1+(h lshift: ¯8)).
    for§ i to: v length do§      "interning compatible with FT atoms - change it soon"
        [h ← h\v length+1.
        v·h@nil⇒      "empty slot"
            [s is: UniqueString⇒ [⇑v·h ← s]  "(when growing)"
            n← 0.  for§ j from: v do§
                [j@nil⇒ [n ← n+1]]    "count # empty slots"
            4*n < v length⇒[⇑false]"grow if not 1/4"
            for§ j to: s length do§      "copy string"
                [super·j ← s·j]"where there-s a will there-s a way"
            ⇑v·h ← self]      "and install self as the atom"
        s=(v·h)⇒[⇑v·h]]
    user notify: "USTable jammed (UniqueString)"]'!
UniqueString understands: 'stringhash
    [⇑super hash]'!
UniqueString understands: '·x ← val
    [user notify: "UniqueStrings are not for writing into"]'!
UniqueString understands: 'printon: strm
    [strm append: self]'!
UniqueString understands: 'isinfix    "one-char non-alpha"
    [self length≠1⇒ [⇑false]  ⇑(self·1) isletter@false]'!
UniqueString understands: 'iskeyword | x"ends with open or closed colon"
    [self length≤1⇒ [⇑false]
    x ← self·self length.
    x=072⇒[⇑true] ⇑x=03]'!
UniqueString understands: 'isuneval "ends with open colon"
    [⇑self·self length=03]'!
UniqueString understands: '= x [⇑self@x]'!"pointer compare"
UniqueString understands: 'hash [] primitive: 46'!"just the object pointer"
UniqueString understands: 'unique'!
```

doesn't work for length ≤ 4

Class new title: 'Substring'; "Substrings *and permutations*"
subclassof: Array;
fields: 'data map'~

```
Substring understands: 'data: data map: map'!
Substring understands: '· x
    [⇑data·(map·x)]'!
Substring understands: '· x ← val
    [⇑data·(map·x) ← val]'!
Substring understands: 'length
    [⇑map length]'!
Substring understands: 'species
    [⇑data species]'!
Substring understands: 'asStream
    [map isIntervalBy1⇒  "direct stream for simple substrings"
        [⇑Stream new of: data from: map start to: map stop]
    ⇑Stream new of: self from: 1 to: map length]'!
```

Class new title: 'Interval'; "*Intervals - Integer or Float*"
subclassof: Array;

```
    fields: 'start stop step length'~
Interval understands: 'from: start to: stop by: step
    [length ← 1+(stop-start/step)]'!
Interval understands: ': x
    [x<1⇒ [↑nil]
    x>length⇒ [↑nil]
    ↑start+(x-1*step)]'!
Interval understands: ': x ← val
    [user notify: "Intervals are not for writing into"]'!
Interval understands: 'length [↑length]'!
Interval understands: 'isIntervalBy1
    [↑step=1]'!
Interval understands: 'start [↑start]'!
Interval understands: 'stop [↑stop]'!

Class new title: 'CoreLocs';        "Proceed at your own risk..."
    subclassof: Array;
    fields: 'base length'~
CoreLocs understands: 'base: base length: length'!
CoreLocs understands: ': x [] primitive: 42'!"contents of memory location x"
CoreLocs understands: ': x ← val [] primitive: 43'!"stores into memory location x"
CoreLocs understands: 'length [↑length]'!
```

"STREAMS"

```
Class new title: 'Stream';
    fields: 'array position limit'~
Stream understands: 'of: array
    [position ← 0. limit ← array length]'!
Stream understands: 'of: array from: position to: limit
    [position ← position-1]'!
Stream understands: 'default
    [self of: (String new: 8)]'!
Stream understands: 'next"simple result"
    [self myend⇒ [↑self pastend]
    ↑array·(position ← position+1)] primitive: 17'!
Stream understands: 'next ← x     "simple arg"
    [self myend⇒ [↑self pastend ← x]
    ↑array·(position ← position+1) ← x] primitive: 18'!
Stream understands: 'append: x | i"Array arg"
    [for° i from: x do°
        [self next ← i].
    ↑x]'!
Stream understands: 'myend
    [↑position≥limit]'!
Stream understands: 'pastend
    [↑false]'!
Stream understands: 'pastend ← x
    [array ← array grow. limit ← array length.
    ↑self next ← x]'!
Stream understands: 'into: x | i   "Array result"
    [for° i to: x length do°
        [x·i ← self next].
    ↑x]'!
Stream understands: 'contents
    [↑(array·(1 to: position)) copy]'!
Stream understands: 'skip: x
    [position ← position+x]'!
Stream understands: 'reset
    [position ← 0]'!
Stream understands: 'end
    [↑position≥limit]'!
Stream understands: 'position
    [↑position]'!
Stream understands: 'loc  "synonym for compiler"
    [↑position]'!
Stream understands: 'empty
    [↑position=0]'!
Stream understands: 'peek | x
    [x← self next⇒ [position ← position-1. ↑x]   "peek at next element"
    ↑false]'!
Stream understands: '≮ x | y
    [y← self next⇒      "peek for matching element"
        [x=y⇒ [↑y]      "gobble it if found"
        position ← position-1. ↑false]
    ↑false]'!
Stream understands: 'pop      "use it as a LIFO"
    [position<1⇒ [↑false]
    position ← position-1. ↑array·(position+1)]'!
Stream understands: 'pop: n | t
    [position<n⇒ [↑false]
    t ← self last: n.
    position ← position-n. ↑t]'!
Stream understands: 'last
    [↑array·position]'!
Stream understands: 'last: n
    [↑(array·(position-n+1 to: position)) copy]'!
Stream understands: 'dequeue      "use it as a FIFO"
    [↑self dequeue: 1]'!
Stream understands: 'dequeue: n | t
    [position<n⇒ [↑false]
```

```
    t ← (array'(1 to: n)) copy.
    array'(1 to: position-n) ← array'(n+1 to: position).
    position ← position-n. ⇑t]'!
Stream understands: 'upto: x | y s
    [s ← Stream default.
    until: [x = (y ← self next)] do:
        [s next ← y].
    self skip: -1. ⇑s contents]'!
Stream understands: '_ x ← val
    [⇑array'x ← val]'!
Stream understands: '_ x
    [⇑array'x]'!
Stream understands: 'space
    [self next ← 040]'!
Stream understands: 'tab
    [self next ← 011]'!
Stream understands: 'cr
    [self next ← 015]'!
Stream understands: 'print: obj
    [obj printon: self]'!
Stream understands: 'asStream'!
Stream understands: 'asVector
    [⇑(Reader new of: self) read]'!
```

"GRAPHICAL OBJECTS"

Class new title: 'Point';
fields: 'x y'~

```
Point understands: 'x: x y: y'!
Point understands: 'x ← x'!
Point understands: 'x [↑x]'!
Point understands: 'y ← y'!
Point understands: 'y [↑y]'!
Point understands: '= pt
    [↑x=pt x and: y=pt y]'!
Point understands: '< pt
    [↑x<pt x and: y<pt y]'!
Point understands: '≤ pt
    [↑x≤pt x and: y≤pt y]'!
Point understands: '+ pt
    [↑Point new x: x+pt x y: y+pt y]'!
Point understands: '- pt
    [↑Point new x: x-pt x y: y-pt y]'!
Point understands: '* scale
    [↑Point new x: x*scale y: y*scale]'!
Point understands: '/ scale
    [↑Point new x: x/scale y: y/scale]'!
Point understands: '| grid
    [↑Point new x: x|grid y: y|grid]'!
Point understands: 'max: t
    [↑Point new x: (x max: t x) y: (y max: t y)]'!
Point understands: 'min: t
    [↑Point new x: (x min: t x) y: (y min: t y)]'!
Point understands: 'rect: p    "infix creation of rectangles"
    [↑Rectangle new origin: self corner: p]'!


Smalltalk insertall: ☞(black white gray ltgray dkgray backround
        storing oring xoring erasing)
    with: ☞(0177777 0 055132 0101202 076575 055132
        0 1 2 3).!
```

Class new title: 'Rectangle';
fields: 'origin corner'~

```
Rectangle understands: 'origin: origin corner: corner'!
Rectangle understands: 'origin [↑origin]'!
Rectangle understands: 'origin ← origin'!
Rectangle understands: 'corner [↑corner]'!
Rectangle understands: 'corner ← corner'!
Rectangle understands: 'extent
    [↑corner-origin]'!
Rectangle understands: 'extent ← extent
    [corner ← origin+extent. ↑extent]'!
Rectangle understands: 'has: pt
    [↑origin≤pt and: pt<corner]'!
Rectangle understands: 'width
    [↑corner x - origin x]'!
Rectangle understands: 'height
    [↑corner y - origin y]'!
Rectangle understands: 'blt: dest mode: mode [] primitive: 47'!
Rectangle understands: 'bltcomp: dest mode: mode [] primitive: 48'!
Rectangle understands: 'brush: dest mode: mode color: color [] primitive: 49'!
Rectangle understands: 'color: color mode: mode [] primitive: 50'!
Rectangle understands: 'clear: color
    [self color: color mode: storing]'!
Rectangle understands: 'clear    "default is backround"
    [self color: backround mode: storing]'!
Rectangle understands: 'comp
    [self color: black mode: xoring]'!
Rectangle understands: 'dragto: dest | b
    [self blt: dest mode: storing.    "copy to new destination"
    b ← corner+dest-origin.    "and clear non-intersecting source"
    [(origin max: dest) ≤ (corner min: b)⇒
        [[dest x>origin x⇒
```

```
            [(origin rect: dest x@corner y) clear]
        dest x<origin x⇒
            [(b x@origin y rect: corner) clear]].
        [dest y>origin y⇒
            [(origin rect: corner x@dest y) clear]
        dest y<origin y⇒
            [(origin x@b y rect: corner) clear]] ]
    self clear]
    origin ← dest. corner ← b]'!
Rectangle understands: 'inset: p1 and: p2
    [↑origin+p1 rect: corner-p2]'!
Rectangle understands: 'inset: p1
    [↑origin+p1 rect: corner-p1]'!
Rectangle understands: 'moveto: pt
    [corner ← corner+pt-origin. origin←pt]'!
Rectangle understands: 'moveby: pt
    [origin ← origin+pt. corner ← corner+pt]'!
Rectangle understands: 'growto: corner '!
Rectangle understands: 'center
    [↑origin+corner/2]'!
Rectangle understands: 'outline: thick | t
    [t ← (¯1@¯1)*thick.
    (self inset: t) clear: black.  self clear: white]'!
Rectangle understands: 'outline   "default border is two thick"
    [self outline: 2]'!                    "one thick is two thin"
Rectangle understands: 'bitsIntoString | extent str
    [extent ← corner-origin.
    str ← String new: 2 * extent y * (extent x+15/16).
    ↑self  bitsIntoString: str]'!
Rectangle understands: 'bitsIntoString: str [] primitive: 51'!
Rectangle understands: 'bitsFromString: str [] primitive: 52'!
Rectangle understands: 'fromuser
    ["redbugcursor showwhile@ ["
    while@ user anybug do@ [].
    until@ user anybug do@ [origin ← user mp].
    while@ user anybug do@ [corner ← user mp. self comp. self comp].
    "]" ]'!
```

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

```
Turtle understands: 'init
    [ink ← ¯3. pen ← width ← 1. dir ← 270.
    frame ← user screenrect.
    self place: frame center]'!
Turtle understands: 'go: length [] primitive: 53'!
Turtle understands: 'goto: pt [] primitive: 54'!
Turtle understands: 'turn: angle
    [dir ← dir+angle \ 360] primitive: 55'!
Turtle understands: 'put: char font: font [] primitive: 56'!
Turtle understands: 'show: str font: font | x
    [for@ x from: str do@
        [self put: x font: font]]'!
Turtle understands: 'place: pt
    [x ← pt x. y ← pt y. xf ← yf ← 0]'!
Turtle understands: 'width: width'!
Turtle understands: 'pen: pen'!
Turtle understands: 'ink: ink'!
Turtle understands: 'up
    [dir ← 270]'!
Turtle understands: 'pendn
    [pen ← 1]'!
Turtle understands: 'penup
    [pen ← 0]'!
Turtle understands: 'black
    [ink ← ¯3]'!
Turtle understands: 'white
    [ink ← ¯1]'!
Turtle understands: 'xor
    [ink ← ¯2]'!
```

```
Turtle understands: 'home
    [self place: frame center. dir ← 270]'!
Turtle understands: 'erase
    [frame clear: 0]'!
Turtle understands: 'frame: frame'!
```

"TEXT DISPLAY"

```
Class new title: 'Textframe';
    fields: 'frame para style reply1 reply2 window'~
Textframe understands: 'para: para frame: frame
    [window ← frame.
    reply1 ← reply1 ← 0.
    style ← DefaultTextStyle]'!
Textframe understands: 'show [] primitive: 57'!
Textframe understands: 'show: para
    [para ← para asParagraph. self show]'!
Textframe understands: 'charofpt: pt [] primitive: 58'!
Textframe understands: 'charnearpt: pt [] primitive: 58'!"synonym"
Textframe understands: 'selectchar: char
    [self selectchar: char asInteger] primitive: 59'!
Textframe understands: 'ptofchar: char
    [self selectchar: char. ↑reply1]'!
Textframe understands: 'rectofchar: char
    [self selectchar: char. ↑reply1 rect: reply2]'!
Textframe understands: 'comp
    [window comp]'!
Textframe understands: 'lineheight
    [↑style lineheight]'!
Textframe understands: 'scrolln: n
    [↑self charofpt: frame origin+(0 @ (n+1*style lineheight))]'!
Textframe understands: 'lastshown
    [↑reply1]'!
Textframe understands: 'window [↑window]'!
Textframe understands: 'put: para at: pt
    [self put: para at: pt centered: false]'!
Textframe understands: 'put: para centered: pt
    [self put: para at: pt centered: true]'!
Textframe understands: 'put: para at: pt centered: center
    [para ← para asParagraph.
    window← frame← pt rect: 1000*1000.
    self ptofchar: para length+1. "find corner of text"
    window growto: reply2.
    [center⇒ [window moveby: pt-window center]]."center it"
    self show]'!
Textframe understands: 'printon: strm
    [strm append: ''a Textframe'']'!


Class new title: 'Paragraph';
    subclassof: Array;
    fields: 'text runs alignment'~
Paragraph understands: 'text: text'!
Paragraph understands: 'text: text runs: runs'!
Paragraph understands: 'text: text alignment: alignment'!
Paragraph understands: 'text: text runs: runs alignment: alignment'!
Paragraph understands: 'flushleft
    [alignment ← 0]'!
Paragraph understands: 'justify
    [alignment ← 1]'!
Paragraph understands: 'center
    [alignment ← 2]'!
Paragraph understands: 'flushright
    [alignment ← 4]'!
Paragraph understands: 'asParagraph'!
Paragraph understands: 'replace: a to: b by: s      "no run support"
    [text ← text replace: a to: b by: s.
    runs ← nil]'!
Paragraph understands: 'copy: a to: b      "no run support"
    [↑(text*(a to: b)) copy]'!
Paragraph understands: 'asVector [↑text asVector]'!
Paragraph understands: 'length
    [↑text length]'!
Paragraph understands: '*x
    [↑text*x]'!
```

```
Class new title: 'TextStyle';
    fields: 'fonts "<Vector of Strings or Integers> which are the fonts.
            An integer entry has a vertical offset in the high 8 bits, a 1 in
            the 200-bit for descent, and another font number (zero-relative)
            in the bottom 4 bits"
        tabandspace "<Integer> =256*tabwidth + spacewidth"
        maxascent "<Integer> max ascent for this fontset"
        maxdescent "<Integer> max descent for this fontset"
        mode "<Integer> =0 for normal, =4 for white-on-black"
        fontnames "<Vector of Strings> corresponding to the fonts"'~
TextStyle understands: 'fonts [↑fonts]'!
TextStyle understands: 'lineheight
    [↑maxascent+maxdescent]'!
TextStyle understands: 'default
    [self mode: 0; tab: 20; space: 5]'!
TextStyle understands: 'tab: t
    [tabandspace ← tabandspace field: leftbyte ← t]'!
TextStyle understands: 'space: t
    [tabandspace ← tabandspace field: rightbyte ← t]'!
TextStyle understands: 'mode: mode'!
TextStyle understands: 'setfont: n name: str | name f"should update max-a/de-scent"
    [FontDict has: (name← str unique)⇒ [fonts'n ← FontDict'name]
    f ← File new old named: str + ".strike".
    f⇒ [FontDict insert: name with: (fonts'n ← f intostring)]
    user notify: "Font " + str + ".strike not on this disk"]'!
TextStyle understands: 'setoffsetfont: n from: m by: d
    [fonts'n ← m + [d<0⇒ [0200] 0] + (d lshift: 8)]'!

Class new title: 'Dispframe';
    subclassof: Stream;
    fields: 'text'~
Dispframe understands: 'rect: r
    [text ← Textframe new para: nil frame: r.
    self of: (String new: 16). self clear]'!
Dispframe understands: 'show
    [text show: self contents.
    until§ text lastshown≥self position do§
        [self dequeue: (text scrolln: 2).
        text show: self contents]]'!
Dispframe understands: 'ev | t
    [while§ [t ← self request: "
Ω"] do§
        [t@nil⇒ [self print: nil doit; show]"redo"
        self print: nil 'st; show]
    ↑false]'!
Dispframe understands: 'request: s
    [self append: s. ↑self read]'!
Dispframe understands: 'read | n t
    [self show. n ← 0.
    while§ true do§
        [t ← user kbd.
        t=4⇒ [self skip: 0-n; append: "done!"; show. ↑false]"ctl-d for done"
        t=8⇒ [n=0⇒[self show]. self skip: ¯1. n ← n-1.
                user kbck⇒[]. self show]  "backspace"
        t=2⇒ [self skip: ¯1-n. ↑nil]"redo"
        t=30⇒ [t ← self last: n."do-it"
            self next ← 30; cr; show. ↑t]
        t=24⇒[self reset; append: "Ω"; show. n←0]
        self next ← t. n ← n+1.
        user kbck⇒ [] self show]]'!
```

```
Dispframe understands: 'clear
    [self reset. self show]'!
Dispframe understands: 'outline
    [text window outline]'!
Dispframe understands: 'moveto: pt
    [(text window inset: -2@-2) dragto: pt-(-2@-2)]'!
```

"SETS AND DICTIONARIES"

```
Class new title: 'HashSet';
    fields: 'objects'~
HashSet understands: 'default
    [self init: 4]'! "default initial size"
HashSet understands: 'init
    [self init: 4]'! "obsolete"
HashSet understands: 'init: size
    [objects ← Vector new: (size max: 4)]'!
HashSet understands: 'insert: obj | i
    [self findorinsert: obj. ↑obj]'!
HashSet understands: 'has: obj
    [self find: obj⇒ [↑true] ↑false]'!
HashSet understands: 'findorinsert: obj | i    "insert if not found, "
    [i ← self findornil: obj⇒ [objects·i ← obj. ↑i]
    self growto: objects length*2. "may cause table to grow"
    ↑self findorinsert: obj]'!
HashSet understands: 'find: obj | i"!index if found, else false"
    [i ← self findornil: obj⇒
        [objects·i@nil⇒ [↑false] ↑i]
    ↑false]'!
HashSet understands: 'findornil: obj | i loc"!index if found or if room, else false"
    [loc ← obj hash\objects length+1."does this match the ucode"
    objects·loc @ nil⇒ [↑loc]
    objects·loc = obj⇒ [↑loc]  "first probe cheap"
    for° i to: (4 max: objects length / 4) do°
        [loc ← loc\objects length+1.  "better delta later if necess"
        objects·loc @ nil⇒ [↑loc]
        objects·loc = obj⇒ [↑loc]]
    ↑false]'!
HashSet understands: 'delete: obj | i
    [i ← self find: obj⇒
        [objects·i ← nil.    "delete, then rehash"
        ↑self growto: objects length]]'!"returns result of growto"
HashSet understands: 'growto: size | copy i
    [copy ← self class new init: size."create a copy"
    for° i from: self do°
        [copy insert: i]"hash each entry into it"
    objects ← copy objects]'!  "then take on all its state"
HashSet understands: 'objects [↑objects]'!
HashSet understands: 'objects← objects'!
HashSet understands: 'contents | obj strm
    [strm ← (Vector new: objects length) asStream.
    for° obj from: objects do°
        [obj@nil⇒ [] strm next← obj]
    ↑strm contents]'!
HashSet understands: 'asStream
    [↑self contents asStream]'!
HashSet understands: 'notthere: name
    [user notify: name asString+" not found."]'!

Class new title: 'Dictionary';
    subclassof: HashSet;
    fields: 'values'~
"Dictionaries have the same lookup properties as Sets, except that they also associate a
value with each object present."
Dictionary understands: 'init: size
    [values ← Vector new: size. super init: size]'!
Dictionary understands: 'insert: name with: value
    [self insert: name. values·(self find: name) ← value]'!
Dictionary understands: '· name | x
    [x ← self find: name⇒ [↑values·x]
    self notthere: name]'!
```

default+ [↑self-init:8]

```
Dictionary understands: '· name ← value | x
    [x ← self find: name⇒ [⇑values·x ← value]
    self notthere: name]'!
Dictionary understands: 'lookup: name | x
    [x ← self find: name⇒ [⇑values·x] ⇑false]'!
Dictionary understands: 'growto: size | name copy
    [copy ← self class new init: size."create a copy of the new size"
    for: name from: self do:
        [copy insert: name with: self·name] "hash each entry into it"
    self copyfrom: copy]'!"then take on all its state"
Dictionary understands: 'copyfrom: dict
    [self objects ← dict objects copy.
    values ← dict values copy]'!
Dictionary understands: 'values [⇑values]'!
Dictionary understands: 'clean | name"release unreferenced entries"
    [for: name from: self do:      "slick, huh"
        [(self·name) refct = 1 ⇒ [self delete: name]]]'!
Dictionary understands: 'insertall: names with: vals | i  "insert many entries"
    [for: i to: names length do:
        [self insert: names·i with: vals·i]]'!
Dictionary understands: 'insertall: names  "default value is nil"
    [self insertall: names with: (Vector new: names length)]'!
```

Class new title: 'SymbolTable';
subclassof: Dictionary;
fields:"~

"SymbolTables have the same properties as Dictionaries, except that an indirect reference is interposed between the value entries and the actual values. This allows compiled code to point directly at a reference which remains valid although the value changes. Notice that the define message checks in Undefined for unresolved references which the compiler may have placed there previously."

```
SymbolTable understands: 'insert: name with: x
    [super insert: name with: (ObjectReference new value← x)]'!
SymbolTable understands: '· name
    [⇑(super·name) value]'!
SymbolTable understands: '· name ← x
    [⇑(super·name) value ← x]'!
SymbolTable understands: 'ref: name
    [⇑super·name]'!
SymbolTable understands: 'define: name as: x
    [self has: name⇒ [self·name ← x]
    Undeclared has: name⇒
        [super insert: name with: (Undeclared ref: name).
        self·name ← x.
        Undeclared delete: name]
    self insert: name with: x]'!
```

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

"Note that insertion and deletion return the updated dictionary. This is because Smalltalk may be executing out of the dictionary. The final switch to the new dictionary is made in Class with one atomic assignment."

```
MessageDict understands: 'init: size
```

```
    [methods ← Vector new: size.  literals ← Vector new: size.
    code ← Vector new: size.   backpointers ← Vector new: size.
    super init: size]'!
MessageDict understands: 'insert: name method: m literals: l
        code: c backpointers: b | i copy
    [i ← self findornil: name⇒      "if name is already there"
        [methods·i ← m. literals·i ← l. code·i ← c. backpointers·i ← b.
        self objects·i ← name]      "then do it, and return self"
    copy ← self growto: methods length*2.    "Otherwise, copy"
    copy insert: name method: m literals: l
            code: c backpointers: b."and insert"
    ↑copy]'!        "and return the new dict without altering old"
MessageDict understands: 'method: name
    [↑methods·(self find: name)]'!
MessageDict understands: 'literals: name
    [↑literals·(self find: name)]'!
MessageDict understands: 'code: name
    [↑code·(self find: name)]'!
MessageDict understands: 'backpointers: name
    [↑backpointers·(self find: name)]'!
MessageDict understands: 'growto: size | name copy i
    [copy ← self class new init: size."create a copy of the new size"
    for: name from: self do:
        [i ← self find: name.   "hash each entry into it"
        copy insert: name method: methods·i literals: literals·i
                code: code·i backpointers: backpointers·i]
    ↑copy]'!   "copy new parts"
```

"INDIRECT REFERENCES"

```
Class new title: 'ObjectReference'
    fields: 'object'~
ObjectReference understands: 'value [↑object]'!
ObjectReference understands: 'value ← object'!
ObjectReference understands: 'printon: strm
    [strm append: "->"; space; print: object class]'!

Class new title: 'FieldReference'
    fields: 'object offset'~
FieldReference understands: 'value [↑object instfield: offset]'!
FieldReference understands: 'value ← value
    [↑object instfield: offset ← value]'!
```