# SMALLTALK SUBSTRECTOR PACKAGE (STRECTOR = STRING or VECTOR)

Smalltalk has strings and vectors:

```
"sa ← string 20
        creates a string of 20 bytes, initiallized to all octal 177
"sc ← vector 41
        creates a vector of 41 elements, initiallized to all nil
```

Strings and vectors respond to "print", "length", and "[".

```
sa print
"q ← sc[10] + sb length
sa[1] ← sa[5]
```

Print is stopped by a null (octal 0) in a string, or by nil in a vector

The SUBSTRECTOR package extends the capabilities of strings and vectors to include searching, copying, and initiallizing contiguous substrings and subvectors. The package can be called from either Smalltalk or machine code.

When an operation applies either to a string or to a vector, this document will refer to a "strector".

The Smalltalk operations on substrectors are:

```
sscan sa[1 to 10] ← all 32
        Initiallize sa[1]...sa[10] all to blanks (decimal 32 = octal 40)

sscan sa[1 to 10] ← sb[5 to 11]
        Copy sa[1]←sb[5], sa[2]←sb[6],... sa[7]←sb[11]
        This works right for copies within the same strector as well

sscan sb[3 to 7]
        Create a new strector 5 long initiallized to sb[3],...sb[7]

sscan sa[1 to 10] find last 32
        Finds the largest i from 1 to 10 such that sa[i]=32

sscan sa[1 to 10] find last non 32
        Finds the largest i from 1 to 10 such that sa[i] not equal 32

sscan sa[1 to 10] find first 32
sscan sa[1 to 10] find 32
        Finds the smallest i from 1 to 10 such that sa[i]=32

sscan sa[1 to 10] find first non 32
sscan sa[1 to 10] find non 32
        Finds the smallest i from 1 to 10 such that sa[i] not equal 32
```

SSCAN "confines bounds" as follows: s[i to j] is interpreted as if it were s[i max 1 to j min s length].

The word "sscan" must appear, because classes string and vector do not presently know about substrectors. At some later date, the word "sscan" will be unnecessary.





Here is the Smalltalk definition of "sscan". It is necessary that your COMM@.SR have code #0 defined as SSCOD (extal), and that you load SSCAN.RB with your Smalltalk. SSCAN.SR is available from Larry, Dan, or Diana.

```
to sscan op byte s lb ub s2 lb2 ub2 (
"lb2 ← "ub2 ← 1.
:#s. %[. :lb. %to. :ub. %].
%find? ("op ← (%first?(1) %last?(2) 1) + (%non?(2) 0). :byte. @CODE #0)
%←?  (%all? (:byte. "op+0. @CODE #0)
        :#s2.%[:lb2.%to.:ub2.%]. "byte ← 0. "op ← 5. @CODE #0)
"op ← 6. "byte ← 0. "ub2 ← ub+1-lb.
"s2 ← (s is string?(string ub2)vector ub2).  @CODE #0)
```

To use the package from machine language, load SSCAN.RB and call any of the following routines (no bounds checking, except by "bound"):

- sscan — Equivalent to Smalltalk sscan, but calling sequence different.
- bound — Confine the bounds of a substrector S to the range 1 : S length
- smf — Substrector Map Fetch. Search until a predicate is true.
- sms — Substrector Map Store. Store into a substrector.
- gpb,gtb,gnb — Get previous,this,next byte of a string
- ppb,ptb,pnb — Put previous,this,next byte of a string
- gpw,gtw,gnw — Get previous,this,next word of a vector
- ppw,ptw,pnw — Put previous,this,next word of a vector
- garg — Get arguments from an activation record

```
;       SMALLTALK SUBSTRECTOR PACKAGE (STRECTOR = STRING or VECTOR)

;       SSCAN creates, inits, searches, and copies substrectors
;               jsr @.sscn
;               Address TBL of argument table (see below)
;       Simple return if result in ac0 is integer (ops 0-5)
;       Skips 1 if result in ac0 is a strector (ops 6-7)
;       TBL:
;               CLASS, 1 for string or 2 for vector
;               OP, a Nova integer
;               BYTE, a Nova integer if substring, a pointer if subvector
;                       ...op 0 only
;               S1, a strector
;               LB1, a Nova integer     ... lower bound
;               UB1, a Nova integer     ... upper bound
;               S2, a strector          ... only for ops 6-7
;               LB2, a Nova integer     ... only for ops 6-7
;               UB2, a Nova integer     ... only for ops 6-7
;       OPeration codes are:
;               0       S1[LB1 to UB1] ← all BYTE
;               1       S1[LB1 to UB1] find first BYTE
```

```
;               2       S1[LB1 to UB1] find last BYTE
;               3       S1[LB1 to UB1] find first non BYTE
;               4       S1[LB1 to UB1] find last non BYTE
;               5       S1[LB1 to UB1] ← S2[LB2 to UB2]
;               6       S2[LB2 to UB2] ← S1[LB1 to UB1]

;       SMF Substrector Map Fetch
;               jsr @.smf
;               dir (+=forward,-=backward;1=string,2=vector)
;               ac0 = mapf (map function):
;                 ; Takes str[i] in ac0
;                 ; i in ac1,ac2 transparent
;                 ; May skip-return:
;                 ;     Ifso: SMF skip-returns.
;                 ; If substrector exhausted, no skip-return:
;               ac1 =ptr to [strector ptr, lower bnd, upper bnd]
;               ac2 = transparent

;       SMS Substrector Map Store
;               jsr @.sms
;               dir (+=forward,-=backward;1=string,2=vector)
;               ac0 = mapf (map function):
;                       ;  Takes i in ac1, ac2 transparent
;                       ;  Returns X in ac0
;                       ;  SMS stores X into str[i]
;               ac1 =ptr to [strector ptr, lower bnd, upper bnd]
;               ac2 = transparent

;       Get/Put This/Next/Previous Byte/Word
;                       (GTB, PPW, etc.).
;       Arguments are inline:
;               RIPT    Subscript to be incd/decd/used
;               STR     Strector to be scanned
;       AC2 PRESERVED THROUGHOUT !!

;       GARG: Get ARGuments from activation
;         jsr @.garg
;         no. of first argument
;         Number of arguments
;         .rdx 2
;         10..01; nth bit from right = 1 if nth arg integer
;         .rdx 8
;         arg1  ; address of argument table
;                 . . . . .
;         arg1: 0         ; argument table
;         ...
;         argn: 0


;       jsr bound
;       pointer to [strector,lb,ub]
;               lb←lb max 1, ub ← ub min length(strector)
;               ac1 and ac2 are preserved
```





```
printing 0

" ( sscan DK0:sscan  DP0:sscan  DP1:sscan )

to sscan op byte s lb ub s2 lb2 ub2 (
"lb2 ← "ub2 ← 1.
:#s. %[. :lb. %to. :ub. %].
%find? ("op ← (%first?(1) %last?(2) 1) + (%non?(2) 0). :byte. @CODE 50)
%←7  (%all? (:byte. "op←0. @CODE 50)
        :#s2.%[:lb2.%to.:ub2.%]. "byte ← 0. "op ← 5. @CODE 50)
"op ← 6. "byte ← 0. "ub2 ← ub+1-lb.
"s2 ← (s is string?(string ub2)vector ub2).  @CODE 50)

printing 1

"sa ← string 20

"sb ← string 20
for i=1 to 20 do (sb[i]←TTY)
AAA BBB CCC DDD EEE

sscan sa[1 to 10] ← all 32

sscan sa[1 to 10] ← sb[5 to 14]

sscan sb[3 to 7]

sscan sa[1 to 10] find last 32

sscan sa[1 to 10] find last non 32

sscan sa[1 to 10] find first 32

sscan sa[1 to 10] find first non 32

sscan sa[1 to 10] find 32

sscan sa[1 to 10] find non 32


"sa ← vector 20

"sb ←"(1 B 3 4 A B C D 1 B 3 4)

sscan sa[1 to 10] ← all "B

sscan sa[1 to 20] ← sb[5 to 14]

sscan sb[3 to 7]

sscan sa[1 to 10] find last "B

sscan sa[1 to 10] find last non "B

sscan sa[1 to 10] find first "B

sscan sa[1 to 10] find first non "B

sscan sa[1 to 10] find "B
```

```
;               2       S1[LB1 to UB1] find last BYTE
;               3       S1[LB1 to UB1] find first non BYTE
;               4       S1[LB1 to UB1] find last non BYTE
;               5       S1[LB1 to UB1] ← S2[LB2 to UB2]
;               6       S2[LB2 to UB2] ← S1[LB1 to UB1]

;       SMF Substrector Map Fetch
;               jsr @.smf
;               dir (+=forward,-=backward;1=string,2=vector)
;               ac0 = mapf (map function):
;                 ; Takes str[i] in ac0
;                 ; i in ac1,ac2 transparent
;                 ; May skip-return:
;                 ;     Ifso: SMF skip-returns.
;                 ; If substrector exhausted, no skip-return:
;               ac1 =ptr to [strector ptr, lower bnd, upper bnd]
;               ac2 = transparent

;       SMS Substrector Map Store
;               jsr @.sms
;               dir (+=forward,-=backward;1=string,2=vector)
;               ac0 = mapf (map function):
;                       ;  Takes i in ac1, ac2 transparent
;                       ;  Returns X in ac0
;                       ;  SMS stores X into str[i]
;               ac1 =ptr to [strector ptr, lower bnd, upper bnd]
;               ac2 = transparent

;       Get/Put This/Next/Previous Byte/Word
;                       (GTB, PPW, etc.).
;       Arguments are inline:
;               RIPT    Subscript to be incd/decd/used
;               STR     Strector to be scanned
;       AC2 PRESERVED THROUGHOUT !!

;       GARG: Get ARGuments from activation
;         jsr @.garg
;         no. of first argument
;         Number of arguments
;         .rdx 2
;         10.,01; nth bit from right = 1 if nth arg integer
;         .rdx 8
;         arg1  ; address of argument table
;         . . . .
;         arg1: 0       ; argument table
;         ...
;         argn: 0


;       jsr bound
;       pointer to [strector,lb,ub]
;               lb←lb max 1, ub ← ub min length(strector)
;               ac1 and ac2 are preserved
```





```
printing 0

" ( sscan DK0:sscan  DP0:sscan  DP1:sscan )

to sscan op byte s lb ub s2 lb2 ub2 (
"lb2 ← "ub2 ← 1.
:#s. %[. :lb. %to. :ub. %].
%find? ("op ← (%first?(1) %last?(2) 1) + (%non?(2) 0). :byte. @CODE 50)
%←?  (%all? (:byte. "op←0. @CODE 50)
       :#s2.%[:lb2.%to.:ub2.%]. "byte ← 0. "op ← 5. @CODE 50)
"op ← 6. "byte ← 0. "ub2 ← ub+1-lb.
"s2 ← (s is string?(string ub2)vector ub2).  @CODE 50)

printing 1

"sa ← string 20

"sb ← string 20
for i=1 to 20 do (sb[i]←TTY)
AAA BBB CCC DDD EEE

sscan sa[1 to 10] ← all 32

sscan sa[1 to 10] ← sb[5 to 14]

sscan sb[3 to 7]

sscan sa[1 to 10] find last 32

sscan sa[1 to 10] find last non 32

sscan sa[1 to 10] find first 32

sscan sa[1 to 10] find first non 32

sscan sa[1 to 10] find 32

sscan sa[1 to 10] find non 32


"sa ← vector 20

"sb ←"(1 B 3 4 A B C D 1 B 3 4)

sscan sa[1 to 10] ← all "B

sscan sa[1 to 20] ← sb[5 to 14]

sscan sb[3 to 7]

sscan sa[1 to 10] find last "B

sscan sa[1 to 10] find last non "B

sscan sa[1 to 10] find first "B

sscan sa[1 to 10] find first non "B

sscan sa[1 to 10] find "B
```

SMALLTALK SUBSTRECTOR PACKAGE (STRECTOR = STRING or VECTOR)

Smalltalk has strings and vectors:

```
"sa ← string 20
        creates a string of 20 bytes, initiallized to all octal 177
"sc ← vector 41
        creates a vector of 41 elements, initiallized ← all nil
```

Strings and vectors respond to "print", "length", and "[".

```
sa print
"q ← sc[10] + sb length
sa[1] ← sa[5]
```

Print is stopped by a null (octal 0) in a string, or by nil in a vector

The SUBSTRECTOR package extends the capabilities of strings and vectors to include searching, copying, and initiallizing contiguous substrings and subvectors. The package can be called from either Smalltalk or machine code.

When an operation applies either to a string or to a vector, this document will refer to a "strector".

The Smalltalk operations on substrectors are:

```
sscan sa[1 to 10] ← all 32
        Initiallize sa[1]...sa[10] all to blanks (decimal 32 = octal 40)

sscan sa[1 to 10] ← sb[5 to 11]
        Copy sa[1]←sb[5], sa[2]←sb[6],... sa[7]←sb[11]
        This works right for copies within the same strector as well

sscan sb[3 to 7]
        Create a new strector 5 long initiallized to sb[3],...sb[7]

sscan sa[1 to 10] find last 32
        Finds the largest i from 1 to 10 such that sa[i]=32

sscan sa[1 to 10] find last non 32
        Finds the largest i from 1 to 10 such that sa[i] not equal 32

sscan sa[1 to 10] find first 32
sscan sa[1 to 10] find 32
        Finds the smallest i from 1 to 10 such that sa[i]=32

sscan sa[1 to 10] find first non 32
sscan sa[1 to 10] find non 32
        Finds the smallest i from 1 to 10 such that sa[i] not equal 32
```

SSCAN "confines bounds" as follows: s[i to j] is interpreted as if it were s[1 max i to j min s length].

The word "sscan" must appear, because classes string and vector do not presently know about substrectors. At some later date, the word "sscan" will be unnecessary.





Here is the Smalltalk definition of "sscan". It is necessary that your COMMA.SR have code 50 defined as SSCOD (octal), and that you load SSCAN.RB with your Smalltalk. SSCAN.SR is available from Larry, Dan, or Diana.

```
to sscan op byte s lb ub s2 lb2 ub2 (
"lb2 ← "ub2 ← 1.
:#s. %[. :lb. %to. :ub. %].
%find? ("op ← (%first?(1) %last?(2) 1) + (%non?(2) 0). :byte. @CODE 50)
%←7  (%all? (:byte. "op←0. @CODE 50)
       :#s2.%[:lb2.%to.:ub2.%]. "byte ← 0. "op ← 5. @CODE 50)
"op ← 6. "byte ← 0. "ub2 ← ub+1-lb.
"s2 ← (s is string?(string ub2)vector ub2).  @CODE 50)
```

To use the package from machine language, load SSCAN.RB and call any of the following routines (no bounds checking, except by "bound"):

- **sscan** — Equivalent to Smalltalk sscan, but calling sequence different.
- **bound** — Confine the bounds of a substrector S to the range 1 : S length
- **smf** — Substrector Map Fetch. Search until a predicate is true.
- **sms** — Substrector Map Store. Store into a substrector.
- **gpb,gtb,gnb** — Get previous,this,next byte of a string
- **ppb,ptb,pnb** — Put previous,this,next byte of a string
- **gpw,gtw,gnw** — Get previous,this,next word of a vector
- **ppw,ptw,pnw** — Put previous,this,next word of a vector
- **garg** — Get arguments from an activation record

```
;       SMALLTALK SUBSTRECTOR PACKAGE (STRECTOR = STRING or VECTOR)

;       SSCAN creates, inits, searches, and copies substrectors
;               jsr @.sscn
;               Address TBL of argument table (see below)
;       Simple return if result in ac0 is integer (ops 0-5)
;       Skips 1 if result in ac0 is a strector (ops 6-7)
;       TBL:
;               CLASS, 1 for string or 2 for vector
;               OP, a Nova integer
;               BYTE, a Nova integer if substring, a pointer if subvector
;                       ...op 0 only
;               S1, a strector
;               LB1, a Nova integer     ... lower bound
;               UB1, a Nova integer     ... upper bound
;               S2, a strector          ... only for ops 6-7
;               LB2, a Nova integer     ... only for ops 6-7
;               UB2, a Nova integer     ... only for ops 6-7
;       OPeration codes are:
;               0       S1[LB1 to UB1] ← all BYTE
;               1       S1[LB1 to UB1] find first BYTE
```