Xtreams

Martin Kobetic
Cincom Smalltalk Development

ESUG Barcelona
September 2010

Why?

* more consistency
* de-emphasize positionability
* strict separation between read and write streams
* scalability
* transformations / stacking

Basics

Terminal: collection, socket, file, ...

Read Stream: source, get, read:, ...

Write Stream: destination, put:, write:, ...

Transform: read or write
- collecting:, selecting:, ...
- character encoding, compression, marshaling, ...
- substreams & slicing

ReadStream

get
read:
read:into:
read:into:at:
rest
source

next
next:
next:into:startingAt: 1
next:into:startingAt:
upToEnd

(iteration)
do:, collect:, select:, ...

ReadStream - Examples

```
input := ’hello world’ reading.
input get.
input read: 4.
input rest.
input get.

current := ObjectMemory someObject.
[    current := ObjectMemory nextObjectAfter: current.
    current == 0 ifTrue: [Incomplete zero raise].
    current
] reading
    inject: 0 into: [ :c :e | c + 1 ]
```

WriteStream

put:
write:
write:from:
write:from:at:
destination

(positionables)
insert:
insert:from:
insert:from:at:

nextPut:
nextPutAll:
next:putAll:startingAt: 1
next:putAll:startingAt:

WriteStream - Examples

```
output := String new writing.
output write: 'Hello World'.
output close.
output terminal.

String new writing
    write: 5 from: 'Hello World' reading;
    conclusion.
```

Terminals

Collections:
    (1 to: 100) reading | String new writing

Blocks:
    [ ] reading | [:x |] writing

Files:
    ObjectMemory imageFilename reading

Sockets & Pipes:
    Stdin reading | Stdout writing

Buffers & SharedQueues:
    buffer := RingBuffer new: 4.
    in := buffer writing. out := buffer reading.

CPointers

Terminals - Examples - Blocks

```
a := 0. b := 1.
fib := [ | x | x := a. a := b. b := b + x. x ] reading.
fib ++ 99; get.

point := 20 @ 20.
[ :x | x printString asComposedText
        displayOn: builder window graphicsContext
        at: (point := point + (0@20))
] writing write: 30 from: fib
```

Terminals - Examples - Pipes

```
[ :in :out |
    [    out writing write: 'Hello'; close.
         in reading read: 5
    ] ensure: [ in close. out close ]
] valueWithArguments: UnixPipeAccessor openPair.

Stdout writing write: 'Hello'; flush.
Stdin reading get.
```

Terminals - Examples - CPointers

```
buffer := CIntegerType char malloc: 50.
[    buffer writing
        length: 50;
        write: 'Hello World!'.
    buffer reading
        contentsSpecies: ByteString;
        read: 12
] ensure: [ buffer free ]
```

Transforms

Collection Style:
    collecting:, selecting:, injecting:into:, doing:, ...

Specialized Transforms:
    encoding:, encodingBase64,
    compressing, en/decrypting:key:iv:, hashing:
    interpreting:, marshaling

General Transforms:
    transforming: [:in :out | ...]

Substreams:
    ending:(inclusive:), limiting:

Transforms - Examples - Collection Style

```
random := Random new reading.
random := random collecting: [ :f | (f * 256) floor ].
random contentsSpecies: ByteArray.

sieve := OrderedCollection new.
ones := [ 1 ] reading.
twoAndUp := ones
    injecting: 1
    into: [ :previous :one | previous + one ].
primes := twoAndUp rejecting: [ :i |
    (sieve anySatisfy: [ :p | i \\ p = 0 ])
        ifTrue: [ true ] ifFalse: [ sieve add: i. false ] ].
primes read: 10.
```

Transforms - Examples - Character Encoding

```
input := ’xtreams.cha’ asFilename reading.
input := input encoding: ’utf8’.
input read: 50.
input close.

input := ’xtreams.cha’ asFilename reading.
input contentsSpecies: String.

(#[13 10 10 13] reading encoding: #ascii) rest.
(ByteArray new writing encoding: #ascii)
    cr; conclusion
```

Transforms - Examples - Cryptographic

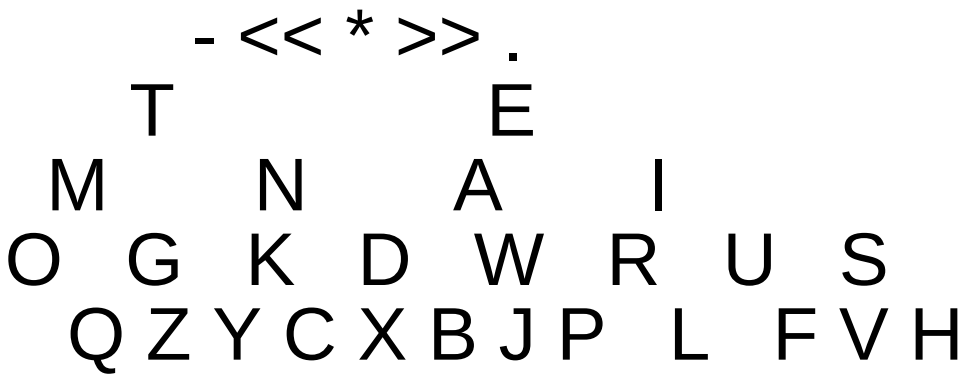
```
(ObjectMemory imageFilename reading hashing: 'md5')
    -= 0; close; digest.

key := random read: 16.
((String new writing
    encodingBase64
    encrypting: 'aes-128-ecb' key: key iv: nil)
    compressing
    encoding: #utf8
)    write: Object comment;
    conclusion.
```

Transforms - Examples - Morse Code

Message

. ... ..- --. -... .- .-. -.-. . .-.. --- -. .- -- -- -..-

Decoding Tree



Transforms - Morse Code Decoding

```
('. ... ..- --.  -... .- .-. -.-. . .-.. --- -. .-  -- -- -..- ' reading
    transforming: [ :in :out || node beep |
        node := MorseTree.
        [    beep := in get.
            beep = $
        ] whileFalse: [
            node := beep = $.
                ifTrue: [ node at: 3 ]
                ifFalse: [ node at: 2 ] ].
        out put: node first ]
) rest
```

Transforms - Morse Code Encoding

```
(String new writing
    transforming: [ :in :out |
        out write: (Morse at: in get);
            put: $  ]
)    write: 'ESUG BARCELONA MMX';
    close;
    terminal
```

Substreams

size
    limiting: 10

bounding criteria
    ending: $a
    ending: [:e | ’abc’ includes: e]
    ending: ’the end’

Substreams - limiting:

```
input := (1 to: 50) reading.
messages := Array new writing.
[     [    message := input limiting: input get.
        messages put: message rest
    ] repeat
] on: Incomplete do: [].
messages conclusion.

output := String new writing.
(output limiting: 40) write: Object comment.
output conclusion.
```

Substreams - ending:

```
(Object comment reading
    ending: $. inclusive: true) rest.
(Object comment reading
    ending: [ :e | '.!?' includes: e ]) rest.
(Object comment reading
    ending: 'Class Variables:') rest.

output := String new writing.
Number withAllSubclasses do: [ :class |
    [    (output ending: $. inclusive: true)
            write: class comment
    ] on: Incomplete do: [].
    output cr ].
output conclusion
```

Substreams - Slicers

```
input := [ 1 ] reading.
slicer := input limiter: 10.
slice := slicer get.
slice rest.

input := ’aaa#bb#c##!1#22#33#444’ reading.
messages := input ender: $!.
parts := messages get ender: $#.
parts collect: [ :p | p rest ].
```

Substreams - Slicers - Writing

```
output := ByteArray new writing.
slicer := output limiter: 3.
1 to: 10 do: [ :i |
    [ slicer get write: (ByteArray new: 10 withAll: i)
    ] on: Incomplete do: [] ].
output conclusion.

output := String new writing.
messages := output closer: [ output put: $! ].
#((aa bb cc dd ee) (xxx yy z)) do: [ :m |
    message := messages get.
    parts := message closer: [ message put: $# ].
    m do: [ :p | parts get write: p ] ].
output conclusion
```

Positioning

Non-positionable streams

++
-= 0

Positionable streams

position / position:
available / length
++ / --
+= / -=
explore:

Positioning - Examples

```
input := ’Hello World!’ reading.
input -= 6; rest.
input += 6; rest.

output := String new writing.
output write: ’Hello World!’.
output += 5; insert: ’, Hello’.
output -= 0; conclusion.
output -= 6; write: ’Earth!’.
output conclusion.
```

Positioning Non-positionable Streams

```
a := 5. b := 10.
input := [
    (a := a + 1) < b ifFalse: [ Incomplete zero raise ].
    a ] reading.
input ++ 2; get.
input -= 0; rest.

input := Random new reading positioning.
input read: 5.
input += 0; read: 10.

input buffer: (RingBuffer on: (Array new: 5)).
```

Positioning Non-positionable Streams

```
output := self newBottomWriter positioning.
output buffer: (RingBuffer on: (String new: 10)).
output write: 'Hello, World!'.
output -= 6; write: 'Earth!'.
output flush
```

Positioning - Exploring

```
separators := '.!?\' withCRs.
lines := Array new writing.
(Object comment reading
    ender: [ :c | separators includes: c ]
) do: [ :s || line |
    line := s positioning.
    (line explore: [
        [ (line read: 6) = 'Object'
        ] on: Incomplete do: [ :ex | false ] ]
    ) ifTrue: [ lines put: line rest ] ].
lines conclusion
```

Main Differences

* strictly separated read and write
* stream composition/stacking
* stream end handling
    * reading/skipping past end => exception
    * no #atEnd -> other patterns #rest, iteration
    [stream atEnd] whileFalse: [...]
    [[...] repeat] on: Incomplete do: [].
* slimmer API, use composition
    * #peek -> #explore: []
    * #upToAll: -> (#ending:) rest

Topics Not Covered

* interpreting
* marshaling
* parsing

Project Structure

Core: defines the API and core classes

Terminals: streams for all supported terminals

Transforms: core transform streams

Substreams: streams embedded in other streams (slicing)

Xtras: non-core transforms

Parsing: PEG parsing

Future Directions

* use => fix bugs, inconsistencies
* ending: aPattern
* xml processing
* backward compatibility layer

References

Project Members:
Michael Lucas-Smith
Martin Kobetic

Project Site:
http://code.google.com/p/xtreams/

Code:
http://www.cincomsmalltalk.com/publicRepository/XtreamsDevelopment(Bundle).html