

Streams

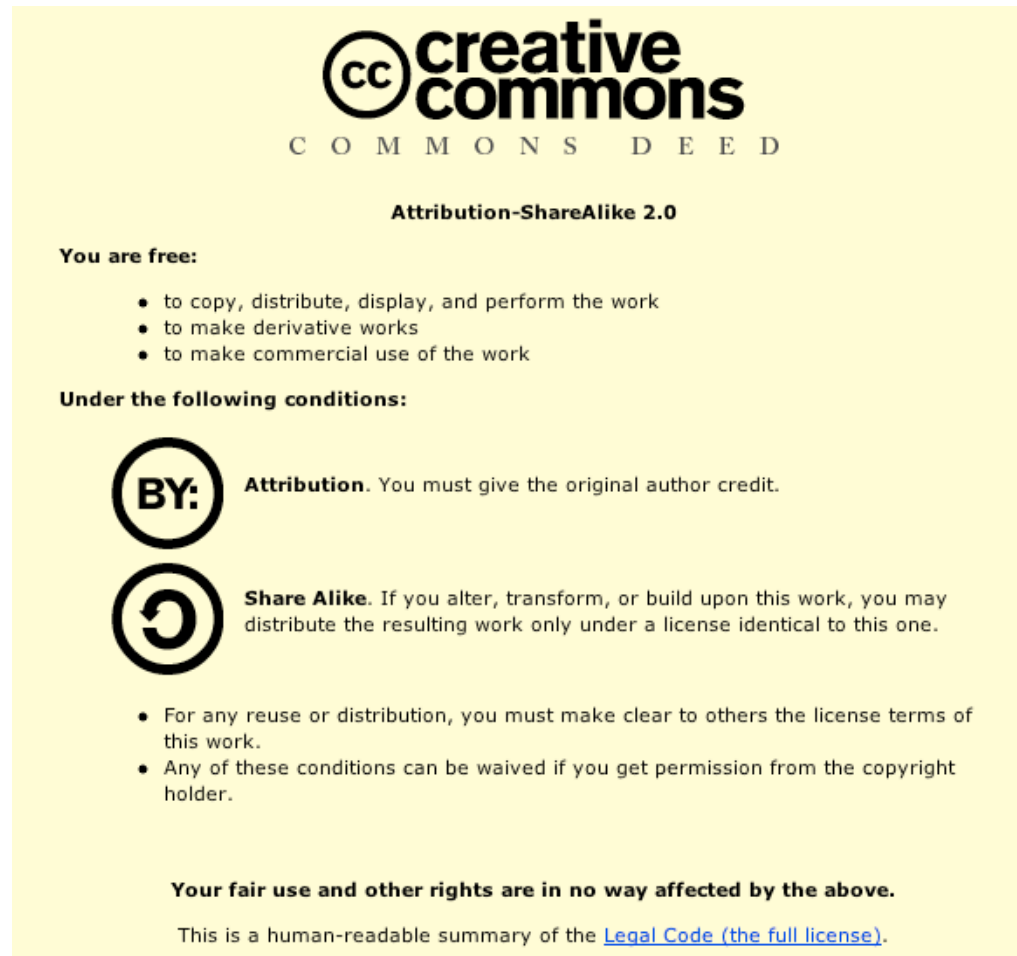
Stéphane Ducasse

Stephane.Ducasse@univ-savoie.fr


<http://www.listic.univ-savoie.fr/~ducasse/>

License: CC-Attribution-ShareAlike 2.0

<http://creativecommons.org/licenses/by-sa/2.0/>



The image shows a summary of the Creative Commons Attribution-ShareAlike 2.0 license. It features the Creative Commons logo at the top, followed by the text 'COMMONS DEED' and 'Attribution-ShareAlike 2.0'. Below this, it lists the freedoms granted under the license: to copy, distribute, display, and perform the work; to make derivative works; and to make commercial use of the work. It then lists the conditions: Attribution (BY) and Share Alike (SA). The Attribution condition requires giving credit to the original author. The Share Alike condition requires that any derivative work be distributed under the same license. Finally, it states that fair use and other rights are not affected by the license and provides a link to the full legal code.


 **creative commons**
COMMONS DEED


Attribution-ShareAlike 2.0

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

 **Attribution.** You must give the original author credit.

 **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the [Legal Code \(the full license\)](http://creativecommons.org/licenses/by-sa/2.0/).

Streams

- Allows the traversal of a collection
- Associated with a collection
 - If the collection is a Smalltalk collection: InternalStream
 - If the collection is a file or an object that behaves like a collection: ExternalStream
- Stores the current position

Stream (abstract)

PeekableStream (abstract)

PositionableStream (abstract)

ExternalStream

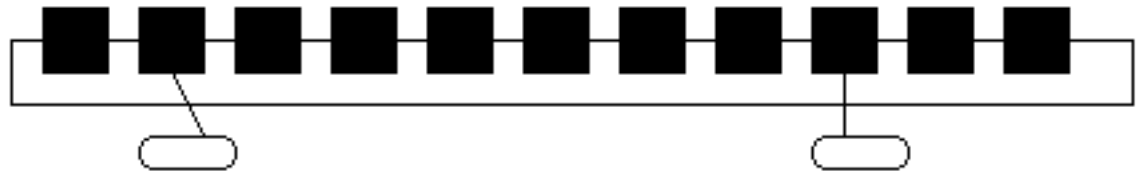
ExternalReadStream

ExternalReadAppendStream

ExternalReadWriteStream

ExternalWriteStream

InternalStream



Example

|st|

st := ReadWriteStream on: (Array new: 6).

st nextPut: 1.

st nextPutAll: (4 8 2 6 7).

st contents. Prlt-> (1 4 8 2 6 7)

st reset.

st next. -> 1

st position: 3.

st next. -> 2

st := (1 2 5 3 7) readStream.

st next. -> 1

printString, printOn:

Object>>printString

"Answer a String whose characters are a description of the receiver."

| aStream |

aStream := WriteStream **on:** (String new: 16).

self printOn: aStream.

^aStream contents

printOn:

```
Node>>printOn: aStream  
    super printOn: aStream.  
    aStream nextPutAll: ' with name: '; print: self name.  
    self hasNextNode  ifTrue: [  
aStream nextPutAll: ' and next node: '; print: self nextNode  
name]
```

Stream Classes

Stream

next returns the next element

next: n returns the n next elements

contents returns all the elements

nextPut: anElement inserts anElement at the next position

nextPutAll: aCollection inserts the collection element from the next position

atEnd returns true if at the end of the collection

Stream Classes (ii)

PeekableStream: Access to the current without passing to the next

peek

skipFor: anArgument

skip: n increases the position of n

skipUpTo: anElement increases the position after anElement

on: aCollection, creates a stream

on: aCol from: firstIndex to: lastIndex (index elements included)

Stream Classes (iii)

PositionableStream

skipToAll: throughAll: upToAll:
position
position: anInteger
reset setToEnd isEmpty

InternalStream

size returns the size of the internal collection

Creation: method with: (without reinitializing the stream)

Stream Tricks

Transcript is a TextCollector that has aStream

```
TextCollector>>show: aString  
  self nextPutAll: aString.  
  self endEntry
```

endEntry via dependencies asks for refreshing the window. If you want to speed up a slow trace, use nextPutAll: + endEntry instead of show:

```
|st sc|  
st := ReadStream on: 'we are the champions'.  
sc := Scanner new on: st.  
[st atEnd] whileFalse: [ Transcript nextPutAll: sc scanToken, ' * '].  
  Transcript endEntry
```

Streams, Blocks, and Files

How to ensure that the open files are closed

```
MyClass>>readFile: aFilename  
    |readStream|  
    readStream := aFilename readStream.  
    [[readStream atEnd] whileFalse: [...]]  
    valueNowOrOnUnwindDo: [readStream close]
```

How to find open files (VW specific)

(ExternalStream classPool at: OpenStreams) copy inspect

Streams, Blocks, and Files (ii)

Filename

appendStream (addition + creation if file doesnot exists)

newReadAppendStream, newReadWriteStream (if
receiver exists, contents removed)

readAppendStream, readWriteStream, readStream,
writeStream

Removing Smalltalk comments from a file

```
|inStream outStream |  
inStream := (Filename named:'/home/ducasse/test.st') readStream.  
outStream := (Filename named:'/home/ducasse/testout.st') writeStream.  
                “(or ‘/home/ducasse/ducasse’ asFilename)”  
[inStream atEnd] whileFalse:  
    [outStream nextPutAll: (inStream upTo: $”)].  
    inStream skipTo: $”].  
^outStream contents
```

“do not forget to close the files too”