

# Package ‘Streamer’

October 17, 2020

**Type** Package

**Title** Enabling stream processing of large files

**Version** 1.34.0

**Author** Martin Morgan, Nishant Gopalakrishnan

**Maintainer** Martin Morgan <martin.morgan@roswellpark.org>

**Description** Large data files can be difficult to work with in R, where data generally resides in memory. This package encourages a style of programming where data is 'streamed' from disk into R via a `producer` and through a series of `consumers` that, typically reduce the original data to a manageable size. The package provides useful Producer and Consumer stream components for operations such as data input, sampling, indexing, and transformation; see `package?Streamer` for details.

**License** Artistic-2.0

**LazyLoad** yes

**Imports** methods, graph, RBGL, parallel, BiocGenerics

**Suggests** RUnit, Rsamtools (>= 1.5.53), GenomicAlignments, Rgraphviz

**biocViews** Infrastructure, DataImport

**Collate** AllGenerics.R Streamer-class.R Producer-class.R  
Consumer-class.R Stream-class.R ConnectionProducer-classes.R  
RawInput-class.R Seq-class.R Downsample-class.R  
FunctionProducerConsumer-classes.R ParallelParam-classes.R  
Team-class.R Team-methods.R Reducer-class.R DAGParam-class.R  
DAGParam-methods.R DAGTeam-class.R Utility-classes.R  
lapply-methods.R stream-methods.R plot-methods.R zzz.R

**git\_url** <https://git.bioconductor.org/packages/Streamer>

**git\_branch** RELEASE\_3\_11

**git\_last\_commit** d38926b

**git\_last\_commit\_date** 2020-04-27

**Date/Publication** 2020-10-16

## R topics documented:

Streamer-package . . . . . 2

ConnectionProducer . . . . .	3
Consumer . . . . .	5
DAGTeam . . . . .	5
Downsample . . . . .	7
Function* . . . . .	8
ParallelParam . . . . .	9
Producer . . . . .	10
RawInput . . . . .	12
Reducer . . . . .	13
reset . . . . .	14
Seq . . . . .	15
status . . . . .	16
Stream . . . . .	17
Team . . . . .	18
Utility . . . . .	20
yield . . . . .	21
<b>Index</b>	<b>22</b>

---

Streamer-package      *Package to enable stream (iterative) processing of large data*

---

## Description

Large data files can be difficult to work with in R, where data generally resides in memory. This package encourages a style of programming where data is 'streamed' from disk into R through a series of components that, typically, reduce the original data to a manageable size. The package provides useful [Producer](#) and [Consumer](#) components for operations such as data input, sampling, indexing, and transformation.

## Details

The central paradigm in this package is a [Stream](#) composed of a [Producer](#) and zero or more [Consumer](#) components. The [Producer](#) is responsible for input of data, e.g., from the file system. A [Consumer](#) accepts data from a [Producer](#) and performs transformations on it. The [Stream](#) function is used to assemble a [Producer](#) and zero or more [Consumer](#) components into a single string.

The [yield](#) function can be applied to a stream to generate one 'chunk' of data. The definition of chunk depends on the stream and its components. A common paradigm repeatedly invokes [yield](#) on a stream, retrieving chunks of the stream for further processing.

## Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

## See Also

[Producer](#), [Consumer](#) are the main types of stream components. Use [Stream](#) to connect components, and [yield](#) to iterate a stream.

**Examples**

```
## About this package
packageDescription("Streamer")

## Existing stream components
getClass("Producer") # Producer classes
getClass("Consumer") # Consumer classes

## An example
fl <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
b <- RawInput(fl, 100L, reader=rawReaderFactory(1e4))
s <- Stream(RawToChar(), Rev(), b)
s
head(yield(s)) # First chunk
close(b)

b <- RawInput(fl, 5000L, verbose=TRUE)
d <- Downsample(sampledSize=50)
s <- Stream(RawToChar(), d, b)
s
s[[2]]

## Processing the first ten chunks of the file
i <- 1
while (10 >= i && 0L != length(chunk <- yield(s)))
{
  cat("chunk", i, "length", length(chunk), "\n")
  i <- i + 1
}
close(b)
```

---

ConnectionProducer      *Producer classes to read file connections*

---

**Description**

ConnectionProducer classes include ScanProducer, ReadLinesProducer, and ReadTableProducer, providing Streamer interfaces to scan, readLines, and read.table.

**Usage**

```
ScanProducer(file, ..., fileArgs=list(), scanArgs=list(...))
ReadLinesProducer(con, ..., conArgs=list(), readLinesArgs=list(...))
ReadTableProducer(file, ..., fileArgs=list(), readTableArgs=list(...))

## S3 method for class 'ConnectionProducer'
close(con, ...)
```

**Arguments**

file, con      The file or connection to be used for input. See [connections](#).  
 ...            Additional arguments, e.g., nlines, to scan, readLines, etc.

fileArgs, conArgs

Arguments, e.g., mode, encoding, to be used when invoking `reset()`.

scanArgs, readLinesArgs, readTableArgs

Arguments to scan, readLines, etc., when reading a file or connection; provide this argument when ... contains arguments (especially `verbose=TRUE`) to be used by the class.

## Methods

See [Producer](#) Methods.

## Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("ReadLinesProducer")$fields`.

Internal methods of this class are described with `getRefClass("ReadLinesProducer")$methods()` and `getRefClass("ReadLinesProducer")$help()`.

## Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

## See Also

[Streamer-package](#), [Producer-class](#), [Streamer-class](#).

## Examples

```
f1 <- system.file(package="Rsamtools", "extdata", "ex1.sam")

p <- ReadLinesProducer(f1, n = 1000) # read 1000 lines at a time
while (length(y <- yield(p)))
  print(length(y))
close(p)

p <- ReadTableProducer(f1, quote="", fill=TRUE, nrows=1000)
while (length(y <- yield(p)))
  print(dim(y))

reset(p)
dim(yield(p))

## connections opened 'under the hood' are closed, with warnings
rm(p); gc()

## avoid warnings by managing connections
p <- ScanProducer(file(f1, "r"), verbose=TRUE,
                 scanArgs=list(what=character()))
length(yield(p))
close(p)
rm(p); gc()
```

---

 Consumer

*Class defining methods for all consumers*


---

### Description

A virtual base class representing components that can consume data from a `Producer`, and yield data to the user or other `Consumer` instances. A `Consumer` typically transforms records from one form to another. `Producer` and `Consumer` instances are associated with each other through the `Stream` function.

### Methods

Methods defined on this class include:

**Stream** Construct a stream from one `Producer` and one or more `Consumer`. See `?Stream`.

### Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("Consumer")$fields`.

Internal methods of this class are described with `getRefClass("Consumer")$methods()` and `getRefClass("Consumer"`

### Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

### See Also

[Streamer-package](#), [Streamer-class](#), [Producer-class](#), [Stream-class](#).

### Examples

```
showClass("Consumer")
```

---

 DAGTeam

*Consumer classes for directed acyclic graph evaluation*


---

### Description

A `Consumer` to route incoming tasks through nodes connected as a directed acyclic graph.

### Usage

```
DAGParam(x, ...)
```

```
DAGTeam(..., dagParam = DAGParam(), teamParam = MulticoreParam(1L))
```

```
## S3 method for class 'DAGTeam'
plot(x, y, ...)
```

## Arguments

x	A matrix or data.frame with columns 'From', 'To', or a graphNEL object (from the graph package) describing a directed acyclic graph.
...	For DAGTeam, named <code>FunctionConsumer</code> instances, one for each node in the graph. The <code>FunctionConsumer</code> corresponding to the first node in the graph must accept one argument; remaining <code>FunctionConsumer</code> instances must have as input arguments the names of the nodes from which the inputs derive, as in the example below. For DAGParam when x is a data.frame or matrix, data.frame columns W, V or additional arguments W, V as described in <code>ftM2graphNEL</code> .
dagParam	A DAGParam instance, with all nodes referenced in the graph represented by <code>FunctionConsumer</code> instances in ...
teamParam	A <code>ParallelParam</code> instance, such as generated by <code>MulticoreParam()</code> . Currently ignored (all calculations are performed on a single thread).
y	Unused.

## Constructors

Use DAGParam and DAGTeam to construct instances of these classes, with ParallelParam instances created by, e.g., MulticoreParam.

## Methods

See `Consumer` Methods.

## Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("MulticoreTeam")$fields`.

Internal methods of this class are described with `getRefClass("MulticoreTeam")$methods()` and `getRefClass("MulticoreTeam")$help()`.

## Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

## See Also

`Team` applies a single function across multiple threads..

## Examples

```
df <- data.frame(From = c("A", "A", "B", "C"),
                To   = c("B", "C", "D", "D"),
                stringsAsFactors=FALSE)
dagParam <- DAGParam(df)
dteam <- DAGTeam(A=FunctionConsumer(function(y) y),
                B=FunctionConsumer(function(A) -A),
                C=FunctionConsumer(function(A) 1 / A),
                D=FunctionConsumer(function(B, C) B + C),
                dagParam=dagParam)

plot(dteam)
```

```
strm <- Stream(Seq(to=10), dteam)
sapply(strm, c)
reset(strm)
```

---

Downsample

*Consumer class to down-sample data*

---

### Description

A [Consumer](#)-class to select records with fixed probability, returning a yield of fixed size. Successive calls to `yield` result in sampling of subsequent records in the stream, until the stream is exhausted.

### Usage

```
Downsample(probability=0.1, sampledSize=1e6, ...)
```

### Arguments

<code>probability</code>	A <code>numeric(1)</code> between 0, 1 indicating the probability with which a record should be retained.
<code>...</code>	Additional arguments, passed to the <code>\$new</code> method of the underlying reference class. Currently unused.
<code>sampledSize</code>	A <code>integer(1)</code> indicating the number of records to return.

### Methods

See [Consumer](#) Methods.

### Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("Downsample")$fields`.

Internal methods of this class are described with `getRefClass("Downsample")$methods()` and `getRefClass("Downsample")$help()`.

### Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

### See Also

[Stream](#)

### Examples

```
showClass("Downsample")
```

Function\*

*Classes for user-defined Producers and Consumers***Description**

The `FunctionProducer` and `FunctionConsumer` classes provide an easy way to quickly create `Producer` and `Consumer` instances from user-provided functions.

**Usage**

```
FunctionProducer(FUN, RESET, ..., state=NULL)
FunctionConsumer(FUN, RESET, ..., state=NULL)
```

**Arguments**

<code>FUN</code>	User defined function to yield successive records in the stream. The <code>FunctionProducer</code> function must return an object of length 0 (e.g., <code>logical(0)</code> ) when the stream is complete.
<code>RESET</code>	An optional function of one argument ('state') to reset the stream to its original state. If missing, the stream cannot be reset.
<code>...</code>	Arguments passed to the <code>Producer</code> -class or <code>Consumer</code> -class constructors.
<code>state</code>	Any information, made available to <code>RESET</code> .

**Constructors**

Use `FunctionProducer` or `FunctionConsumer` to construct instances of this class.

**Methods**

See `Producer` and `Consumer` Methods.

**Internal Class Fields and Methods**

Internal fields of this class are described with, e.g., `getRefClass("FunctionProducer")$fields`.

Internal methods of this class are described with `getRefClass("FunctionProducer")$methods()` and `getRefClass("FunctionProducer")$help()`.

**Author(s)**

Nishant Gopalakrishnan [ngopalak@fhcrc.org](mailto:ngopalak@fhcrc.org)

**See Also**

[Stream](#)



**Examples**

```
## A ProducerFunction
producerFun <- function()
  ## produce the mean of 10 random uniform numbers
  ## stop when the mean is greater than 0.8
  {
    x <- mean(runif(10))
    if (x > .8) numeric(0) else x
  }
randomSampleMeans <- FunctionProducer(producerFun)
result <- sapply(randomSampleMeans, c)
length(result)
head(result)

## A FunctionConsumer:
consumerFun <- function(y)
  ## transform input by -10 log10
  {
    -10 * log10(y)
  }

neg10log10 <- FunctionConsumer(consumerFun)

strm <- Stream(randomSampleMeans, neg10log10)
result <- sapply(strm, c)
length(result)
head(result)
```

---

ParallelParam

*Classes to configure parallel evaluation*


---

**Description**

Configure and register parallel calculations, e.g., for [Team](#) evaluation.

**Usage**

```
MulticoreParam(size = getOption("mc.cores", 2L),
  mc.set.seed = TRUE, ...)
register(param)
```

**Arguments**

size	The number of members in the parallel cluster.
mc.set.seed	logical(1); see ?mcp parallel on unix platforms.
param	A ParallelParam instance, such as generated by MulticoreParam().
...	Additional arguments, e.g., verbose, passed to the Streamer class.

**Constructors**

Use MulticoreParam to construct instances of this class.

**Methods**

**register** Invoked with an argument `param` stores the `param` for use in subsequent parallel computation. Use `NULL` to clear the register. The function returns, invisibly, the previously registered parameter instance, if any.

**Internal Class Fields and Methods**

Internal fields of this class are described with, e.g., `getRefClass("MulticoreParam")$fields`.

Internal methods of this class are described with `getRefClass("MulticoreParam")$methods()` and `getRefClass("MulticoreParam")$help()`.

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Team](#) to apply one function in parallel, [DAGTeam](#) to evaluate functions whose dependencies are represented as directed acyclic graphs.

**Examples**

```
if (.Platform$OS.type != "windows") {
  oparam <- register()      ## previous setting
  param <- MulticoreParam() ## default multicore settings
  register(param)          ## register for future use, e.g., . Team
  register(oparam)        ## reset original
}
```

---

Producer

*Class defining methods for all Producers*

---

**Description**

A virtual class representing components that can read data from connections, and yield records to the user or a Consumer instance. A Producer represents a source of data, responsible for parsing a file or other data source into records to be passed to Consumer classes. Producer and Consumer instances are associated with each other through the [Stream](#) function.

**Usage**

```
## S4 method for signature 'Producer'
lapply(X, FUN, ...)

## S4 method for signature 'Producer'
sapply(X, FUN, ..., simplify=TRUE, USE.NAMES=TRUE)
```

**Arguments**

X	An instance of class <code>Producer</code>
FUN	A function to be applied to each successful <code>yield()</code> of X.
...	Additional arguments to FUN.
<code>simplify</code>	See <code>?base::sapply</code> .
<code>USE.NAMES</code>	See <code>?base::sapply</code> but note that names do not usually make sense for instances of class <code>Producer</code> .

**Methods**

Methods defined on this class include:

**Stream** Construct a stream from one `Producer` and one or more `Consumer`. See `?Stream`.

**yield** Yield a single result (e.g., `data.frame`) from the `Producer`.

**reset** Reset, if possible, the `Producer`.

**lapply, sapply** Apply FUN to each result applied to `yield()`, simplifying (using `simplify2array`) if possible for `sapply`. Partial results on error can be recovered using `tryCatch`, as illustrated below. Infinite producers will of course exhaust memory.

**Internal Class Fields and Methods**

Internal fields of this class are described with, e.g., `getRefClass("Producer")$fields`.

Internal methods of this class are described with `getRefClass("Producer")$methods()` and `getRefClass("Producer"`

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Streamer-package](#), [Consumer-class](#), [Streamer-class](#).

**Examples**

```
showClass("Producer")
showMethods(class="Producer", where="package:Streamer")

sapply(Seq(to=47, yieldSize=7), function(elt) {
  c(n = length(elt), xbar = mean(elt))
})

## recover partial results
fun = function(i) if (i == 5) stop("oops, i == 5") else i
res <- tryCatch(sapply(Seq(to=10), fun), error=function(err) {
  warning(conditionMessage(err),
    "\n only partial results available")
  simplify2array(err$partialResult)
})
res
```

RawInput

*Class "RawInput"***Description**

A **Producer**-class to interpret files as raw (binary) data. Users interact with this class through the constructor `RawInput` and methods `yield`, `reset`, and `Stream`.

This class requires two helper functions; the ‘factory’ methods defined on this page can be used to supply these. `rawReaderFactory` creates a ‘reader’, whose responsibility it is to accept a connection and return a vector of predefined type, e.g., `raw`. `rawParserFactory` creates a ‘parser’, responsible for parsing a buffer and vector of the same type as produced by the reader into records. The final record may be incomplete (e.g., because reader does not return complete records), and regardless of completion status is the content of `buf` on the subsequent invocation of `parser`. `length(buf)` or `length(bin)` may be 0, as when the first or final record is parsed.

**Usage**

```
RawInput(con, yieldSize = 1e+06, reader = rawReaderFactory(),
         parser = rawParserFactory(), ...)
rawReaderFactory(blockSize = 1e+06, what)
rawParserFactory(separator = charToRaw("\n"), trim = separator)
```

**Arguments**

<code>con</code>	A character string or connection (opened as "rb" mode) from which raw input will be retrieved.
<code>yieldSize</code>	The number of records the input parser is to yield.
<code>reader</code>	A function of one argument ( <code>con</code> , an open connection positioned at the start of the file, or at the position the <code>con</code> was in at the end of the previous invocation of the reader function) that returns a vector of type <code>raw</code> .
<code>parser</code>	A function of two arguments ( <code>buf</code> , <code>bin</code> ), parsing the raw vector <code>c(buf, bin)</code> into records.
<code>...</code>	Additional arguments, passed to the <code>\$new</code> method of this class. Currently ignored.
<code>blockSize</code>	The number of bytes to read at one time.
<code>what</code>	The type of data to read, as the argument to <code>readBin</code> .
<code>separator</code>	A raw vector indicating the unique sequence of bytes by which record starts are to be recognized. The parser supplied here includes the record separator at the start of each record.
<code>trim</code>	A raw vector that is a prefix of <code>separator</code> , and that is to be removed from the record during parsing.

**Fields**

`con`: Object of class connection. An R `connection` opened in "rb" mode from which data will be read.

`blockSize`: Object of class integer. Size (e.g., number of raw bytes) input during each `yield`.

`reader`: Object of class function. A function used to input `blockSize` elements. See `rawReaderFactory`.

- `parser`: Object of class function. A function used to parse raw input into records, e.g., breaking a raw vector on new lines `'\n'`. See [rawParserFactory](#)
- `.buffer`: Object of class raw. Contains read but not parsed raw stream data.
- `.records`: Object of class list. Parsed but not yet yield-ed records.
- `.parsedRecords`: Object of class integer. Total number of records parsed by the Producer.

### Class-Based Methods

- `reset()`: Remove buffer and current records, reset record counter, re-open con.

### Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

### See Also

[Stream](#)

### Examples

```
f1 <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
b <- RawInput(f1, 100L, reader=rawReaderFactory(1e4))
length(value <- yield(b))
head(value)
close(b)
```

---

Reducer

*Consumer class to combine successive records*

---

### Description

A [Consumer](#)-class to reduce N successive records into a single yield.

### Usage

```
Reducer(FUN, init, ..., yieldNth = NA_integer_)
```

### Arguments

- |                       |  |
|-----------------------|--|
| <code>FUN</code>      | A function of two arguments, where the first argument is the result of the previous reduction (or <code>init</code> , if specified, for the first record) and the second argument is the current record. |
| <code>init</code>     | An optional initial value to initiate the reduction. When present, <code>init</code> is used to initial each yield.  |
| <code>...</code>      | Additional arguments, passed to the <code>\$new</code> method of the underlying reference class. Currently unused.   |
| <code>yieldNth</code> | A positive integer indicating how many upstream yields are combined before the Reducer yields. A value of <code>NA_integer_</code> indicates reduction of all records in the input stream.               |

**Methods**

See [Consumer](#) Methods.

**Internal Class Fields and Methods**

Internal fields of this class are described with, e.g., `getRefClass("Reducer")$fields`.

Internal methods of this class are described with `getRefClass("Reducer")$methods()` and `getRefClass("Reducer")`

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Stream](#)

**Examples**

```
s <- Stream(Seq(to=10), Reducer("+"))
yield(s)    ## sum(1:10), i.e., Reduce over the entire stream
s <- Stream(Seq(to=10), Reducer("+", yieldNth=5))
yield(s)    ## sum(1:5)
yield(s)    ## sum(6:10)
s <- Stream(Seq(to=10), Reducer("+", init=10, yieldNth=5))
sapply(s, c) ## 10 + c(sum(1:5), sum(6:10))
if (.Platform$OS.type != "windows") {
  s <- Stream(Seq(to=10),
              Team(function(i) { Sys.sleep(1); i },
                    param=MulticoreParam(10L)),
              Reducer("+"))
  system.time(y <- yield(s))
  y
}
```

---

reset

*Function to reset a Stream, Producer, or Consumer*

---

**Description**

`reset` on a stream invokes the `reset` method of all components of the stream; on a component, it invokes the `reset` method of the component and all inputs to the component.

**Usage**

```
reset(x, ...)
```

**Arguments**

`x`                    A Stream, Producer, or Consumer object.  
`...`                  Additional arguments, currently unused.

**Value**

A reference to `x`, the stream or component on which `reset` was invoked.

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Stream](#), [Producer](#), [Consumer](#).

**Examples**

```
## see example(Stream)
```

---

 Seq

*Producer class to generate (numeric) sequences*

---

**Description**

A [Producer](#)-class to generate a sequence (possibly long) of numbers.

**Usage**

```
Seq(from = 1L, to=.Machine$integer.max, by = 1L, yieldSize=1L,
    ...)
```

**Arguments**

<code>from</code>	A starting value of any type (e.g., integer, numeric supported by <code>base::seq</code> ).
<code>to</code>	An ending value, typically of the same type as <code>from</code> .
<code>by</code>	A value, typically of the same class as <code>from</code> , indicating the increment between successive numbers in the sequence. <code>by = 0</code> can create an infinite stream.
<code>yieldSize</code>	A <code>integer(1)</code> indicating the length of the output sequence each time <code>yield()</code> is invoked.
<code>...</code>	Additional arguments passed to <a href="#">Producer</a> .

**Constructors**

Use `Seq` to construct instances of this class.

**Methods**

See [Producer](#) Methods.

**Internal Class Fields and Methods**

Internal fields of this class are described with `getRefClass("Seq")$fields`.

Internal methods of this class are described with `getRefClass("Seq")$methods()` and `getRefClass("Seq")$help()`.

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Stream](#)

**Examples**

```
s <- Seq(1, 10, yieldSize=5)
while(length(y <- yield(s)))
  print(y)
```

---

status

*Function to report current status of a stream*

---

**Description**

status invoked on a stream yields the current status of the stream, as reported by the status methods of each component.

**Usage**

```
status(x, ...)

## S4 method for signature 'Streamer'
status(x, ...)
```

**Arguments**

x                    A Stream, Producer, or Consumer object.  
...                   Additional arguments, currently unused.

**Value**

A component-specific summary the current status

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Stream](#), [Producer](#), [Consumer](#).

**Examples**

```
## see example(Stream)
```



Stream

*Class to represent a Producer and zero or more Consumers***Description**

An ordered collection of Consumer and Producer components combined into a single entity. Applying a method such as `yield` to `Stream` invokes `yield` on the terminal Consumer component of the stream, yielding one batch from the stream. The result of `yield` is defined by the Producer and Consumer components of the stream.

**Usage**

```
Stream(x, ..., verbose=FALSE)

## S4 method for signature 'Stream'
length(x)

## S4 method for signature 'Stream,numeric'
x[[i, j, ...]]

## S4 method for signature 'Stream'
lapply(X, FUN, ...)

## S4 method for signature 'Stream'
sapply(X, FUN, ..., simplify=TRUE, USE.NAMES=TRUE)
```

**Arguments**

<code>x, X</code>	For <code>Stream</code> , <code>x</code> is a <code>Producer</code> instance. For other functions, an instance of class <code>Stream</code> .
<code>FUN</code>	A function to be applied to each successful <code>yield()</code> of <code>X</code> .
<code>i, j</code>	Numeric index of the <code>i</code> th stream element ( <code>j</code> is ignored by this method).
<code>...</code>	For <code>Stream</code> , zero or more <code>Consumer</code> instances. For <code>lapply</code> , <code>sapply</code> , additional arguments to <code>FUN</code> .
<code>simplify</code>	See <code>?base::sapply</code> .
<code>USE.NAMES</code>	See <code>?base::sapply</code> but note that names do not usually make sense for instances of class <code>Producer</code> .
<code>verbose</code>	A <code>logical(1)</code> indicating whether status information should be reported.

**Constructors**

Arguments to `Stream` must consist of a single `Producer` and zero or more `Consumer` components.

When invoked with the `Producer` as the first argument, `Stream(P, C1, C2)` produces a stream in which the data is read by `P`, then processed by `C1`, then processed by `C2`.

When invoked with the `Consumer` as the first argument, the `...` must include a `Producer` as the *last* argument. `Stream(C1, C2, P)` produces a stream in which the data is read by `P`, then processed by `C2`, then processed by `C1`.

## Methods

Methods defined on this class include:

**length** The number of components in this stream.

**[[** The *i*th component (including inputs) of this stream.

**yield** Yield a single result (e.g., `data.frame`) from the stream.

**reset** Reset, if possible, each component of the stream.

**lapply, sapply** Apply FUN to each result applied to `yield()`, simplifying (using `simplify2array`) if possible for `sapply`. Partial results on error can be recovered using `tryCatch`, as illustrated on the help page [Producer](#).

## Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("FunctionProducer")$fields`.

Internal methods of this class are described with `getRefClass("FunctionProducer")$methods()` and `getRefClass("FunctionProducer")$help()`.

## Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

## See Also

[Streamer-package](#), [Consumer-class](#), [Producer-class](#).

## Examples

```
f1 <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
b <- RawInput(f1, 100L, reader=rawReaderFactory(1e4))
s <- Stream(b, Rev(), RawToChar())
s
yield(s)
reset(s)
while (length(yield(s))) cat("tick\n")
close(b)

strm <- Stream(Seq(to=10), FunctionConsumer(function(y) 1/y))
sapply(strm, c)
```

---

Team

*Consumer classes for parallel evaluation*

---

## Description

A [Consumer](#) to divide incoming tasks amongst processes for parallel evaluation; not supported on Windows.

## Usage

```
Team(FUN, ..., param)
```

## Arguments

FUN	A function of one argument (the input to this consumer), to be applied to each element of the stream. The return value of the function is the value yield'ed.
...	Additional arguments (e.g., verbose, passed to the <a href="#">Consumer</a> constructor.
param	If provided, a <code>ParallelParam</code> instance, such as generated by <code>MulticoreParam()</code> .

## Constructors

Use `Team` to construct instances of this class.

When `param` is missing, `Team` consults the registry (see [register](#)) for a parallel parameter class. If none is found and `.Platform$OS.type == "unix"`, a default `MulticoreParam` instance is used. An error is signaled on other operating systems (i.e., Windows)

## Methods

See [Consumer](#) Methods.

## Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("MulticoreTeam")$fields`.

Internal methods of this class are described with `getRefClass("MulticoreTeam")$methods()` and `getRefClass("MulticoreTeam")$help()`.

## Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

## See Also

[ParallelParam](#) for configuring parallel environments. [DAGTeam](#) apply functions organized as a directed acyclic graph.

## Examples

```
if (.Platform$OS.type != "windows") {
  param <- MulticoreParam(size=5)
  team <- Team(function(x) { Sys.sleep(1); mean(x) }, param=param)
  s <- Stream(Seq(to=50, yieldSize=5), team)
  system.time({while(length(y <- yield(s)))
    print(y)
  }) ## about 2 seconds
}
```

---

Utility

*Consumer classes with simple functionality, e.g., RawToChar, Rev*

---

### Description

Utility is a virtual class containing components to create light weight Consumer classes.

RawToChar is a class to convert raw (binary) records to char, applying rawToChar to each record.

Rev reverses the order of current task.

### Usage

```
RawToChar(...)  
Rev(...)
```

### Arguments

... Arguments passed to the [Consumer](#)-class.

### Construction

Use constructors RawToChar, Rev.

### Methods

See [Consumer](#) Methods.

### Internal Class Fields and Methods

Internal fields of this class are described with, e.g., `getRefClass("Utility").$fields`.

Internal methods of this class are described with `getRefClass("Utility").$methods()` and `getRefClass("Utility")`

### Author(s)

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

### See Also

[Streamer-package](#), [Consumer-class](#), [Streamer-class](#).

### Examples

```
showClass("Utility")
```

---

`yield`*Function to yield one task from a Stream or Producer*

---

**Description**

`yield` invoked on a stream yields one chunk of data or, if the stream is complete, a length zero element of the data. Successive invocations of `yield` produce successive chunks of data.

**Usage**

```
yield(x, ...)
```

**Arguments**

<code>x</code>	A Stream, Producer, or Consumer object.
<code>...</code>	Additional arguments, currently unused.

**Value**

A chunk of data, with the specific notion of chunk defined by the final component of the stream.

**Author(s)**

Martin Morgan [mtmorgan@fhcrc.org](mailto:mtmorgan@fhcrc.org)

**See Also**

[Stream](#), [Producer](#), [Consumer](#).

**Examples**

```
## see example(Stream)
```

# Index

## \* classes

- ConnectionProducer, 3
- DAGTeam, 5
- Downsample, 7
- Function\*, 8
- ParallelParam, 9
- Producer, 10
- RawInput, 12
- Reducer, 13
- Seq, 15
- Stream, 17
- Team, 18
- Utility, 20

## \* manip

- status, 16
- yield, 21

## \* methods

- reset, 14

## \* package

- Streamer-package, 2
- [[, Stream, numeric-method (Stream), 17

close.ConnectionProducer  
(ConnectionProducer), 3

connection, 12

ConnectionProducer, 3

ConnectionProducer-class  
(ConnectionProducer), 3

ConnectionProducer-classes  
(ConnectionProducer), 3

connections, 3

Consumer, 2, 5, 5, 6–8, 11, 13–16, 18–21

Consumer-class (Consumer), 5

DAGParam (DAGTeam), 5

DAGParam, data.frame-method (DAGTeam), 5

DAGParam, graphNEL-method (DAGTeam), 5

DAGParam, matrix-method (DAGTeam), 5

DAGParam, missing-method (DAGTeam), 5

DAGParam-class (DAGTeam), 5

DAGTeam, 5, 10, 19

DAGTeam-class (DAGTeam), 5

Downsample, 7

Downsample-class (Downsample), 7

ftM2graphNEL, 6

Function\*, 8

FunctionConsumer, 6

FunctionConsumer (Function\*), 8

FunctionConsumer-class (Function\*), 8

FunctionProducer (Function\*), 8

FunctionProducer-class (Function\*), 8

FunctionProducerConsumer-classes  
(Function\*), 8

lapply, Producer-method (Producer), 10

lapply, Stream-method (Stream), 17

length, Stream-method (Stream), 17

MulticoreParam, 19

MulticoreParam (ParallelParam), 9

MulticoreParam-class (ParallelParam), 9

MulticoreTeam-class (Team), 18

ParallelParam, 9, 19

ParallelParam-class (ParallelParam), 9

ParallelRegister-class (ParallelParam),  
9

plot.DAGParam (DAGTeam), 5

plot.DAGTeam (DAGTeam), 5

Producer, 2, 4, 5, 8, 10, 12, 15, 16, 18, 21

Producer-class (Producer), 10

RawInput, 12, 12

RawInput-class (RawInput), 12

rawParserFactory, 13

rawParserFactory (RawInput), 12

rawReaderFactory, 12

rawReaderFactory (RawInput), 12

RawToChar (Utility), 20

RawToChar-class (Utility), 20

readBin, 12

ReadLinesProducer (ConnectionProducer),  
3

ReadLinesProducer-class  
(ConnectionProducer), 3

ReadTableProducer (ConnectionProducer),  
3

ReadTableProducer-class  
(ConnectionProducer), 3

Reducer, [13](#)  
Reducer-class (Reducer), [13](#)  
register, [19](#)  
register (ParallelParam), [9](#)  
reset, [4](#), [12](#), [14](#)  
reset, Streamer-method (reset), [14](#)  
reset-methods (reset), [14](#)  
Rev (Utility), [20](#)  
Rev-class (Utility), [20](#)

sapply, Producer-method (Producer), [10](#)  
sapply, Stream-method (Stream), [17](#)  
ScanProducer (ConnectionProducer), [3](#)  
ScanProducer-class  
    (ConnectionProducer), [3](#)  
Seq, [15](#)  
Seq-class (Seq), [15](#)  
show, Consumer-method (Consumer), [5](#)  
status, [16](#)  
status, Streamer-method (status), [16](#)  
status-methods (status), [16](#)  
Stream, [2](#), [5](#), [7](#), [8](#), [10](#), [12–16](#), [17](#), [21](#)  
Stream, Consumer-method (Stream), [17](#)  
Stream, Producer-method (Stream), [17](#)  
Stream-class (Stream), [17](#)  
Stream-methods (Stream), [17](#)  
Streamer, [4](#), [5](#), [11](#), [20](#)  
Streamer (Streamer-package), [2](#)  
Streamer-class (Streamer-package), [2](#)  
Streamer-package, [2](#)

Team, [6](#), [9](#), [10](#), [18](#)  
Team, missing-method (Team), [18](#)  
Team, MulticoreParam-method (Team), [18](#)  
Team-class (Team), [18](#)  
tryCatch, [11](#), [18](#)

Utility, [20](#)  
Utility-class (Utility), [20](#)  
Utility-classes (Utility), [20](#)

yield, [2](#), [12](#), [21](#)  
yield, Streamer-method (yield), [21](#)  
yield-methods (yield), [21](#)