Streamer

March 24, 2012

`BufferInt-class` *Class "BufferInt"*

Description

An internal reference class container used by the `Consumer`-class to store functions that operate on the records stored in the `.records` field of the `Consumer`-class. Operations performed on the `.records` field by the `Consumer`-class include `length`, `append`, `subset`.

Users have the options of modifying the behaviour of the above mentioned operations for records of different data types by declaring an S4 method `BufferInterface` that returns an object of `BufferInt`-class.

Constructors

Instances from this class are constructed with calls to `BufferInt` constructor.

Fields

`length`: Object of class `function` that returns the length of the records.

`append`: Object of class `function` that appends records together. This function is called when a new records is read and is to be added to the existing buffer.

`subset`: Object of class `function` that subsets records. This function is called when records have been yielded and are to be removed from the buffer

Methods

Users have the option of controlling the beaviour of the functions `length`, `append` and `subset` used to manipulate the `.records` field of the `Consumer`-class by declaring a function `BufferInterface`.

BufferInterface

Returns an object of class `BufferInt` that holds functions for manipulating the record of the `Consumer` class.

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

Streamer-package, Consumer-class.

Examples

```
showClass("BufferInt")
selectMethod("BufferInterface", "data.frame")
```

ConnectionProducer-class

Class "ConnectionProducer"

Description

A virtual class containing components that are required to create a custom Producer-class to read data from file connections. Users can inherit from the ConnectionProducer-class to create their own Producer classes that interact with files. Users are expected to pass in appropriate reader and parser functions for files when creating instances of classes that inherit from ConnectionProducer-class.

Fields

The ConnectionProducer class inherits the fields verbose, inUse and yieldSize fields from the Streamer class. Please refer to the Streamer class for more details.

An object of class connection.

reader: A function that reads data from a file connection

parser: A function that parses data to records.

Class-Based Methods

The ConnectionProducer class inherits the methods initialize, msg, reset, status and yield from the Streamer virtual class. Please refer to the Streamer class for more details.

Derived classes should implement an appropriate yield method to return the contents of the current stream. The default method for the base virtual Streamer class returns a list()

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

Streamer-package, Producer-class, Streamer-class.

Examples

```
showClass("ConnectionProducer")
```

Consumer-class *Class "Consumer"*

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 or using the connect function.

Methods

Methods defined on this class include:

stream signature(x = "Consumer", ...): see ?stream.

show signature(object = "Consumer"): Display the stream.

Fields

inputPipe: Object of class Streamer, representing the Producer or Consumer connected up-stream to it and from which records are yielded.

.records: Object of classlist which is used as a temporary buffer for storing records.

The Consumer class inherits the fields yieldSize, verbose and inUse from the virtual Streamer class. Please refer to the Streamer class for more details.

Class-Based Methods

initialize(..., inputPipe): A method to initialize the fields of the Consumer class.

inputPipe: An object of class Streamer connected up-stream to it. The class could be a Consumer or Producer which yields data to the Consumer class.

...: Additonal arguments, currently unused.

verbose: A logical(1) instance indicating whether methods invoked on the class should be reported to the user.

reset(): Return the result of delegating reset() to the object in the field inputPipe.

yield(): Return the result of delegating yield() to the object in the field inputPipe.

inputs(): Return a character vector representing up-stream components.

status{}: Reports the status of the Consumer class. A list of the status of the length of the object in the .records field, the classes connected to the inputPipe field and the status of the fields of the virtual class Streamer are returned.

.fill(): An internal method that fills the .records field with yieldSize records if available.

.add(input): An internal method that appends the value passed to the argument input to the .records field.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

Streamer-package, Streamer-class, Producer-class, Stream-class.

Examples

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

Downsample-class *Class "Downsample"*

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. Users interact with this class through the constructor Downsample and methods yield, reset, and stream.

Usage

```
Downsample(probability=0.1, ..., yieldSize=1e6, verbose=FALSE)
```

Arguments

probability	A numeric(1) between 0, 1 indicating the probability with which a record should be retained.
...	Additional arguments, passed to the $new method of the underlying reference class. Currently unused.
yieldSize	A integer(1) indicating the number of records to yield.
verbose	logical(1) indicating whether class methods should report to the user.

Fields

inputPipe: Object of class ANY. The component from which input is retrieved.

probability: Object of class numeric. The probability of including a record in the yield.

yieldSize: Object of class integer storing the number of records to produced each time yield is invoked.

.buffer: Object of class ANY, used internally to store read but not yet parsed records.

verbose: Object of class logical. Display method invokation messages to the user.

Class-Based Methods

initialize(..., probability, yieldSize, verbose): Initialize the instance.

- probability: The probability with which a record is included in the sample.
- yieldSize: The number of records to return when yield is invoked.
- ...: Additional arguments, currently ignored.
- verbose: Display method invokation messages to the user.

reset(): Reset sample buffer and delegate reset to inputPipe.

yield(): Continually invoke yield on inputPipe, accumulating a random sample of yieldSize records until the yield of inputPipe has length 0. The result is a list of length yieldSize.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

stream

Examples

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

NetCDFFile-class *Class "NetCDFFile"*

Description

A NetCDFFile-class to interpret data stored in NetCDF files. Users interact with this class through the constructor NetCDFFile and methods precision, and dimensions.

Usage

```
NetCDFFile(file = character(), ...)
## S4 method for signature 'NetCDFFile'
dimensions(x, ...)
## S4 method for signature 'NetCDFFile'
precision(x, ...)
```

Arguments

file	A character string for the path to the NetCDF file.
x	An insance of the NetCDFFile class.
...	Additional arguments, passed to the $new method of this class. Currently ignored.

Class Methods

precision(): Returns a named character vector corresponding to the storage precision of the variables in the NetCDF file.

dimensions(): Returns a named list containing the names and lengths of the dimensions for each variable in the NetCDF file.

Class Internal Fields: (For developers)

con: Object of class ncdf4. An R ncdf4 connection obtained by opening a NetCDF file from which data is to be read using the nc_open function.

dimensions: A named list corresponding to the names of the variables in the NetCDF file. Each element of the list is a named integer vector, with names of the dimensions for each variable and values the length of the dimension in the NetCDF file.

precision: A named character vector of the number of precision for each variable stored in the NetCDF file.

Class Internal Methods: (For developers)

`initialize(file=character(), ...)`: Called during object creation with `file` being the path to a valid NetCDF file.

`getPath()`: Retrive the path to the NetCDF file.

`getDimensions()`: Retrieves a list of variables, with each element in the list containing a named integer vector of dimensions and their lengths.

`getPrecision()`: Retrieves the precision of each variable in the NetCDF file.

`finalize()`: Close the NetCDF connection `con`.

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

`stream`

Examples

```
## Not run:
pth <- system.file("extdata", "NetCDFData.nc", package = "Streamer")
ncFile <- NetCDFFile(pth)
dimensions(ncFile)
precision(ncFile)
ncFile

## End(Not run)
```

NetCDFInput-class *Class "NetCDFInput"*

Description

A `NetCDFInput`-class to retrieve data store in NetCDF files. Users interact with this class through the constructor `NetCDFInput` and methods `yield`, `status`, and `reset`.

Usage

```
NetCDFInput(ncdf, var, slice, ..., verbose = FALSE)
## S4 method for signature 'NetCDFInput'
dimensions(x, ...)
```

Arguments

`ncdf`	An object of class `NetCDFFile` rrepresenting the file from which data is to be read.
`var`	A `character(1)` string naming the variable to be read from the NetCDF file.
`slice`	A named integer vector specifying the slice to be iterated over. The names correspond to dimensions of `var`, the values to the number of elements to be retrieved with each `yield`.

...	Additional arguments, passed to the $new method of this class. Currently ignored.
verbose	logical(1) indicating whether class methods should report to the user.
x	An instance of the NetCDFInput class.

Class Methods

dimensions(): Return the dimensions associated with the variable this object is iterating over.

yield(): Processes the NetCDF file and retrieves a matrix of data from the NetCDF file corresponding to the slice size that has been set. Repeated calls to the yield function retrieves the next block of data until the end of file has been reached.

reset(): Resets the cursor that tracks the next block of data to be read to the start of the file.

status(): Returns a named numeric vector for the position of the start of the block from which data will be read for the next call to the yield function.

Class Internal Fields: (For developers)

ncdf: An object of class NetCDFFile from which data is being read.

name: A character(1) specifying the name of the variable that is being read.

slice: A named numeric vector specifying the size of the chunk of data that will be retrived along each dimension using the yield method.

start: A named numeric vector specifying the position along each dimension from which data will start to be read for the next call to the yield function.

verbose: Report messages from evalaution?

Class Internal Methods: (For developers)

initialize(ncdf, var, slice, ..., verbose): Called during object creation with values to initialize fields.

yield(): Processes the NetCDF file and retrieves a block of data from the NetCDF file corresponding to the slice size that has been set. Repeated calls to the yield function retrieves the next block of data until the end of file has been reached.

reset(): Resets the cursor that tracks the next block of data to be read to the start of the file.

status(): Retrieves the position of the start of the block from which data will be read for the next call to the yield function.

.getCounts(): Retrieve the dimensions of the next slice.

.getNextStart(): Retrieve coordinates at which next yield starts.

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

NetCDFFile

Examples

```
## Not run:
    showClass("NetCDFInput")

pth <- system.file("extdata", "NetCDFData.nc", package = "Streamer")
ncFile <- NetCDFFile(pth)
dimensions(ncFile) # variable names and dimensions

ncProd <- NetCDFInput(ncFile, "2dIntData", c(sampleDim=5, snpDim=2))
yield(ncProd)
status(ncProd)
reset(ncProd)
yield(ncProd)

## End(Not run)
```

ParallelConnector-class

Class "ParallelConnector"

Description

The `ParallelConnector` `Consumer`-class can be used to parallelize the computations done by blocks directly connected to the `ParallelConnector` and all blocks down-stream to the `ParallelConnector`. i.e Computations performed by the block directly connected up-stream to the `ParallelConnector` and all blocks connected down-stream to the `ParallelConnector` in a stream happen simultaneously.

Usage

```
ParallelConnector(..., yieldSize=1e6, verbose=FALSE)
```

Arguments

`...`	Additional arguments to be passed to the constructor.
`yieldSize`	The number of records the input parser is to yield.
`verbose`	`logical(1)` indicating whether class methods should report to the user.

Constructors

Use `ParallelConnector` to construct instances of this class.

Fields

`.upstream`**:** Object of class `ANY`. The output of a call to the `parallel` function from the `multicore`-package. This field is internal to the `ParallelConnector` class and will be populated by a call to the `stream` method or the `connect` function used to connect the `ParallelConnector` to other blocks in a `stream`.

Methods

`initialize(...)`: Initializes the fields of the `ParallelConnector` class.

`yield()`: Reads data from the child processes and converts the result (which must be a list of `raw`) into a vector of `character`.

`finalize()`: Collects results from the parallel processes and closes the threads started by the `ParallelConnector`.

Author(s)

Nishant Gopalakrishnan, Martin Morgan

See Also

`stream`

Examples

```
showClass("ParallelConnector")
```

Producer-class *Class "Producer"*

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 into records to be passed to `Consumer` classes. `Producer` and `Consumer` instances are associated with each other through the `stream` function or using the `connect` function.

Methods

Methods defined on this class include:

stream `signature(x = "Producer", ...)`: see `?stream`.

show `signature(object = "Streamer")`: Display the stream.

Fields

The `Producer` class inherits the fields `verbose`, `inUse` and `yieldSize` fields from the `Streamer` class. Please refer to the `Streamer` class for more details.

Class-Based Methods

The `Producer` class inherits the methods `initialize`, `msg`, `reset`, `status` and `yield` from the `Streamer` virtual class. Please refer to the `Streamer` class for more details.

Derived classes should implement an appropriate `initialize` method to initialize the fields of the derived class. Additionally, a `yield` method should be implemented to return the contents of the current stream. The default method for the base virtual `Streamer` class returns a `list()`

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

Streamer-package, Consumer-class, Streamer-class.

Examples

```
showClass("Producer")
```

RawInput-class *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(), ..., verbose = FALSE)
rawReaderFactory(blockSize = 1e+06, what)
rawParserFactory(separator = charToRaw("\n"), trim = separator)
```

Arguments

con	A character string or connection (opened as "rb" mode) from which raw input will be retrieved.
yieldSize	The number of records the input parser is to yield.
reader	A function of one argument (con, an open connection positioned at the start of the file, or at the position the con was in at the end of the previous invocation of the reader function) that returns a vector of type raw.
parser	A function of two arguments (buf, bin), parsing the raw vector c(buf, bin) into records.
...	Additional arguments, passed to the $new method of this class. Currently ignored.
verbose	logical(1) indicating whether class methods should report to the user.
blockSize	The number of bytes to read at one time.
what	The type of data to read, as the argument to readBin.

separator — 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.

trim — A raw vector that is a prefix of separator, 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.

verbose: Object of class logical. Should progress be reported?

Class-Based Methods

initialize(con, blockSize, reader, parser, verbose): Called during object creation with values to initialize fields.

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

status(): Summarize status of stream.

yield(): Process stream to yield as many complete records as are represented in the current blockSize elements.

finalize(): Close con.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

stream, connect

Examples

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

RawToChar-class *Class "RawToChar"*

Description

A Consumer-class to convert raw (binary) records to char, applying rawToChar to each record.

Usage

```
RawToChar(yieldSize = 1e6,verbose = FALSE)
```

Arguments

yieldSize	A integer(1) indicating the number of records to yield.
verbose	logical(1) indicating whether class methods should report to the user.

Constructors

Use RawToChar to construct instances of this class.

Fields

inputPipe: Object of class ANY. The component from which input is retrieved.

yieldSize: A integer(1) indicating the number of records to yield.

verbose: Object of class logical. Display method invokation messages to the user.

Methods

yield(): Convert the result of applying yield to inputPipe (which must be a list of raw) into a vector of character.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

stream

Examples

```
showClass("RawToChar")
```

`ReadLinesInput-class`
Class "ReadLinesInput"

Description

A `Producer`-class to interpret text files. Users interact with this class through the constructor `ReadLinesInput` 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. `readLinesReaderFactory` creates a 'reader', whose responsibility it is to accept a connection and return a `character` vector. `readLinesParserFactory` creates a 'parser', responsible for parsing a buffer and vector of the same type as produced by the reader into records.

Usage

```
ReadLinesInput(con, reader = readLinesReaderFactory(),
    parser = readLinesParserFactory(), ..., yieldSize = 1e+06,
    verbose = FALSE)
readLinesReaderFactory(blockSize=1e+06, ...)
scanReaderFactory(blockSize=1e06, ...)
```

Arguments

`con`	A character string or connection (opened as `"r"` mode) from which character data will be retrieved.
`yieldSize`	The number of records the input parser is to yield.
`reader`	A function of one argument (`con`, an open connection positioned at the start of the file, or at the position the `con` was in at the end of the previous invocation of the reader function) that returns a vector of type `character`.
`parser`	A function of two arguments (`buf`, `bin`), parsing the `raw` vector `c(buf, bin)` into records.
`verbose`	`logical(1)` indicating whether class methods should report to the user.
`blockSize`	The number of characters to read at one time.
`...`	Additional arugments.

Fields

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

`blockSize`**:** Object of class `integer`. Size of input during each `yield`.

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

`parser`**:** Object of class `function`. A function used to parse character input into records. See `readLinesParserFactory`

`.records`**:** Object of class `character`. Records that have been read and parsed but not yet yield-ed records.

`verbose`**:** Object of class `logical`. Should progress be reported?

Class-Based Methods

`initialize(..)`: Called during object creation with values to initialize fields.

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

`status()`: Summarize status of stream.

`yield()`: Process stream to yield as many complete records as are represented in the current `blockSize` elements.

`finalize()`: Close `con`.

Author(s)

Martin Morgan `mtmorgan@fhcrc.org`

See Also

`stream`, `connect`

Examples

```
showClass("ReadLinesInput")
```

`Rev-class` *Class "Rev"*

Description

A `Consumer`-class to reverse the order of records. Note that the content of the `yield` is reversed, and not the entire stream.

Usage

```
Rev(yieldSize = 1e6, verbose=FALSE)
```

Arguments

`yieldSize`	A `integer(1)` indicating the number of records to yield.
`verbose`	`logical(1)` indicating whether class methods should report to the user.

Constructors

Use `Rev` to construct instances of this class.

Fields

`inputPipe`: Object of class `ANY`. The component from which input is retrieved.

`yieldSize`: A `integer(1)` indicating the number of records to yield.

`verbose`: Object of class `logical`. Display method invokation messages to the user.

Methods

`yield()`: Reverse the result of applyng `yield` to `inputPipe`.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

stream

Examples

```
showClass("Rev")
```

Stream-class *Class "Stream"*

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.

Constructors

Instances from this class are constructed with calls to stream; see ?stream

Methods

This class inherits the following methods:

reset signature(x = "Streamer", ...): see ?reset.

yield signature(x = "Streamer", ...): see ?yield.

Methods defined on this class include:

length signature(x = "Stream"): the number of components in this stream

[[signature(x = "Stream", i = "numeric"): The ith component (including inputs) of this stream.

show signature(object = "Stream"): Display the stream.

Fields

inputPipe**:** Object of class ANY ~~

verbose**:** A logical(1) instance indicating whether methods invoked on the class should be reported to the user.

Class-Based Methods

The following methods are inherited (from the corresponding class): initialize ("Streamer"), yield ("Streamer"), msg ("Streamer"), yield ("Consumer"), initialize ("Consumer"), reset ("Consumer"), reset ("Streamer"), inputs ("Consumer")

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

Streamer-package, Consumer-class, Producer-class.

Examples

```
showClass("Stream")
```

Streamer-class *Class "Streamer"*

Description

A virtual base class from which all classes in the Streamer package derive.

Methods

reset `signature(x = "Streamer")`: see `?reset`.

yield `signature(x = "Streamer")`: see `?yield`.

Fields

`yieldSize`: An `integer` for the number of records to be returned.

`verbose`: A `logical(1)` instance indicating whether methods invoked on the class should be reported to the user.

`inUse`: A `logical(1)` instance indicating whether the object instantiated has been used in a stream.

Class-Based Methods

`initialize(..., verbose = FALSE)`: Initialize `Streamer`, setting `verbose`, `yieldSize` and `inUse` fields, returning `.self` invisibly.

`msg(fmt, ...)`: Use `msg` to print `sprintf(fmt, ...)` messages to user.

`reset()`: Reset `Streamer`; this default method is a no-op.

`yield()`: Yield default value `list()`.

`status()`: Reports the status of the `Streamer` class. A `list` of the status of `yieldSize`, `verbose` and `inUse` fields is returned.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

Streamer-package, Consumer-class, Producer-class, Stream-class.

Examples

```
showClass("Streamer")
```

Streamer-package *Enable stream processing of large files*

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

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

b <- RawInput(fl, 5000L, verbose=TRUE)
d <- Downsample(yieldSize=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
}
```

TConnector-class *Class "TConnector"*

Description

A Consumer-class that is used to connect the output of one stream to several Consumer stream's that perform different operations on the records. The TConnector manages the records supplied to it to ensure that all the streams connected to it get acess to all the records passed to the TConnector irrespective of the number of records processed at a time by each stream connected down-stream.

A TConnector can be connected to other Producer and Consumer objects using the connect function.

Usage

```
TConnector(..., yieldSize=1e6, verbose=FALSE)
```

Arguments

...	Additional arguments
verbose	logical(1) indicating whether class methods should report to the user.
yieldSize	The number of records the input parser is to yield.

Constructors

Use TConnector to construct instances of this class.

Fields

.records: A temporary buffer used to save records retrieved from the Producer or Consumer class connector up-stream to the TConnector. This field is used internally by class methods and is not intende to be manipulated directly by the user.

.tOuts: A list of objects of class TOut of length equal to the number of streams connected down-stream to it. This field is used internally by the TConnector-class method and not intended to be manipulated directly by the user.

Methods

initialize(...): A method to initialize the fields of the TConnector class.

.fill(): An internal method used to retrieve records from the Producer or Consumer class connected up-stream to the TConnector.

.add(): An internal method to add records to the internal buffer(.records).

.dump(): An internal method to remove records that have been passed down to all the down-stream classes (and are no longer needed) from the internal buffer(.records).

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

stream,YConnector, connect

Examples

```
### Two Streams b, c1 and b, c2 connected with a Tconnector t
fl <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
b <- RawInput(fl, 100L, reader=rawReaderFactory(1e4))
### c1 and c2 return different number of records
c1 <- RawToChar(10L)
c2 <- RawToChar(20L)
t <- TConnector()

### Connect the blocks together using the connect function
blocks <- structure(list(b, c1, c2, t), names = c("b", "c1", "c2", "t"))
df <- data.frame(from =c("b", "t", "t"),
                 to = c("t", "c1", "c2"))
res <- connect(blocks, df)

### yield on c2 returns 20 records
yield(res$c2)
### yield on c1 returns the same records as with yield on c2
### 10 records at a time
yield(res$c1)
yield(res$c1)
```

TOut-class *Class "TOut"*

Description

A Consumer-class that is used internally to connect several Consumer streams to a TConnector-class.

This class is only for use by functions internal to the Streamer package. The TOut-class is responsible for filing the .records field of the TConnector with adequate number of records.

Usage

```
TOut(..., yieldSize=1e6, verbose=FALSE)
```

Arguments

...	Additional arguments. Currently not used
verbose	logical(1) indicating whether class methods should report to the user.
yieldSize	The number of records the input parser is to yield.

Constructors

Use `TOut` to construct instances of this class.

Fields

`start`: A `integer(1)` indicating the start position of the record to be read next.

`.records`: A temporary buffer of the `TConnector` from which the next set of records are to be read.

Methods

`initalize(...)`: Initializes the fields of the `TOut` class.

`yield()`: Retrieves records from the `TConnector`-class connected up-stream to it.

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

`TConnector`

Examples

```
showClass("TOut")
```

`UserFunction-class` *Class "UserFunction"*

Description

The `UserFunction` class is provided as a convenience class enabling users to quickly create `Consumer`-classes that can be added to a stream without having to go into more complex details about the implementation of the classes hierarchy provided by the `Streamer`-package.

The users pass in a function `fun` to the constructor of the `UserFunction`-class to manipulate the records returned by the class intended to be connected upstream. The constructor returns an instance of the `UserFunction`-class with a with a `yield` method that the user can directly invoke.

Usage

```
UserFunction(fun, ..., yieldSize=1e6, verbose=FALSE)
```

Arguments

`fun`	User defined function that operates on records yielded by the class connected upstream.
`...`	Additional arguments, passed to the `$new` method of this class. Currently ignored.
`yieldSize`	A `integer(1)` indicating the number of records to yield.
`verbose`	`logical(1)` indicating whether class methods should report to the user.

Constructors

Use `UserFunction` to construct instances of this class.

Fields

`.fun`: A user supplied`function` that operates on records yielded by the class connected upstream.

Methods

`initialize(...)`: Initializes the fields of the `UserFunction`-class.

`yield()`: Applies the function `fun` to the records retrieved from the class connected upstream to the `UserFunction` class.

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

`stream`

Examples

```
f <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
b <- RawInput(f, 100L, reader=rawReaderFactory(1e4))
### Create a user defined function to convert raw bytes to character
myFun <- function(x) {
    sapply(x, rawToChar)
}

#### Pass the function to the UserFunction constructor
d <- UserFunction(fun=myFun)

#### Create a stream
s <- stream(b, d)
yield(s)
```

`Utility-class` *Class "Utility"*

Description

A virtual class containing components that are required to create light weight `Consumer` classes that process data from other `Producer` or `Consumer` classes. Users can inherit from the `Utility`-class to create their own `Consumer`-classes that performs some operation on the records passed down from a class upstream. The classes `RawToChar` and `Rev` implemented in the `Streamer`-package derive from the `Utility`-class.

Fields

The Utility class inherits the fields verbose, inUse and yieldSize fields from the Streamer class. Please refer to the Streamer class for more details.

The Utility class inherits the fields inputPipe, and .records from the Consumer class. Please refer to the Consumer-class class for more details.

Class-Based Methods

The ConnectionProducer class inherits the methods initialize, msg, reset, status and yield from the Streamer virtual class. Please refer to the Streamer class for more details.

Derived classes should implement an appropriate yield method to return the contents of the current stream. The default method for the base virtual Streamer class returns a list()

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

Streamer-package, Consumer-class, Streamer-class.

Examples

```
showClass("ConnectionProducer")
```

YConnector-class *Class "YConnector"*

Description

The YConnector Consumer-class can be used to combine the output of multiple stream's together. The output records of the stream's are combined using a user supplied function (fun) passed to the constructor of the YConnector class. The output of the YConnector can then be used to feed a Consumer-class connected down-stream to it.

The YConnector can be connector to other Producer and Consumer objects using the connect function.

Usage

```
YConnector(fun, ..., yieldSize=1e6, verbose=FALSE)
```

Arguments

fun	A function that is used to combine the output of the streams connected up-stream to the YConnector. The function fun takes named arguments. The names correspond to the names of the objects passed to the connect function used to connect the YConnector to up-stream Streamer classes.
...	Additional arguments. Currently not used
verbose	logical(1) indicating whether class methods should report to the user.
yieldSize	The number of records the input parser is to yield.

Methods

Methods defined on this class include:

show `signature(object = "YConnector")`: Displays the names of the up-stream components to which the `YConnector`-class has been connected.

Constructors

Use `YConnector` to construct instances of this class.

Fields

`.fun`: User defined `function` to combine the output of several streams. The function is applied on the named outputs obtained by calling the `yield` method on the named streams connected upstream to it.

`.upstream`: A named `list` of objects connected up-stream to the `YConnector`-class. This field is meant to be internal to the class and is only modified by using the `connect` function to connect the `YConnector` to other `Streamer` objects.

Methods

`initialize(..., fun)`: A method to initialize the fields of the `YConnector`-class.

`yield()`: Yields the records obtained by applying the function `fun` to the result obtained by calling `yield` on all the `stream`'s connected up-stream to it.

Author(s)

Nishant Gopalakrishnan ngopalak@fhcrc.org

See Also

`stream`,`TConnector`, `connect`

Examples

```
fl <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
#### Blocks for stream1
b1 <- RawInput(fl, 100L, reader=rawReaderFactory(1e4))
c1 <- RawToChar(10L)

#### Blocks for stream2
b2 <- RawInput(fl, 100L, reader=rawReaderFactory(1e4))
c2 <- RawToChar(20L)
#### YConnector with function list for combining the blocks
y <- YConnector(fun=list)

blocks <- structure(list(b1,c1, b2, c2, y),
                    names = c("b1", "c1", "b2", "c2","y"))
df <- data.frame(from =c("b1", "b2", "c1", "c2"), to = c("c1", "c2", "y", "y"))
#### Connect the blocks using the connect function
res <- connect(blocks, df)
y

#### Yield data from the y connector
```

```
yield(res$y)
```

`connect` *Connect 'Producer' and 'Consumer' streams together and return a named 'list' of 'stream"s that the user can invoke the method 'yield' on.*

Description

The function `connect` can be used to connect `Producer` and `Consumer` components together.

For simple streams, it may be more appropriate to use the `stream` method. The `connect` function is useful for connecting together more complex `streams` involving classes such as `YConnector`, `TConnector`, `ParallelConnector` etc which cannot be handled by the `stream` method.

The `connect` function returns a named list of possible `streams` from the connection information provided by the user. The user can then call `yield` on the streams to obtain records.

Usage

```
connect(blocks, df)
```

Arguments

`blocks`	A named list of instances of classes `Consumer` and `Producer` to the connected together in a `stream`
`df`	A `data.frame` with two columns: "from" and "to" which are character vectors corresponding to the names of the blocks. Each row of `df` describes a connection between `Consumer` or `Producer` blocks.

Details

Arguments `blocks` must consist of a named list of a single `Producer` and zero or more `Consumer` components.

Value

A named `list` of instances of class `Stream`.

Author(s)

Nishant Gopalakrishnan `ngopalak@fhcrc.org`

See Also

`yield`,`connect`, `Stream-class`.

Examples

```
###  A simple stream involving a Producer and Consumer class
fl <- system.file("extdata", "s_1_sequence.txt", package="Streamer")
b <- RawInput(fl, 100L, reader=rawReaderFactory(1e4))
c <- RawToChar(10L)

###  Create a named list of the blocks to be connected together
blocks <- structure(list(b,c), names = c("b", "c"))

## Create a data.frame that describes the connection between blocks
df <- data.frame(from ="b", to = "c")
res <- connect(blocks, df)
yield(res$c)
reset(res$c)
while (length(yield(res$c))) cat("tick\n")
```

`reset` *Reset a stream, or a stream component and all inputs.*

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, ...)
## S4 method for signature 'Streamer'
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`

See Also

`stream`, `Producer`, `Consumer`.

Examples

```
## see example(stream)
```

status *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

See Also

stream, Producer, Consumer.

Examples

```
## see example(stream)
```

stream *Create a stream from Consumer and Producer components.*

Description

stream is used to create a stream from a single Producer and zero or more Consumer instances.

Usage

```
stream(x, ..., verbose=FALSE)
## S4 method for signature 'Producer'
stream(x, ..., verbose=FALSE)
## S4 method for signature 'Consumer'
stream(x, ..., verbose=FALSE)
```

Arguments

x	An instance of a Consumer or Producer
...	Additional Consumer or Producer instances.
verbose	A logical(1) indicating whether status information should be reported.

Details

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.

Value

An instance of class Stream.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

yield, Stream-class.

Examples

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

---

yield *Iterate a stream to yield one chunk of data.*

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

Arguments

x	A Stream, Producer, or Consumer object.
...	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

See Also

stream, Producer, Consumer.

Examples

```
## see example(stream)
```

Index