

---

# **fcs-io Documentation**

***Release 0.1.0***

**Jason L. Weirather**

**Jun 04, 2017**



**CONTENTS:**

<b>1</b>	<b>Module</b>	<b>1</b>
1.1	FCS . . . . .	1
1.2	HEADER data . . . . .	5
1.3	TEXT segment . . . . .	7
1.4	DATA segment . . . . .	14
<b>2</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



## 1.1 FCS

**class** `fcsio.FCS` (*bytes=None, fcs=None*)

Bases: `object`

The primary class for working with FCS file data is the FCS class.

---

**Note:** Use this class to work with FCS data.

---

A complete list of classes within the module is included because many of the methods and properties of this class are helper classes and descriptions of those classes will explain their available properties and methods.

### Parameters

- **bytes** (*bytearray*) – The raw data of the FCS file
- **fcs** – `fcsio.FCS` object to create a new FCS from. Used by copy.

### `copy()`

Output an fcs object that is the same content as self

Creates a new object for everything EXCEPT the OTHER fields (for now)

**Warning:** The copy is neither a perfect copy, nor is it memory independent. OTHER fields are still passed by reference. And no attempt is made at outputting identical bytes as input. If you need a completely new FCS object unlinked to the old, you can use `totally_new = FCS(myoldfcs.output_constructor().fcs_bytes)`

**Returns** Make a new FCS object that is a copy of this one.

**Return type** `fcsio.FCS`

### `data`

access the DATA segment

**Returns** Get an object for accessing the DATA segment

**Return type** `fcsio.data.Data`

### `filter`

Filter an FCS file through a variety of methods defined by the `fcsio.filter.Filter` class. Filter methods will then return a new FCS object based on a copy of the current, see the Filter class for more information.

**Returns** an object for filtering the FCS object

**Return type** `fcsio.filter.Filter`

#### **other**

access the OTHER segments (user defined fields specified at the end of the header)

**Returns** Get the data from OTHER user defined segments at the end of the header

**Return type** list of bytearrays

**output\_constructor** (*essential=False, adjust\_range=True*)

Get the bytes of an actual file for an FCS object through the output\_constructor method is required to be called.

---

**Note:** You can just access this functionally if you like, or you can assign it to a variable if you have some reason to access it multiple times to reduce computation.

---

**Warning:** If you need to access multiple propertyys from the factory, you should save the factory and access them all from that same instance of factory. The common use case will only be to access fcs\_bytes, so this shouldn't usually be an issue.

#### **Parameters**

- **essential** (*bool*) – An optional argument, where if set to True, will trim off the OTHER segements. The other objects are passed as references so clearing them may not be generally necessary for conserving memory.
- **adjust\_range** (*int*) – An optional argument, default True, to adjust range of parameter to that parameters largest current value

**Returns** Generate an object with methods necessary for outputing a new FCS file (bytes that can be written)

**Return type** `fcsio.FCSFactory`

#### **parameters**

access to parameters. These are originally defined in keywords but are not accessible through `fcsio.text.Text` by keyword name.

Parameters needs to be accessed after both text and data have been intialized because data is coupled to parameters. Any changes is parameters will also affect data.

**Returns** get an object for accessing/modifying paramters

**Return type** `fcsio.text.parameters.Parameters`

#### **standard**

access and set where possible standard TEXT fields through here. You can access these fields through `fcsio.text.Text` also but are limited to string input and outputs.

**Warning:** When first read in, these will have original values, but values can change while altering the file, and after the output constructor is called, expect byte ranges to shift.

**Alsosee** `fcsio.text.Text`

**Returns** get an object for accessing/modifying standard keywords in TEXT

**Return type** `fcsio.text.standard.Standard`

## text

access the TEXT segment

**Note:** The object containing the TEXT segment initially separates the keywords from per-parameter keywords. The per-parameter information is better accessed through the FCS property `fcsio.FCS.parameters`. The standard FCS keywords are accessible via properties of the `fcsio.FCS.standard`. This is a better choice for getting and setting values if you want to use more logical types instead of strings for everything. Finally keep in mind that although TEXT may contain information such as DATABEGINS or DATAENDS, these values are not updated until the `fcsio.FCS.output_constructor` method is called. At this point they will be updated.

**Warning:** When first read in, these will have original values, but values can change while altering the file, and after the output constructor is called, expect byte ranges to shift.

**Alsosee** `fcsio.FCS.parameters`

**Alsosee** `fcsio.FCS.standard`

**Returns** Get the object for accessing the TEXT segment data

**Return type** `fcsio.text.Text`

## version

Version as listed in the first 10 bytes of the header

**Returns** version

**Return type** `string`

**class** `fcsio.FCSFactory` (*fcs*, *essential=False*, *adjust\_range=True*)

Bases: `object`

A class to hold a created header and byte values so that data and text bytes corresponding to the header don't need to be recomputed

**Warning:** FCSFactory can do some modifications to the TEXT segment defined in the `fcsio.FCS` object used to initialize the class. These are done to set appropriate byte conditions since keywords, parameters, and data may have been modified.

**Note:** There is a bit of a conundrum. We need to set the TEXT size, but part of the TEXT is the `data_start` and `data_end`. Setting `data_start` and `data_end` could change the size of TEXT, thus changing `data_start` and `data_end`. We deal with this by using buffers that leave enough room between segments to accomodate size changes of segments based on value replacements.

## Parameters

- **fcs** (`fcsio.FCS`) – The FCS object being staged for output

- **essential** (*bool*) – optional, False by default, and if True, trim off the OTHER segments
- **adjust\_range** (*bool*) – optional, True by default, adjust the range of each parameter to have a max in that is rounded above the or equal to the highest value

**data\_bytes**

get the DATA segment bytes of the FCS object

**Returns** the bytes in the DATA segment

**Return type** *bytearray*

**fcs\_bytes**

get the real data as a bytearray from the FCS object

**Returns** The FCS raw data

**Return type** *bytearray*

**header\_bytes**

get the header bytes of the FCS object

**Returns** the bytes bound for the header

**Return type** *bytearray*

**other**

get the OTHER segment bytes as a list of bytearrays

**Returns** any data segments defined as OTHER

**Return type** list of bytearrays

**parameters**

get the parameters accessing object from TEXT after adjustments

**Warning:** This will have different ranges unless you keep them the same by output constructor options

**Returns** object to access standard keywords and values

**Return type** *fcsio.text.parameters.Parameters*

**standard**

get the standard TEXT object after adjustment for output

**Warning:** This will likely be different than what was read in for byte ranges

**Returns** object to access standard keywords and values

**Return type** *fcsio.text.standard.Standard*

**text**

get the text object after adjustment for output

**Warning:** This will likely be different than what was read in for byte ranges



**Returns** Text object ready for output

**Return type** *fcsio.text.Text*

**text\_bytes**

get the TEXT segment bytes of the FCS object

**Returns** the bytes in the TEXT segment

**Return type** *bytearray*

**class** *fcsio.filter.Filter* (*fcs*)

Bases: *object*

Filter an FCS class according by various options and output an FCS

---

**Note:** This class is should be accessed through the *fcsio.FCS.filter* method.

---

When filter is called, a copy is created of the FCS class used to initialize it is generated. This copy is subsequently modified as necessary in the filtering process.

**Parameters** *fcs* (*fcsio.FCS*) – Start filtering from an FCS class.

**events** (*row\_indecies*)

Filter the events (or cells) by index. This is to facilitate subsetting the data with random sampling.

**Parameters** *row\_indecies* (*list*) – A list of indecies (0-indexed) of events to include

**Returns** A filtered FCS

**Return type** *fcsio.FCS*

**gate** (*short\_name*, *min=None*, *max=None*)

Filter the FCS file based on values of a parameter

include greater than or equal to min if set include less than or equal to max if set

**Parameters**

- **short\_name** (*string*) – PnN short name
- **min** (*float*) – remove anything less than this
- **max** (*float*) – remove anything greater than this

**minimize** ()

Trim down the FCS to all but the minimal number of required fields

by its very nagture this is a very lossy filter, but could concievably help with some memory issues that could come up.

**none** ()

**parameters** (*short\_names=None*)

## 1.2 HEADER data

**class** *fcsio.header.ByteIndecies* (*start*, *end*, *in\_header*)

Bases: *tuple*

1-indexed byte locations of start and end with a bool specifying if it was defined in the header or not

**count** (*value*) → integer – return number of occurrences of value

**end**

Alias for field number 1

**in\_header**

Alias for field number 2

**index** (*value*[, *start*[, *stop*]]) → integer – return first index of value.

Raises ValueError if the value is not present.

**start**

Alias for field number 0

**class** `fcsio.header.Header` (*data*)

Bases: `object`

The header class is only used temporarily when reading the FCS from data. The version is the only value that doesn't change depending on how the data is altered.

3.0 from the spec 1997 Seamer Current Protocols in Cytometry

1. FCS 3.0 followed by spaces 00-09
2. ASCII-encoded offset to first byte of TEXT segment 10–17
3. ASCII-encoded offset to last byte of TEXT segment 18–25
4. ASCII-encoded offset to first byte of DATA segment 26–33
5. ASCII-encoded offset to last byte of DATA segment 34–41
6. ASCII-encoded offset to first byte of ANALYSIS segment 42–49
7. ASCII-encoded offset to last byte of ANALYSIS segment 50-57
8. OTHER segments 58 to start of TEXT

`fcsio.header.ByteIndecies` tell the 'start', the 'end' and whether or not (bool) a numerical range was read from the header. If no numerical range was reported it is likely either absent from the file or coded in the TEXT

**Parameters** *data* (`bytearray`) – The FCS file data (not just the first 58 bytes)

**analysis\_range**

Can be zero if the end exceeds 99,999,999. To indicate it is set within the TEXT segment. or that its absent.

**Returns** get the analysis range. it is required to be present.

**Return type** `fcsio.header.ByteIndecies`

**data\_range**

Can be zero if the end exceeds 99,999,999. To indicate it is set within the TEXT segment.

**Returns** get the data range. it is required to be present.

**Return type** `fcsio.header.ByteIndecies`

**other\_ranges**

If there is extra data in the header return it

Assume pairs of coordinates for now as the only thing we'll see for one or more user segments.

**Returns** Get the coordinates of OTHER segments

**Return type** list of `fcsio.header.ByteIndecies`

**text\_range**

Should be between the end of the header and less than 99,999,999

**Returns** get the text range. it is required to be present.

**Return type** `fcsio.header.ByteIndecies`

**validate** (*verbose=True*)

Validate for 3.0 and 3.1 together for now

**Returns** Is it a valid header?

**Return type** `bool`

**version**

get the version

**Returns** version string from the beginning of the header with whitespace removed

**Return type** `string`

## 1.3 TEXT segment

Core for dealing with the TEXT definitions

Note that byte-offset fields may change, and these will need be set during the process of preparing an output FCS file. These fields will be available for reading prior to this but they will not have meaning until output.

The `fcsio.StageFCS` class will do all this cleaning up of coordinates prior to outputting the FCS file.

**class** `fcsio.text.KeyWordDict` (*kvs=[]*)

Bases: `object`

A dictionary for accessing keywords that

1. Preserves the original keywords
2. Provides case-INSENSITIVE access to keywords as per the spec
3. Does not allow empty strings as keywords or values

Since keywords are case insensitive, take them to uppercase,

**Parameters** *kvs* (*list of key-value pairs*) – key-value pairs

**keys** ()

Keywords of TEXT that can be accessed and modified. This does not include the parameter keywords, because they must be accessed through the `fcsio.FCS.parameters` property, or through the property here of `fcsio.text.Text.parameter_data`

**Returns** list of keywords

**Return type** `list`

**parameter\_data**

Parameter data keyed by index then generic keyword

**Returns** dict of parameters keyed by their index and generic keyword

**Return type** `dict`

**class** `fcsio.text.RegexDescriptor` (*regex\_string, keyword, regex*)

Bases: `tuple`

Store regular expressions that describe documented keywords

**count** (*value*) → integer – return number of occurrences of value

**index** (*value*[, *start*[, *stop*]]) → integer – return first index of value.  
Raises ValueError if the value is not present.

**keyword**

Alias for field number 1

**regex**

Alias for field number 2

**regex\_string**

Alias for field number 0

**class** `fcsio.text.Text` (*data=None*)

Bases: `fcsio.text.KeyWordDict`

Parse the TEXT portion of the file. Assumes that the header has been processed in set in `self._header`.

In version 3.0 of the FCS specification, the TEXT must fit entirely in the first 99,999,999 bytes of data.

Text object can be accessed as a dictionary, as it is composed of key/value pairs, but these will all be strings keys and string values.

Can take an input data from an FCS file, but this object is mutable.

**bytes**

Get the data bytes from TEXT.

**Warning:** To properly prepare the TEXT data with the byte ranges set correctly, you should not call `fcsio.text.Text.bytes` property directly. Rather you should use `fcsio.FCS.construct_fcs`

**Returns** the data

**Return type** `bytearray`

**keys** ()

Keywords of TEXT that can be accessed and modified. This does not include the parameter keywords, because they must be accessed through the `fcsio.FCS.parameters` property, or through the property here of `fcsio.text.Text.parameter_data`

**Returns** list of keywords

**Return type** list

**miter = None**

Bake removing the escape characters into generating the key value pairs

**parameter\_data**

Parameter data keyed by index then generic keyword

**Returns** dict of parameters keyed by their index and generic keyword

**Return type** `dict`

`fcsio.text.get_required_keywords` ()

Return a dictionary of the required keywords

**Returns** keywords that are required, as keys to their description

**Return type** `dict`

**class** `fcsio.text.standard.Standard`(*text*)

Bases: `object`

interact with text fields through standard key words, these EXCLUDE parameters which are available in 'parameter'

**Warning:** While you may be able to read and alter parameters here, any byte ranges will hold very little meaning except when first read and describing an original file or after an `output_constructor` has been called and you

#### **BEGINANALYSIS**

Get the byte offset for BEGINANALYSIS

**Setter:** assign an *int* to the keyword

**Returns** the BEGINANALYSIS value

**Return type** `int`

#### **BEGINDATA**

Get the byte offset for BEGINDATA

**Setter:** assign an *int* to the keyword

**Returns** the BEGINDATA value

**Return type** `int`

#### **BEGINTEXT**

Get the byte offset for BEGINTEXT supplemental text

**Setter:** assign an *int* to the keyword

**Returns** the BEGINTEXT value

**Return type** `int`

#### **BYTEORD**

Get the byte order type as either 'little endian' or 'big endian'

**Setter:** set the key word from either 'little endian' or 'big endian' string

**Returns** the BYTEORD description

**Return type** string that is either 'little endian' or 'big endian'

#### **DATATYPE**

Type of the DATA in the data segment

**Setter:** assign an *char* to the keyword

**Returns** the DATATYPE

**Return type** `char`

#### **ENDANALYSIS**

Get the byte offset for ENDANALYSIS

**Setter:** assign an *int* to the keyword

**Returns** the ENDANALYSIS value

**Return type** `int`

**ENDDATA**

Get the byte offset for ENDDATA

**Setter:** assign an *int* to the keyword

**Returns** the ENDDATA value

**Return type** *int*

**ENDSTEXT**

Get the byte offset for ENDSTEXT supplemental text

**Setter:** assign an *int* to the keyword

**Returns** the ENDSTEXT value

**Return type** *int*

**MODE**

Get the MODE of data layout

**Setter:** assign an *char* to the keyword

**Returns** the MODE value

**Return type** *char*

**NEXTDATA**

Get the byte offset for NEXTDATA the next fcs data

**Setter:** assign an *int* to the keyword

**Returns** the NEXTDATA value

**Return type** *int*

**PAR**

Get the number of parameters

**Warning:** You cannot set the PAR, as it is inferred from the parameters present

**Returns** the PAR value

**Return type** *int*

**TOT**

Get the number of events

**Warning:** You cannot set the TOT, as it is inferred from the parameters present

**Returns** the TOT value

**Return type** *int*

**class** `fcsio.text.parameters.Parameter` (*i, pdata*)

Bases: `object`

A class defining a single Parameter. These are attributes being recorded by the machine for each event, and they come with a number of keywords describing each one.

**Parameters**

- **i** (*int*) – index of the parameter (0-indexed)
- **pdata** (*dict*) – The dict keyed by index, then generic parameter keyword accessible through `fcsio.text.Text`

---

**Note:** Can be accessed as a dictionary by keyword. Use `get_keywords()` to get available keywords, e.g. ‘\$PnS’

---

#### **amplification\_gain**

\$PnG gain used to amplify the value of this parameter

**Setter:** assign a *float* to \$PnG

**Returns** \$PnG

**Return type** *float* or *None*

#### **amplification\_type**

\$PnE attribute for this paramete.

##### **REQUIRED ATTRIBUTE**

Specifies whether the the parameter data is stored on linear or logarithmic scale. For a linear scale, (0,0) will be used. When floating point storage is used, this linear scale will be used

Here it returns as a tuple pair of floats and the first float is the number of log decades and the second is the linear value that would have a log value of 0

---

**Note:** Can be converted with the formula  $10^{(f1*xc/r)}*f2$  where xc is the channel value.

---

**Setter:** assign an (*float1, float2*) to \$PnE

**Returns** \$PnE

**Return type** (*float*, ‘float’)

#### **bits**

\$PnB attribute for this parameter

##### **REQUIRED ATTRIBUTE**

Please keep in mind that bits is required to be 32 for type F. If \$DATATYPE is F this will represent the number of characters for each data value

**Setter:** assign an *int* to \$PnB

**Returns** \$PnB

**Return type** *int*

#### **calibration**

\$PnCALIBRATION short name attribute for this parameter

Parameter has 1 per *float* scale units.

**Setter:** assign a tuple (*float*, ‘string’) to \$PnCALIBRATION

**Returns** \$PnCALIBRATION

**Return type** (*float*, ‘string’) or *None*

#### **detector\_type**

\$PnT detector type for this parameter

**Setter:** assign a *string* to \$PnT

**Returns** \$PnT

**Return type** string or *None*

**detector\_voltage**

\$PnT detector type for this parameter

**Setter:** assign a *string* to \$PnT

**Returns** \$PnT

**Return type** string or *None*

**emitted\_light**

**\$PnP amount of light collected by dector for a parmeter as a** percentage of emitted light

**Setter:** assign an *int* to \$PnP

**Returns** \$PnP as an integer percent

**Return type** int or *None*

**excitation\_power**

\$PnO power of lightsource in mW used to excite the parameter

**Setter:** assign an *int* to \$PnO

**Returns** \$PnO

**Return type** int or *None*

**excitation\_wavelengths**

\$PnL excitation wavelength(s) for parameter N

**Setter:** assign a list of ints to \$PnL

**Returns** \$PnL

**Return type** list of ints or *None*

**get\_keywords ()**

Get all the keywords available for this parameter

**Returns** keywords available

**Return type** *list* of strings

**index**

get the 0-indexed order number of the parameter :return: index (0-based) :rtype: int

**keys ()**

an alias for get\_keywords

**Alsosee** `fcsio.text.parameters.Parameter.get_keywords`

**long\_name**

\$PnS defines a long name for the parameter

**Setter:** assign a *string* to \$PnS

**Returns** \$PnS long name

**Return type** string or *None*



**optical\_filter**

\$PnF optical filter attribute for this parameter

**Setter:** assign a *string* to \$PnF

**Returns** \$PnF

**Return type** *string* or *None*

**range**

\$PnR max range attribute for this parameter

**REQUIRED ATTRIBUTE**

Holds the maximum value that data for this parameter can reach. The value stored can exceed the true max.

**Setter:** assign an *int* to \$PnR

**Returns** \$PnR

**Return type** *int*

**short\_name**

\$PnN short name attribute for this parameter

**REQUIRED ATTRIBUTE**

**Setter:** assign a *string* to \$PnN

**Returns** \$PnN

**Return type** *string*

**visualization\_scale**

\$PnD short name attribute for this parameter

visualization scale recommendations for parameter

**Setter:** assign a tuple (*string*, 'float', 'float') to \$PnD

**Returns** \$PnD

**Return type** (*string*, 'float', 'float') or *None*

**class** `fcsio.text.parameters.Parameters` (*text*, *data*)

Bases: `object`

class to access and modify the parameters defined by the TEXT segment with data stored in the DATA segment  
input values are usually accessed through FCS properties

**Parameters**

- **text** (`fcsio.text.Text`) – the text object associated with an FCS object
- **data** (`fcsio.data.Data`) – the data object associated iwth an FCS object

**delete** (*short\_names*=[])

Remove a list of parameters defined by the list of short names

**Parameters** **short\_names** – the PnP short names as a list

**indexOf** (*short\_name*=None)

Get the index in a list of parameters according to a short\_name of a parameter

**Parameters** **short\_name** (*string*) – the PnN short name

**Returns** get he index of the short name

**Return type** `int`

**parameters**

Return the list of parameters as `Parameter` objects

**Returns** parameters

**Return type** list of `fcsio.test.parameters.Parameter`

## 1.4 DATA segment

**class** `fcsio.data.Data` (*data*, *standard*, *text*)

Bases: `object`

The class for working with the data attached to an fcs object

**Parameters**

- **data** (`bytearray`) – bytes of the Data segment
- **standard** (`fcsio.text.standard.Standard`) – class for interfacing with the TEXT
- **text** (`fcsio.text.Text`) – main TEXT class

**bytes**

Get the data

**Returns** constructed data

**Return type** `bytearray`

**event\_count**

Get the number of events

**Returns** the event count

**Return type** `int`

**matrix**

Get the data matrix

**Returns** stored data matrix

**Return type** 2D matrix list of rows, rows are lists of float

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### f

- `fcsio`, [1](#)
- `fcsio.data`, [14](#)
- `fcsio.filter`, [5](#)
- `fcsio.header`, [5](#)
- `fcsio.text`, [7](#)
- `fcsio.text.parameters`, [10](#)
- `fcsio.text.standard`, [8](#)



## A

amplification\_gain (fcsio.text.parameters.Parameter attribute), 11  
 amplification\_type (fcsio.text.parameters.Parameter attribute), 11  
 analysis\_range (fcsio.header.Header attribute), 6

## B

BEGINANALYSIS (fcsio.text.standard.Standard attribute), 9  
 BEGINDATA (fcsio.text.standard.Standard attribute), 9  
 BEGINTEXT (fcsio.text.standard.Standard attribute), 9  
 bits (fcsio.text.parameters.Parameter attribute), 11  
 ByteIndecies (class in fcsio.header), 5  
 BYTEORD (fcsio.text.standard.Standard attribute), 9  
 bytes (fcsio.data.Data attribute), 14  
 bytes (fcsio.text.Text attribute), 8

## C

calibration (fcsio.text.parameters.Parameter attribute), 11  
 copy() (fcsio.FCS method), 1  
 count() (fcsio.header.ByteIndecies method), 5  
 count() (fcsio.text.RegexDescriptor method), 7

## D

Data (class in fcsio.data), 14  
 data (fcsio.FCS attribute), 1  
 data\_bytes (fcsio.FCSFactory attribute), 4  
 data\_range (fcsio.header.Header attribute), 6  
 DATATYPE (fcsio.text.standard.Standard attribute), 9  
 delete() (fcsio.text.parameters.Parameters method), 13  
 detector\_type (fcsio.text.parameters.Parameter attribute), 11  
 detector\_voltage (fcsio.text.parameters.Parameter attribute), 12

## E

emitted\_light (fcsio.text.parameters.Parameter attribute), 12  
 end (fcsio.header.ByteIndecies attribute), 6  
 ENDANALYSIS (fcsio.text.standard.Standard attribute), 9

ENDDATA (fcsio.text.standard.Standard attribute), 9  
 ENDSTEXT (fcsio.text.standard.Standard attribute), 10  
 event\_count (fcsio.data.Data attribute), 14  
 events() (fcsio.filter.Filter method), 5  
 excitation\_power (fcsio.text.parameters.Parameter attribute), 12  
 excitation\_wavelengths (fcsio.text.parameters.Parameter attribute), 12

## F

FCS (class in fcsio), 1  
 fcs\_bytes (fcsio.FCSFactory attribute), 4  
 FCSFactory (class in fcsio), 3  
 fcsio (module), 1  
 fcsio.data (module), 14  
 fcsio.filter (module), 5  
 fcsio.header (module), 5  
 fcsio.text (module), 7  
 fcsio.text.parameters (module), 10  
 fcsio.text.standard (module), 8  
 Filter (class in fcsio.filter), 5  
 filter (fcsio.FCS attribute), 1

## G

gate() (fcsio.filter.Filter method), 5  
 get\_keywords() (fcsio.text.parameters.Parameter method), 12  
 get\_required\_keywords() (in module fcsio.text), 8

## H

Header (class in fcsio.header), 6  
 header\_bytes (fcsio.FCSFactory attribute), 4

## I

in\_header (fcsio.header.ByteIndecies attribute), 6  
 index (fcsio.text.parameters.Parameter attribute), 12  
 index() (fcsio.header.ByteIndecies method), 6  
 index() (fcsio.text.RegexDescriptor method), 8  
 indexOf() (fcsio.text.parameters.Parameters method), 13

## K

keys() (fcsio.text.KeyWordDict method), 7

keys() (fcsio.text.parameters.Parameter method), 12  
keys() (fcsio.text.Text method), 8  
keyword (fcsio.text.RegexDescriptor attribute), 8  
KeyWordDict (class in fcsio.text), 7

## L

long\_name (fcsio.text.parameters.Parameter attribute), 12

## M

matrix (fcsio.data.Data attribute), 14  
minimize() (fcsio.filter.Filter method), 5  
miter (fcsio.text.Text attribute), 8  
MODE (fcsio.text.standard.Standard attribute), 10

## N

NEXTDATA (fcsio.text.standard.Standard attribute), 10  
none() (fcsio.filter.Filter method), 5

## O

optical\_filter (fcsio.text.parameters.Parameter attribute), 12  
other (fcsio.FCS attribute), 2  
other (fcsio.FCSFactory attribute), 4  
other\_ranges (fcsio.header.Header attribute), 6  
output\_constructor() (fcsio.FCS method), 2

## P

PAR (fcsio.text.standard.Standard attribute), 10  
Parameter (class in fcsio.text.parameters), 10  
parameter\_data (fcsio.text.KeyWordDict attribute), 7  
parameter\_data (fcsio.text.Text attribute), 8  
Parameters (class in fcsio.text.parameters), 13  
parameters (fcsio.FCS attribute), 2  
parameters (fcsio.FCSFactory attribute), 4  
parameters (fcsio.text.parameters.Parameters attribute), 14  
parameters() (fcsio.filter.Filter method), 5

## R

range (fcsio.text.parameters.Parameter attribute), 13  
regex (fcsio.text.RegexDescriptor attribute), 8  
regex\_string (fcsio.text.RegexDescriptor attribute), 8  
RegexDescriptor (class in fcsio.text), 7

## S

short\_name (fcsio.text.parameters.Parameter attribute), 13  
Standard (class in fcsio.text.standard), 8  
standard (fcsio.FCS attribute), 2  
standard (fcsio.FCSFactory attribute), 4  
start (fcsio.header.ByteIndices attribute), 6

## T

Text (class in fcsio.text), 8  
text (fcsio.FCS attribute), 3  
text (fcsio.FCSFactory attribute), 4  
text\_bytes (fcsio.FCSFactory attribute), 5  
text\_range (fcsio.header.Header attribute), 6  
TOT (fcsio.text.standard.Standard attribute), 10

## V

validate() (fcsio.header.Header method), 7  
version (fcsio.FCS attribute), 3  
version (fcsio.header.Header attribute), 7  
visualization\_scale (fcsio.text.parameters.Parameter attribute), 13