# biorad1sc_reader Documentation

***Release 0.4***

**Matthew A. Clapp**

**Nov 30, 2020**

# Contents:

# CHAPTER 1

# Introduction

biorad1sc_reader is a python3 module designed to allow reading, parsing and extracting images and metadata from Bio-Rad *.1sc files, notably from Quantity One software.

For information on the file format for 1sc files, please see the document File Specification for Bio-Rad 1sc Files

## 1.1 License Information

The package biorad1sc_reader is licensed under the MIT License.

```
MIT License

Copyright (c) 2017 Matthew A. Clapp

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

# CHAPTER 2

# Tutorial

Reading a 1sc file starts with importing the `biorad1sc_reader` package

```
import biorad1sc_reader
```

Then to access the data of a given 1sc file, we instance the class `Reader`

```
myreader = biorad1sc_reader.Reader()
```

For convenience, you can specify the name of the file to read as an argument of the class initialization

```
myreader = biorad1sc_reader.Reader("path/to/some/file.1sc")
```

You can also initialize the class with a file-like object set for read/binary access:

```
my1sc_fh = open("path/to/some/file.1sc", 'rb')
myreader = biorad1sc_reader.Reader(my1sc_fh)
```

After you instance the class `Reader` into your own variable, you can use that to access and decode the 1sc file's data.

For example, to get a succinct data structure of all metadata in 1sc file:

```
my_img_metadata = myreader.get_metadata_compact()
```

To save the image data as a 16-bit TIFF with no processing, use:

```
myreader.save_img_as_tif("exactly_as_in_1sc.tif")
```

# CHAPTER 3

# Examples

```
# import library
import bioradlsc_reader

# setup reader with input file
myreader = bioradlsc_reader.Reader("my_biorad_file.1sc")

# setup reader with file-like object
my2sc_fh = open("my_biorad_file2.1sc", 'rb')
myreader2 = bioradlsc_reader.Reader(my2sc_fh)

# get list/dict of all metadata in 1sc file
my_img_metadata = myreader.get_metadata()

# get a more succinct data structure of all metadata in 1sc file
my_img_metadata = myreader.get_metadata_compact()

# get a quick summary of some metadata about the image in the 1sc file
my_img_metadata = myreader.get_img_summary()

# Different options for writing image data out as a TIFF file
myreader.save_img_as_tif("unscaled_brightness.tif")
myreader.save_img_as_tif("unscaled_inverted_brightness.tif", invert=True)
myreader.save_img_as_tif_sc("scaled_brightness.tif")
myreader.save_img_as_tif_sc("scaled_brightness_more.tif", scale=0.8)
myreader.save_img_as_tif_sc("scaled_inverted_brightness.tif", invert=True)
```

CHAPTER 4

# Command-line Utilities

Accompanying this python package are the following command-line utilities, which enable some of the same tasks to be accomplished without writing python code.

## 4.1 bio1sc2tiff

Convert input 1sc file(s) to TIFF image(s).

### 4.1.1 Usage

```
bio1sc2tiff [-h] [-s] [-i] [-o OUTPUT_FILENAME] src_1sc_file [src_1sc_file
...]
```

### 4.1.2 Positional Arguments

**src_1sc_file** Source 1sc file.

### 4.1.3 Optional Arguments

**-h, --help** show this help message and exit

**-s, --scale** Scale brightness of output image to maximize dynamic range between darkest and lightest pixels in input file.

**-i, --invert** Invert brightness scale of image.

**-o OUTPUT_FILENAME, --output_filename OUTPUT_FILENAME** Name of output image. (Defaults to <input_image>.tif) in same directory as source file.

## 4.2 bio1scmeta

Print all metadata contained in 1sc file(s).

### 4.2.1 Usage

```
bio1scmeta [-h] [-v VERBOSITY] [-o OUTPUT_FILENAME] src_1sc_file [src_1sc_file
...]
```

### 4.2.2 Positional Arguments

**src_1sc_file** Source 1sc file.

### 4.2.3 Optional Arguments

**-h, --help** show this help message and exit

**-v VERBOSITY, --verbosity VERBOSITY** Verbosity of report, number, 0, 1, or 2 (default 0).

**-o OUTPUT_FILENAME, --output_filename OUTPUT_FILENAME** Name of output text file. (Defaults to <filename>_meta.txt in same directory as source file.

---

## 4.3 bio1scread

Read/Parse Bio-Rad *.1sc file(s) and produce reports detailing their internal structure. Reports for <filename>.1sc are placed in <filename>_reports directory.

### 4.3.1 Usage

```
bio1scread [-h] [-S] srcfile [srcfile ...]
```

### 4.3.2 Positional Arguments

**srcfile** Source 1sc file(s).

### 4.3.3 Optional Arguments

**-h, --help** show this help message and exit

**-S, --omit_strings** Do not include Type 16 String fields in reports. (But include the strings when listing references to them.)

# CHAPTER 5

# API Reference

This is the public interface to the biorad1sc_reader functionality.

## 5.1 biorad1sc_reader.Reader

**class** biorad1sc_reader.**Reader**(*in_file=None*)

Object to manage reading a Bio-Rad 1sc file and extracting data from it, including image.

Assumes the 1sc file does not change while this instance has it open.

**Instantiation:**

**Args:**

**in_file (str or file-like obj): filepath (str) or file-like** object, 1sc file to read with this instance

**Raises:** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**get_img_data**(*invert=False*)

Return image_x_size, image_y_size, and list containing image data. Also ability to invert brightness.

**Parameters** **invert** (*bool, optional*) – True to invert the brightness scale of output image data compared to 1sc image data (black <-> white)

**Returns**

(xsize, ysize, image_data) where xsize and ysize are integers specifying the size of the image.

image_data is a list of uint16 numbers comprising the image data starting from upper-left and progressing to lower-right.

**Return type** tuple

**get_img_summary**()

NOTE: Safer to use get_metadata() or get_metadata_compact()

Read from Data Block 7, containing strings describing image.

**Returns**

dict containing data from strings in Data Block 7:

```
{
    'Scanner Name':'ChemiDoc XRS'
    'Number of Pixels':'(<x pix size> x <y pix size>)'
    'Image Area':'(<x float size> mm x <y float size> mm)'
    'Scan Memory Size': '<size in bytes>'
    'Old file name': '<orig file name>'
    'New file name': '<new file name>'
    'path':'CHEMIDOC\Chemi'
    'New Image Acquired':'New Image Acquired'
    'Save As...':'Save As...'
    'Quantity One':'Quantity One <version> build <build number>'
}
```

**Return type** dict

**get_metadata**()

Fetch All Metadata in File, return hierarchical dict/list

**Returns**

collections where each item in list collections is a dict:

```
collection_dict = {
    'data':<list items>
    'label':'<str name of collection>'
}
```

where items is a list of dicts, each with the structure:

```
item_dict = {
    'data':<list regions>
    'id':<uint32 Field ID>
    'label':'<str name of item>'
    'type':'<int Field Type>'
}
```

where regions is a list of dicts, each with the structure:

```
region_dict = {
    'data': <dict data_of_region>
    'dtype': <str written type of data>
    'dtype_num': <int data type code of data>
    'key_iter': <??>
    'label': <str name of region>
    'num_words': <int number of words in data>
    'region_idx': <int 1sc-given index>
    'word_size': <int number of bytes per word of data>
}
```

where data_of_region has the structure:

```
data_of_region = {
    'raw': <bytes raw bytes, unconverted data>
    'proc': <various unpacked/decoded data from bytes>
    'interp': <various 'interpreted' data>
}
```

**data_of_region['interp'] can also be another item_dict, if this** region contained a reference to another field, creating a hierarchical structure.

e.g. `collections[0]['data'][0]['data'][0]['label'] = 'array'`

**Return type** list

**Raises** `BioRadParsingError` – if there was an error in parsing the file

**get_metadata_compact**()

Fetch All Metadata in File, return compact version of hierarchical dict/list

Convert dict(list()) of Collections, Items to dict(). Leave Regions as list, because they are not guaranteed to have unique labels.

Remove everything except 'label' and most-interpreted form of 'data' available.

**Returns**

collections:

```
collections = {
    '<collection name1>':<dict collection_dict1>
    '<collection name2>':<dict collection_dict2>
    ...
}
```

where each collection_dict is:

```
collection_dict = {
    '<name of item1>':<list regions1>
    '<name of item2>':<list regions2>
    ...
}
```

where regions is a list of dicts, each with the structure:

```
region_dict = {
    'data': <various most interpreted version possible of data>
    'label': <str name of region>
}
```

**region_dict['data'] can also be another regions list, if this** region contained a reference to another field, creating a hierarchical structure.

e.g. `collections['Overlay Header']['OverlaySaveArray'][0]['label] = 'array'`

**Return type** dict

**open_file**(*in_filename*)

Open file and read into memory.

Raises Errors if File is not valid 1sc file.

**Parameters** **in_filename** (*str*) – filepath to 1sc file to read with object instance

**Raises** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**read_stream**(*in_fh*)

Read file-like object into memory.

Raises Errors if File is not valid 1sc file. Give it object returned by: open(<filename>, 'rb')

**Parameters** **in_fh** (*byte stream*) – filehandle to 1sc filedata to read with object instance. e.g. result from open(<filename>, 'rb')

**Raises** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**refresh**()
Reset and refresh all internal state using same input 1sc file.

**reset**()
Reset all internal state. (Must load file afterwards.)

**save_img_as_tiff**(*tiff_filename*, *invert=False*)
Save image data as TIFF image

Also ability to invert brightness

**Parameters**

- **tiff_filename** (*str*) – filepath for output TIFF file
- **invert** (*bool, optional*) – True to invert the brightness scale of output TIFF image compared to 1sc image data (black <-> white)

**save_img_as_tiff_sc**(*tiff_filename*, *imgsc=1.0*, *invert=False*)
Save image data as TIFF image, with brightness dynamic range expanded

Also ability to invert brightness

**Parameters**

- **tiff_filename** (*str*) – filepath for output TIFF file
- **imgsc** (*float, optional*) – Expand brightness scale. Value of 1.0 means that dynamic range of output TIFF will be maximum, with brightest pixel having value 65535 and darkest pixel having value 0.

  imgsc > 1.0 will cause the brightness dynamic range to be expanded less than imgsc=1.0, and imgsc < 1.0 will cause the dynamic range to be expanded more than the imgsc=1.0 case.

  For non-inverted images, the pixel with the minimum brightness will always be 0. For inverted images, the pixel with the maximum brightness will always be 65535.
- **invert** (*bool, optional*) – True to invert the brightness scale of output TIFF image compared to 1sc image data (black <-> white)

# CHAPTER 6

# Internal Code Reference

The code detailed here is not for users of the module, but as an aid to development of the module itself.

For regular usage, see Section *API Reference* .

## 6.1 biorad1sc_reader.parsing

Low-level routines to parse a Bio-Rad 1sc file. Intended to be used internally only.

bioradlsc_reader.parsing.**fix_wordsize_zero**(*field_payload_regions*, *byte_offsets*, *data_key_total_bytes*)

Fix Datakey data when Field Type 100 doesn't list word_size

In certain 1sc files, the word_size sub_field of Field Type 100 can be 0. This function detects regions in field_payload_regions dict for word_size==0 and changes word_size to the appropriate number of bytes based on data_type codes.

field_payload_regions is modified in place.

**Parameters**

- **field_payload_regions** (*dict*) – dict containing region info
- **byte_offsets** (*list*) – all starting byte offsets for all regions
- **data_key_total_bytes** (*int*) – total number of bytes in data container that field_payload_regions is defining

bioradlsc_reader.parsing.**is_ascii**(*byte_stream*)

Determine if all bytes in a bytes object are "good" ASCII

If all bytes are normal ascii text with no control characters other than NULL, LF, CR, TAB, then return True, else False.

**Parameters** **byte_stream** (*bytes*) – arbitrary length

**Returns** True if all bytes are printable ASCII codes or one of the following ASCII codes: 0 (null), 9 (Tab), 10 (LF), 13 (CR); False otherwise.

**Return type** bool

bioradlsc_reader.parsing.**process_data_region**(*region*, *payload*, *field_ids*, *field_types*, *visited_ids*)

Process one region of one data container field.

**Parameters**

- **region** (*dict*) – info from datakey about the format of this region
- **payload** (*bytes*) – bytes of the payload just for this region
- **field_ids** (*dict*) – keys are Field IDs, items are dicts containing all data for that Field instance
- **field_types** (*dict*) – explanation of each Field Type from 'items' returned from process_payload_type101
- **visited_ids** (*list*) – uint32 Field IDs of fields that have been visited

**Returns**

comprised of the following structure:

```
region = {
    'raw': <bytes raw data from payload>
    'proc': <various unpacked/decoded numbers/strings from raw data>
    'interp': <various interpreted version of proc data, or None
                if no interpretation possible. can also be list
                of another field's regions if region data is a
                reference to another field>
}
```

**Return type** dict

bioradlsc_reader.parsing.**process_payload_data_container**(*field_info*, *field_types*, *field_ids*, *visited_ids*)

Process the payload of a 1sc data container field.

Process the payload of a 1sc Field Type > 102, (a data container field,) returning the relevant data to a dict.

**Parameters**

- **field_info** (*dict*) – contains info about current field
- **field_types** (*dict*) – explanation of each Field Type from 'items' returned from process_payload_type101
- **field_ids** (*dict*) – keys are Field IDs, items are dicts containing all data for that Field instance
- **visited_ids** (*list*) – keeps track of all Field IDs that have been processed into the hierarchical output data

**Returns**

regions, where each item of list is a dict of the form:

```
region = {
    'raw': <bytes raw data from payload>
    'proc': <various unpacked/decoded numbers/strings from raw data>
    'interp': <various interpreted version of proc data, or None
                if no interpretation possible. can also be list
                of another field's regions if region data is a
```

(continues on next page)

(continued from previous page)

```
                reference to another field>
}
```

**Return type** list

bioread1sc_reader.parsing.**process_payload_type100**(*field_payload*, *data_key_total_bytes*, *field_ids=None*)

Process the payload of a 1sc Field Type 100

Process the payload of a 1sc Field Type 100, a description of the format of a particular Field Type of data container field. Return dict of dict containing region info.

**Parameters**

- **field_payload** (*bytes*) – all the contents of a Field Type 100 after the header bytes
- **field_ids** (*dict*) – keys of Field IDs (uint32 numbers) and items which are dicts containing all information on that field instance

**Returns**

info dict, with the form:

```
info = {
    'regions':<dict regions>
}
```

where dict regions is in the form of:

```
regions = {
    <number1>:<dict region1>,
    <number2>:<dict region2>,
    ...
}
```

where each key <number> is a number from 0 to Total Regions - 1 where each dict <region> is in the form of:

```
region = {
    'data_type': <uint16 number coding for data type of region>,
    'label': <str name of region>,
    'index': <int index that orders data regions>,
    'num_words': <int number of words in region>,
    'byte_offset': <int byte offset from start of Data Container␣
↪payload>,
    'word_size':<int number of bytes in each word>,
    'ref_field_type':<uint16 Field Type of ref. if data_type is ref.>,
}
```

**Return type** dict

bioread1sc_reader.parsing.**process_payload_type101**(*field_payload*, *field_ids=None*)

Process the payload of a 1sc Field Type 101

Process the payload of a 1sc Field Type 101, a summary of every type of Data Container type available in this Data Collection, and return information dict.

**Parameters**

- **field_payload** (*bytes*) – all the contents of a Field Type 101 after the header bytes

- **field_ids** (*dict*) – keys of Field IDs (uint32 numbers) and items which are dicts containing all information on that field instance

**Returns**

dict containing a summary all data_container item types possible for the associated Collection in the form:

```
collection_item_definitions {
    'items':<int tot_items>,
    <Field1 Type>:<dict item_info1>,
    <Field2 Type>:<dict item_info2>,
    ...
}
```

where each key <Field Type> is the uint16 Field Type of a Data Container field that is possibly found after this definition in the next Data Block. Each <item_info> gives a summary of how to process a possible future data container field:

```
item_info {
    'num_regions': <int number of regions>,
    'data_key_ref': <uint32 Field ID of a Field Type 100>,
    'total_bytes': <int total bytes in region>,
    'label': <str name of item>,
}
```

**Return type** dict

biorad1sc_reader.parsing.**process_payload_type102**(*field_payload*, *field_ids=None*)

Process the payload of a 1sc Field Type 102

Process the payload of a 1sc Field Type 102, a Collection definition, and return and informational dict.

**Parameters**

- **field_payload** (*bytes*) – all the contents of a Field Type 102 after the header bytes
- **field_ids** (*dict*) – keys of Field IDs (uint32 numbers) and items which are dicts containing all information on that field instance

**Returns**

collection_info with the form:

```
collection_info {
    'collection_num_items': <int num of items in collection>,
    'collection_label': <str name of collection>,
    'collection_ref': <uint32 Field ID of a Field Type 101>
}
```

**Return type** dict

biorad1sc_reader.parsing.**unpack_double**(*byte_stream*, *endian='<'*)

Return list of doubles, (either endian) from bytestring.

Unpack a bytes object into list of double-precision floating-point numbers. Each 8 input bytes decodes to a double.

**Parameters**

- **byte_stream** (*bytes*) – length is a multiple of 8

- **endian** (*char, optional*) – "<" means little-endian unpacking, and ">" means big-endian unpacking

**Returns** unpacked double numbers

**Return type** list

bioradlsc_reader.parsing.**unpack_string**(*byte_stream*)
Return decoded ASCII string from bytestring.

Decode a bytes object via UTF-8 into a string

**Parameters** **byte_stream** (*bytes*) – arbitrary length

**Returns** UTF-8 decoded string

**Return type** string

bioradlsc_reader.parsing.**unpack_uint16**(*byte_stream*, *endian='<'*)
Return list of uint16s, (either endian) from bytes object.

Unpack a bytes object into list of 16-bit unsigned integers. Each 2 input bytes decodes to a uint16.

**Parameters**

- **byte_stream** (*bytes*) – length is a multiple of 2
- **endian** (*char, optional*) – "<" means little-endian unpacking, and ">" means big-endian unpacking

**Returns** unpacked uint16 numbers

**Return type** list

bioradlsc_reader.parsing.**unpack_uint32**(*byte_stream*, *endian='<'*)
Return list of uint32s, (either endian) from bytes object.

Unpack a bytes object into list of 32-bit unsigned integers. Each 4 input bytes decodes to a uint32.

**Parameters**

- **byte_stream** (*bytes*) – length is a multiple of 4
- **endian** (*char, optional*) – "<" means little-endian unpacking, and ">" means big-endian unpacking

**Returns** unpacked uint32 numbers

**Return type** list

bioradlsc_reader.parsing.**unpack_uint64**(*byte_stream*, *endian='<'*)
Return list of uint64s, (either endian) from bytes object.

Unpack a bytes object into list of 64-bit unsigned integers. Each 8 input bytes decodes to a uint64.

**Parameters**

- **byte_stream** (*bytes*) – length is a multiple of 8
- **endian** (*char, optional*) – "<" means little-endian unpacking, and ">" means big-endian unpacking

**Returns** unpacked uint64 numbers

**Return type** list

## 6.2 biorad1sc_reader.reader

Main reader module for Bio-Rad 1sc files. Includes public API class Reader.

**class** bioradlsc_reader.reader.**Reader**(*in_file=None*)

Object to manage reading a Bio-Rad 1sc file and extracting data from it, including image.

Assumes the 1sc file does not change while this instance has it open.

**Instantiation:**

**Args:**

**in_file (str or file-like obj): filepath (str) or file-like** object, 1sc file to read with this instance

**Raises:** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**__init__**(*in_file=None*)

Initialize Reader class

**Parameters** **in_file** (*str or file-like obj*) – filepath (str) or file-like object, 1sc file to read with this instance

**Raises** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**__weakref__**

list of weak references to the object (if defined)

**_first_region**(*item*, *region_name*)

Fetch data from first region in item with name region_name

Convenience function to fetch data for the first region with label region_name in item

**Parameters**

- **item** (*list*) – list of regions from get_metadata() or get_metadata_compac()
- **region_name** (*str*) – region label to search for

**Returns** whatever is contained in 'data' key of region dict

**Return type** various

**_get_img_size**()

Get img_size x and y, load into instance

Determine image size from metadata of 1sc file and set internal instance attributes self.img_size_x and self.img_size_y

**_get_next_data_block_end**(*byte_idx*)

Given a byte index, find the next Data Block end, return byte at start of the following Data Block

**Parameters** **byte_idx** (*int*) – file byte offset to search for the end of the next Data Block

**Returns**

**(block_num, end_idx) where block_num is the Data Block** that ends at end_idx-1

**Return type** tuple

**_make_compact_item**(*item*)

Given an Item from metadata data collection, return a compact version of it for use in get_metadata_compact()

Remove everything except 'label' and most-interpreted form of 'data' available

**Parameters** **item** (*dict*) – item_dict from get_metadata(), e.g. collections[<num>]['data'][<num>]

**Returns**

**compact representation of input dict for** get_metadata_compact()

**Return type** dict

**_parse_file_header**()
Read and process the start of the file (header)

**Raises** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**_process_field_header**(*byte_idx*)

**Parameters** **byte_idx** (*int*) – file byte offset, start of the field to read header

**Returns**

**(field_type, field_len, field_id) where field_type is** uint16 Field Type, field_len is int length in bytes of field, field_id is uint32 Field ID

**Return type** tuple

**_read_field_lite**(*byte_idx*)

**Parameters** **byte_idx** (*int*) – file byte offset, start of the field to read

**Returns**

**(file_byte_offset_next_field, field_info) where field info** is a dict:

```
{
    'type':<uint16 Field Type>
    'id':<uint32 Field ID>
    'start':<byte offset of start of field>
    'len':<total length in bytes of field>
    'payload':<field payload bytes>
}
```

**Return type** tuple

**get_img_data**(*invert=False*)
Return image_x_size, image_y_size, and list containing image data. Also ability to invert brightness.

**Parameters** **invert** (*bool, optional*) – True to invert the brightness scale of output image data compared to 1sc image data (black <-> white)

**Returns**

(xsize, ysize, image_data) where xsize and ysize are integers specifying the size of the image.

image_data is a list of uint16 numbers comprising the image data starting from upper-left and progressing to lower-right.

**Return type** tuple

**get_img_summary**()
NOTE: Safer to use get_metadata() or get_metadata_compact()

Read from Data Block 7, containing strings describing image.

**Returns**

dict containing data from strings in Data Block 7:

```
{
    'Scanner Name':'ChemiDoc XRS'
    'Number of Pixels':'(<x pix size> x <y pix size>)'
    'Image Area':'(<x float size> mm x <y float size> mm)'
    'Scan Memory Size': '<size in bytes>'
    'Old file name': '<orig file name>'
    'New file name': '<new file name>'
    'path':'CHEMIDOC\Chemi'
    'New Image Acquired':'New Image Acquired'
    'Save As...':'Save As...'
    'Quantity One':'Quantity One <version> build <build number>'
}
```

**Return type** dict

**get_metadata**()

Fetch All Metadata in File, return hierarchical dict/list

**Returns**

collections where each item in list collections is a dict:

```
collection_dict = {
    'data':<list items>
    'label':'<str name of collection>'
}
```

where items is a list of dicts, each with the structure:

```
item_dict = {
    'data':<list regions>
    'id':<uint32 Field ID>
    'label':'<str name of item>'
    'type':'<int Field Type>'
}
```

where regions is a list of dicts, each with the structure:

```
region_dict = {
    'data': <dict data_of_region>
    'dtype': <str written type of data>
    'dtype_num': <int data type code of data>
    'key_iter': <??>
    'label': <str name of region>
    'num_words': <int number of words in data>
    'region_idx': <int 1sc-given index>
    'word_size': <int number of bytes per word of data>
}
```

where data_of_region has the structure:

```
data_of_region = {
    'raw': <bytes raw bytes, unconverted data>
    'proc': <various unpacked/decoded data from bytes>
    'interp': <various 'interpreted' data>
}
```

**data_of_region['interp'] can also be another item_dict, if this** region contained a refer-

ence to another field, creating a hierarchical structure.

e.g. `collections[0]['data'][0]['data'][0]['label'] = 'array'`

**Return type** list

**Raises** `BioRadParsingError` – if there was an error in parsing the file

**get_metadata_compact**()

Fetch All Metadata in File, return compact version of hierarchical dict/list

Convert dict(list()) of Collections, Items to dict(). Leave Regions as list, because they are not guaranteed to have unique labels.

Remove everything except 'label' and most-interpreted form of 'data' available.

**Returns**

collections:

```
collections = {
    '<collection name1>':<dict collection_dict1>
    '<collection name2>':<dict collection_dict2>
    ...
}
```

where each collection_dict is:

```
collection_dict = {
    '<name of item1>':<list regions1>
    '<name of item2>':<list regions2>
    ...
}
```

where regions is a list of dicts, each with the structure:

```
region_dict = {
    'data': <various most interpreted version possible of data>
    'label': <str name of region>
}
```

**region_dict['data'] can also be another regions list, if this** region contained a reference to another field, creating a hierarchical structure.

e.g. `collections['Overlay Header']['OverlaySaveArray'][0]['label] = 'array'`

**Return type** dict

**open_file**(*in_filename*)

Open file and read into memory.

Raises Errors if File is not valid 1sc file.

**Parameters** **in_filename** (*str*) – filepath to 1sc file to read with object instance

**Raises** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**read_stream**(*in_fh*)

Read file-like object into memory.

Raises Errors if File is not valid 1sc file. Give it object returned by: open(<filename>, 'rb')

**Parameters** **in_fh** (*byte stream*) – filehandle to 1sc filedata to read with object instance. e.g. result from open(<filename>, 'rb')

**Raises** BioRadInvalidFileError if file is not a valid Bio-Rad 1sc file

**refresh**()
Reset and refresh all internal state using same input 1sc file.

**reset**()
Reset all internal state. (Must load file afterwards.)

**save_img_as_tiff**(*tiff_filename*, *invert=False*)
Save image data as TIFF image

Also ability to invert brightness

**Parameters**

- **tiff_filename** (*str*) – filepath for output TIFF file
- **invert** (*bool, optional*) – True to invert the brightness scale of output TIFF image compared to 1sc image data (black <-> white)

**save_img_as_tiff_sc**(*tiff_filename*, *imgsc=1.0*, *invert=False*)
Save image data as TIFF image, with brightness dynamic range expanded

Also ability to invert brightness

**Parameters**

- **tiff_filename** (*str*) – filepath for output TIFF file
- **imgsc** (*float, optional*) – Expand brightness scale. Value of 1.0 means that dynamic range of output TIFF will be maximum, with brightest pixel having value 65535 and darkest pixel having value 0.

  imgsc > 1.0 will cause the brightness dynamic range to be expanded less than imgsc=1.0, and imgsc < 1.0 will cause the dynamic range to be expanded more than the imgsc=1.0 case.

  For non-inverted images, the pixel with the minimum brightness will always be 0. For inverted images, the pixel with the maximum brightness will always be 65535.
- **invert** (*bool, optional*) – True to invert the brightness scale of output TIFF image compared to 1sc image data (black <-> white)

bioradlsc_reader.reader.**save_u16_to_tiff**(*u16in*, *size*, *tiff_filename*)
Save 16-bit uints to TIFF image file

Since Pillow has poor support for 16-bit TIFF, we make our own save function to properly save a 16-bit TIFF.

**Parameters**

- **u16in** (*list*) – u16int image pixel data
- **size** (*tuple*) – (xsize, ysize) where xsize and ysize are integers specifying the size of the image in pixels
- **tiff_filename** (*str*) – filepath for the output TIFF file

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## b

# Index

## Symbols



## B



## F



## G



## I



## O



## P



## R

## S



## U