SPIF–Single Particle Image Format

The SPIF Working Group

March 2021
v0.86

Abstract

This document defines the Single Particle Image Format (SPIF) file structure and and variables. The SPIF file is a NetCDF file and a universal file format for single particle image probe data.

Contents

1 Motivation

A variety of file formats are used by particle imaging instrument manufacturers. The formats are dictated by hardware and bandwidth limitations, and so may not have a convenient or conventional structure. However, after data has been acquired, and as hardware improves, the imposed limitations vanish. We propose a new file format for these imaging probes. The new format will specify a single structure which all imaging probe data formats may be mapped onto, leading to improved accessibility for users and refinement of data processing routines. We aim for this format to be adopted by instrument manufacturers. The format is also suitable for representing historical datasets and is suitable for long term archiving of raw data. File converters for some of the most commonly used probes are provided.

2 Description

The SPIF file format uses the NetCDF4 format. NetCDF4 is a structured binary file format capable of containing large datasets and has automatic compression utilities. NetCDF4 is widely supported on a variety of platforms and environments.

Datasets will be contained in groups within a SPIF file. Only data from a single instrument (or instrument channel) may be stored in a given group although SPIF files can contain data from multiple instruments through the use of multiple groups. Metadata will be provided for each instrument to provide information on data sources etc.

3 SPIF File Definition

root The root of the file contains global attributes.

instrument group Each instrument data included within the file has a separate group.

- **root** Instrument metadata
- **core** The SPIF-Core group contains the data necessary for subsequent analysis
- **aux** The SPIF-Aux group contains additional data which is generated by a given instrument, but is not essential for data processing. This data is included to maintain integrity of the original dataset, making SPIF a suitable format for long term archiving.
- **lvlX** Processed data may be included as a subgroup of either `core` or `aux` or as individual variables within the same group but with an appropriate `level` attribute.

3.1 SPIF Root

The root of the SPIF file has all of the global data and attributes relating to the production of the file.

char `title` = "SPIF–Single Particle Image Format";
char `institution` = Institution that produced the file;
char `version` = Conventions being used – SPIF-1.0, if following this document;
char `references` = Link to web, paper, document reference describing datasets, project, etc;
char `history` = creation/modification datetime, version number;
char `comment` = datetime, comment, changelog(?);

char `project` = Name of project;
char `start_date` = Reference date of all datasets;
char `conventions` = Conventions used in file; e.g., SPIF-0.86;

3.2 Instrument Group

The root of the instrument group contains information pertaining to the individual instrument. It may also include information about the probe software module that was used to create it, this may be different to the version of the overall processing code.

Some probes will have multiple channels, for example the 2DS has two orthogonal arms, with each detecting particles independently, essentially behaving as independent instruments. Since the primary coordinate applies only to particle sequences from a single instrument, for a probe that is made up of multiple instruments (e.g., 2DS, 3V-CPI, etc) where the particle detection will not be coincident, data is split into different instrument groups. These groups should be named using the core instrument name, followed by the channel name, separated by a dash. So for a 2DS instrument, the Instrument Groups would be named '2DS-H' and '2DS-V'.

Metadata that should be defined in the Instrument group include:

char `instrument_name` = Short name of instrument;
char `instrument_long_name` = Full descriptive name of instrument;
char `instrument_channel` = Instrument channel (if applicable);
char `institution` = Institution operating instrument;
char `reference` = Link to web, paper, document reference describing instrument;
char `instrument_serial_number` = Serial number or instrument identifier;
char `manufacturer` = Manufacturer of instrument;

char `instrument_firmware` = Instrument firmware version;
char `instrument_software` = Instrument software version;
char `platform` = Name or description of platform instrument is mounted on;
array of char `raw_filenames` = Raw data filename(s) used to generate the current instrument dataset;
char `spif_processor` = Name and version of code used to generate SPIF output;
char `comment` = Any further notes about instrument, platform, location, orientation, etc;

Although envisaged to be used exclusively with optical array probes, there is no reason that this format could not be used for any instrument type. Therefore there may not be suitable universal descriptive attributes for all instrument types. It may thus be desirable that these attributes are placed in a sub-group although this may add a layer of complexity for little gain. Below are variables which may be used for an optical array probe.

int `pixels`()

`pixels:long_name` =	Numbers of pixels in `instrument`;
`pixels:comment` =	Any relevant comments;

float `resolution`(scalar or 2)

`resolution:long_name` =	Physical resolution of array pixels `instrument`;
`resolution:units` =	“micrometer”;
`resolution:ancillary_variables` =	`instrument:resolution_err`;
`resolution:comment` =	Any relevant comments;

float `resolution_err`(scalar or 2)

`resolution_err:long_name` =	Uncertainty of physical resolution of array pixels `instrument`;
`resolution_err:units` =	`instrument:resolution`;
`resolution_err:comment` =	Any relevant comments;

float `arm_separation`

`arm_separation:long_name` =
Physical distance between probe arms;
`arm_separation:units` =
"millimeter";
`arm_separation:ancillary_variables` =
`instrument:arm_separation_err`;
`arm_separation:comment` =
Any relevant comments;

float `arm_separation_err`

`arm_separation_err:long_name` =
Uncertainty of physical distance between probe arms;
`arm_separation_err:units` =
`instrument:arm_separation`;
`arm_separation_err:comment` =
Any relevant comments;

boolean `antishatter_tips`

`antishatter_tips:long_name` =
Use of antishatter-, or Korolev-, tips on probe arms;
`antishatter_tips:comment` =
Any relevant comments;

int `bpp`

`bpp:long_name` = Bits per pixel used in image data;

`bpp:comment` = Any relevant comments;

int `wavelength`

`wavelength:long_name` =
Laser wavelength of instrument;
`wavelength:units` =
"nm";
`wavelength:comment` =
Any relevant comments;

3.3 Instrument Core

The Instrument Core group contains data necessary for analysis and processing of the raw data. There is a single coordinate for all variables in this group. This means that some of the arrays are larger than would otherwise be the case, but it also means that all the arrays are the same length which makes for more efficient storage, chunking, and retrieval. The coordinate is *images*, the length of `image_ns` defines the length of the coordinate dimension, where the arrival time of the *n*th image is given by `image_ns`(n) + `image_sec`(n).

There are two additional dimensions defined for the image data – *slices* and *array*. The *slices* dimension corresponds to the image dimension in the direction of flight, and is set as the maximum number of slices encountered (or allowed) by the instrument. The *array* dimension corresponds to the image dimension along the diode array, and is set as the number of diodes of the instrument.

The instrument core data group consists of:

float `image_ns`(*images*)

`image_ns:long_name` = "image arrival time in nanoseconds";
`image_ns:units` = ns since `image_sec`;
`image_ns:ancillary_variables` = `image_sec`;
`image_ns:comment` = Any relevant comments;

int `image_sec`(*images*)

`image_sec:standard_name` = "time";
`image_sec:long_name` = "image arrival time in seconds";
`image_sec:timezone` = "UTC";
`image_sec:units` = "seconds since `start_date` 00:00:00 +0000";
`image_sec:strftime_format` = "%F %T %z";
`image_sec:comment` = Any relevant comments;

int `image_len`(*images*)

`image_len:long_name` = "image event length in number of slices";
`image_len:units` = "number of slices";
`image_len:comment` = Any relevant comments;

uint8 `image`(*images*, *slices*, *array*)

`image:long_name` = "Image array";
`image:comment` = Any relevant comments;

int `buffer_index`(*images*)

```
  buffer_index:long_name =
```
"index of data buffer from which image was extracted";
```
  buffer_index:units =
```
"buffer number";
```
  buffer_index:comment =
```
Buffer information can be found in the Instrument Aux section;

boolean `overload`(*images*)

```
  overload:long_name =
```
"Probe particle overload status";
```
  overload:comment =
```
Any relevant comments;

In some cases, image buffers may contain pieces of information about each image in addition to the time, length, and image, as covered above. For example, DMT monoscale instruments include an image counter and depth of field (DOF) flag for each image; likewise, DMT greyscale instruments include an image counter and TAS for each image. In these cases where the additional information encoded in the data stream apply to single images, these parameters may be included in the Instrument Core section as additional parameters.

3.4 Instrument Aux

The Instrument Aux group contains auxiliary data relevant to a given instrument. This data is included to maintain integrity of the original dataset, making SPIF a suitable format for long term archiving. This group has its own time coordinate; this accommodates 1 Hz one dimensional data that may be transmitted in parallel to the two dimensional image data.

In this group, there are three parameters with defined names covering the aux timestamp and true airspeed (TAS). It is important to include TAS as a defined name, since this parameter is required to calculate sample volume for use in L2 derived microphysical parameters such as number concentration, liquid water content, etc.

int `time`(*time*)

```
time:standard_name =
                    “time”;
time:long_name =    “aux time in seconds”;
time:timezone =     “UTC”;
time:units =        “seconds since start_date 00:00:00 +0000”;
time:strftime_format =
                    “%F %T %Z”;
time:comment =      Any relevant comments;
```

int `TAS_original`(*time*)

```
TAS_original:long_name =
                    “TAS used during original data acquisition”;
TAS_original:units =
                    “m/s”;
TAS_original:comment =
                    Any relevant comments;
```

int `TAS_corrected`(*time*)

```
TAS_corrected:long_name =
                    “Corrected TAS”;
TAS_corrected:units =
                    “m/s”;
TAS_corrected:comment =
                    Any relevant comments;
```

Note there are two TAS parameters defined. *TAS_original* represents the TAS used during original acquisition of the data. If this TAS is accurate, and no corrections are required, this is the only TAS that needs to be supplied, and will be used in all subsequent calculations. However, if post-flight corrections were required to the TAS data, these should be input to the *TAS_corrected* parameter. If present, *TAS_corrected* will be used in all subsequent calculations. The ratio of corrected to original TAS values will also be used to recalculate particle diameters, to account for any stretching or contraction of images that occurred due to incorrect probe sample rates.

If desired, other parameters can be added to the Aux group, and may include:

- Housekeeping data
- Buffer time stamps
- Image counters
- Data acquisition timing words

- Temperature
- Altitude
- Standard PADS aggregated data

TODO: Add description of 'data discovery' terms.

3.5 Level 0 Processed Data

Following extraction of image data into SPIF format, images can be analyzed to extract information about the particles they contain. At the most basic level, parameters of interest describe geometric and physical measurements of the identified particles. Thus, the Level 0 data contains basic information about identified particles such as:

- Diameters (more discussion on this below)
- Area
- Perimeter
- Bounding box within image
- Orientation
- Right edge pixel count
- Left edge pixel count
- Center-in (boolean)
- All-in (boolean)

Interpretation of particle diameter presents a challenge, as there are currently several definitions of particle diameter in use by the community, and a standard definition likely isn't reasonable, since different diameters are useful depending on the measurement scenario. Thus, to make SPIF useful to the broader community, it should include a wide set of diameters in use by the community.

An additional consideration for the inclusion of various particle diameters is how these diameters are named. Throughout the literature, varying names have been given to essentially identical diameters. In the diameter definitions here, an attempt will be made to standardize the names, while referencing other names used for a given diameter definition.

Note that the Level 0 particles will be sized using number of pixels – conversion to sizing in microns takes place in Level 1. To differentiate measurements made in the different coordinate systems, sizing measurements in reference to number of pixels will have the prefix N, whereas measurements with units of microns will have the prefix D.

N_p	Max diameter in the photodiode-array dimension. Equivalent to N_y (Korelev et al. 2000, Leroy 2016) and L_5 (Lawson 2011).
N_t	Max diameter in the time dimension. Equivalent to N_x (Korolev et al. 2000, Leroy 2016) and L_1 (Lawson 2011).
N_{eq}	Diameter of circle with area equivalent to particle area.
N_s	Diameter of minimum enclosing circle. Equivalent to N_{max} (Heymsfield et al. 2013).
N_h	Hypotenuse of triangle formed by N_p and N_t.
N_m	Mean of N_p and N_t.
N_{slice_count}	Diameter in slice with maximum number of shaded pixels. Equivalent to L_2 (Lawson 2011).
N_{slice_diff}	Diameter in slice with greatest pixel separation. Equivalent to L_4 (Lawson 2011).
$N_{reconst}$	Reconstructed circle diameter for center-in particles.
N_{hole}	Max hole diameter as defined in Korolev 2007.

Each of the parameters discussed above applies to individual particles. For most OAPs, there can be multiple particles in a single image. Given this n-to-one relationship, the Level 0 particle data will require use of a new dimension corresponding to the number of particles detected, which is likely to be different than the number of images captured. The *particles* dimension thus covers all parameters described in this section. With the additional dimension, there is a need for supplemental parameters which describe the relationship of detected particles to their original image, both in terms of a reference to the additional image, as well as a more exact temporal location, based on the particle's location in the image frame.

image_index	Reference to source image index.
particle_sec	Particle arrival time in seconds.

particle_ns Particle arrival time in nanoseconds.

With these factors in mind, the Level 0 data should be implemented as described below:

int `image_index`(*particles*)

`image_index:long_name` =	"reference to index of image containing current particle";
`image_index:units` =	"image number";
`image_index:comment` =	Any relevant comments;

float `particle_ns`(*particles*)

`particle_ns:long_name` =	"particle arrival time in nanoseconds";
`particle_ns:units` =	ns since `particle_sec`;
`particle_ns:ancillary_variables` =	`particle_sec`;
`particle_ns:comment` =	Any relevant comments;

int `particle_sec`(*particles*)

`particle_sec:standard_name` =	"time";
`particle_sec:long_name` =	"image arrival time in seconds";
`particle_sec:timezone` =	"UTC";
`particle_sec:units` =	"seconds since `start_date` 00:00:00 +0000";
`particle_sec:strftime_format` =	"%F %T %z";
`particle_sec:comment` =	Any relevant comments;

float `N_p`(*particles*)

`N_p:long_name` =	"Max diameter in the photodiode-array dimension";
`N_p:equivalent_names` =	"N_y, L5"
`N_p:units` =	pixels;
`N_p:comment` =	Any relevant comments;

float `N_t`(*particles*)

```
N_t:long_name =          "Max diameter in the time dimension";
N_t:equivalent_names =
                         "N_x, L1"
N_t:units =              pixels;
N_t:comment =            Any relevant comments;
```

float `N_eq`(*particles*)

```
N_eq:long_name =         "Diameter of circle with area equivalent to particle area";
N_eq:equivalent_names =

N_eq:units =             pixels;
N_eq:comment =           Any relevant comments;
```

float `N_s`(*particles*)

```
N_s:long_name =          "Diameter of minimum enclosing circle";
N_s:equivalent_names =
                         "N_max"
N_s:units =              pixels;
N_s:comment =            Any relevant comments;
```

float `N_h`(*particles*)

```
N_h:long_name =          "Hypotenuse of triangle formed by N_p and N_t";
N_h:equivalent_names =

N_h:units =              pixels;
N_h:comment =            Any relevant comments;
```

float `N_m`(*particles*)

```
N_m:long_name =          "Mean of N_p and N_t";
N_m:equivalent_names =

N_m:units =              pixels;
N_m:comment =            Any relevant comments;
```

float `N_slice_count`(*particles*)

`N_slice_count:long_name` =
"Diameter of slice with maximum number of shaded pixels";
`N_slice_count:equivalent_names` =
"L2"
`N_slice_count:units` =
pixels;
`N_slice_count:comment` =
Any relevant comments;

float `N_slice_diff`(*particles*)

`N_slice_diff:long_name` =
"Diameter of slice with greatest pixel separation";
`N_slice_diff:equivalent_names` =
"L4"
`N_slice_diff:units` =
pixels;
`N_slice_diff:comment` =
Any relevant comments;

float `N_reconst`(*particles*)

`N_reconst:long_name` =
"Reconstructed circle diameter for center-in particles";
`N_reconst:equivalent_names` =

`N_reconst:units` = pixels;
`N_reconst:comment` =
Any relevant comments;

float `N_hole`(*particles*)

`N_hole:long_name` =
"Max hole diameter";
`N_hole:equivalent_names` =

`N_hole:units` = pixels;

`N_hole:comment` = Any relevant comments;

float `area`(*particles*)

`area:long_name` = "Number of shaded pixels";

`area:units` = pixels;

`area:comment` = Any relevant comments;

float `perimeter`(*particles*)

`perimeter:long_name` =
“Number of pixels around perimeter of particle”;

`perimeter:units` = pixels;

`perimeter:comment` =
Any relevant comments;

series of int `bbox`(*particles*, *coords*)

`bbox:long_name` = “Bounding box of particle”;

`bbox:units` = p0, t0, pN, tN pixel coordinates;

`bbox:comment` = Any relevant comments;

float `l_edge_count`(*particles*)

`l_edge_count:long_name` =
“Number of shaded pixels along left (0th) photodiode boundary for current particle”;

`l_edge_count:units` =
pixels;

`l_edge_count:comment` =
Any relevant comments;

float `r_edge_count`(*particles*)

`r_edge_count:long_name` =
“Number of shaded pixels along right (Nth) photodiode boundary for current particle”;

`r_edge_count:units` =
pixels;

`r_edge_count:comment` =
Any relevant comments;

boolean `center_in`(*particles*)

`center_in:long_name` =
“Flag indicating if particle is center-in”;

`center_in:units` = pixels;

`center_in:comment` =
Any relevant comments;

boolean `all_in`(*particles*)

`all_in:long_name` =
"Flag indicating if particle is all-in";

`all_in:units` = pixels;

`all_in:comment` = Any relevant comments;

float `ellipse_maj_axis`(*particles*)

`ellipse_maj_axis:long_name` =
"The length of the major axis of the ellipse fit to particle";

`ellipse_maj_axis:units` =
pixels;

`ellipse_maj_axis:comment` =
Any relevant comments;

float `ellipse_min_axis`(*particles*)

`ellipse_min_axis:long_name` =
"The length of the minor axis of the ellipse fit to particle";

`ellipse_min_axis:units` =
pixels;

`ellipse_min_axis:comment` =
Any relevant comments;

float `ellipse_angle`(*particles*)

`ellipse_angle:long_name` =
"Orientation of particle fit to ellipse";

`ellipse_angle:units` =
radians;

`ellipse_angle:comment` =
"Given in radians with range from -pi/2 to pi/2. Reference is parallel to photodiode array.";

float `fit_maj_axis`(*particles*)

`fit_maj_axis:long_name` =
"The length of particle along the least-squares fit line orientation";

`fit_maj_axis:units` =
pixels;

`fit_maj_axis:comment` =
Any relevant comments;

float `fit_min_axis`(*particles*)

`fit_min_axis:long_name` =	"The length of particle perpendicular to the least-squares fit line orientation";
`fit_min_axis:units` =	pixels;
`fit_min_axis:comment` =	Any relevant comments;

float `fit_angle`(*particles*)

`fit_angle:long_name` =	"Orientation of particle fit to line using least-squares technique";
`fit_angle:units` =	radians;
`fit_angle:comment` =	"Given in radians with range from -pi/2 to pi/2. Reference is parallel to photodiode array.";

float `fit_r2`(*particles*)

`fit_r2:long_name` =	"R squared value from least-squares line fit";
`fit_r2:units` =	unitless;
`fit_r2:comment` =	Any relevant comments;

float `F`(*particles*)

`F:long_name` =	"Shape factor of particle.";
`F:units` =	unitless;
`F:comment` =	"As defined in Holroyd 1986.";

float `S`(*particles*)

`S:long_name` =	"Ratio of fully shaded slices to total slices";
`S:units` =	unitless;
`S:comment` =	"As defined in Holroyd 1986.";

3.6 Level 1 Processed Data

Whereas Level 0 data presents particle information as simply properties of an image, Level 1 contains particle properties linked to physical, real-world quantities. In Level 1, there are two primary categories of data:

1. Particle properties scaled to physical dimensions (μm, etc.) using the reso-

lution of the instrument

2. Parameters classifying particles into habits or other categories

When generating scaled particle properties, care must be taken to correct for improper scaling in the image time direction due to inconsistencies between the probe sampling rate and the speed of the aircraft. These inconsistencies can happen for various reasons – the most common reasons include exceeding TAS limits of the probe; having incorrect or constant airspeed inputs supplied to the probe; or problems with local pitot measurements due to icing, blockages, or other reasons. In these cases, the image scaling along the time direction can be corrected by scaling it by the correct TAS divided by the original TAS.

Scaled particle properties should include the following parameters:

int `image_index`(*particles*)

`image_index:long_name` =	"reference to index of image containing current particle";
`image_index:units` =	"image number";
`image_index:comment` =	Any relevant comments;

float `D_p`(*particles*)

`D_p:long_name` =	"Max diameter in the photodiode-array dimension";
`D_p:equivalent_names` =	"D_y, L5"
`D_p:units` =	microns;
`D_p:comment` =	Any relevant comments;

float `D_t`(*particles*)

`D_t:long_name` =	"Max diameter in the time dimension";
`D_t:equivalent_names` =	"D_x, L1"
`D_t:units` =	microns;
`D_t:comment` =	Any relevant comments;

float `D_eq`(*particles*)

`D_eq:long_name` =	"Diameter of circle with area equivalent to particle area";
`D_eq:equivalent_names` =	
`D_eq:units` =	microns;
`D_eq:comment` =	Any relevant comments;

float `D_s`(*particles*)

`D_s:long_name` = “Diameter of minimum enclosing circle”;
`D_s:equivalent_names` =
“D_max”
`D_s:units` = microns;
`D_s:comment` = Any relevant comments;

float `D_h`(*particles*)

`D_h:long_name` = “Hypotenuse of triangle formed by D_p and D_t”;
`D_h:equivalent_names` =

`D_h:units` = microns;
`D_h:comment` = Any relevant comments;

float `D_m`(*particles*)

`D_m:long_name` = “Mean of D_p and D_t”;
`D_m:equivalent_names` =

`D_m:units` = microns;
`D_m:comment` = Any relevant comments;

float `D_slice_count`(*particles*)

`D_slice_count:long_name` =
“Diameter of slice with maximum number of shaded pixels”;
`D_slice_count:equivalent_names` =
“L2”
`D_slice_count:units` =
microns;
`D_slice_count:comment` =
Any relevant comments;

float `D_slice_diff`(*particles*)

`D_slice_diff:long_name` =
“Diameter of slice with greatest pixel separation”;
`D_slice_diff:equivalent_names` =
“L4”
`D_slice_diff:units` =
microns;
`D_slice_diff:comment` =
Any relevant comments;

float `D_reconst`(*particles*)

`D_reconst:long_name` =
"Reconstructed circle diameter for center-in particles";
`D_reconst:equivalent_names` =

`D_reconst:units` = microns;
`D_reconst:comment` =
Any relevant comments;

float `D_korolev`(*particles*)

`D_korolev:long_name` =
"Korolev-corrected diameter";
`D_korolev:equivalent_names` =

`D_korolev:refererence` =
"Korolev, 2007"

`D_korolev:units` = microns;
`D_korolev:comment` =
Any relevant comments;

float `N_korolev`(*particles*)

`N_korolev:long_name` =
"Korolev-corrected diameter";
`N_korolev:equivalent_names` =

`N_korolev:refererence` =
"Korolev, 2007"

`N_korolev:units` = pixels;
`N_korolev:comment` =
Any relevant comments;

float `ellipse_maj_axis`(*particles*)

`ellipse_maj_axis:long_name` =
"The length of the major axis of the ellipse fit to particle";
`ellipse_maj_axis:units` =
microns;
`ellipse_maj_axis:comment` =
Any relevant comments;

float `ellipse_min_axis`(*particles*)

`ellipse_min_axis:long_name` =	"The length of the minor axis of the ellipse fit to particle";
`ellipse_min_axis:units` =	microns;
`ellipse_min_axis:comment` =	Any relevant comments;

float `fit_maj_axis`(*particles*)

`fit_maj_axis:long_name` =	"The length of particle along the least-squares fit line orientation";
`fit_maj_axis:units` =	microns;
`fit_maj_axis:comment` =	Any relevant comments;

float `fit_min_axis`(*particles*)

`fit_min_axis:long_name` =	"The length of particle perpendicular to the least-squares fit line orientation";
`fit_min_axis:units` =	microns;
`fit_min_axis:comment` =	Any relevant comments;

float `area`(*particles*)

`area:long_name` =	"Area of shaded pixels";
`area:units` =	microns^2;
`area:comment` =	Any relevant comments;

The second component of Level 1 data contains parameters used to classify particles. Examples of such classification parameters could include results from the Holroyd habit classification technique, or the Korolev habit classification technique. There are no set requirements regarding which classification methods are included, but any included parameters should follow the structure outlined below.

float `habit`(*particles*)

`habit:long_name` =	"Name of habit classification technique";
`habit:key` =	parameter1: char1, parameter2: char2, etc;
`habit:comment` =	Any relevant comments;

Where the *key* attribute uses keyword: value pairs to specify the classification types and their corresponding character code for the given parameter.

In cases where classifications have auxiliary parameters necessary for classification (e.g., the *R* parameter in the Korolev classification scheme), these should also be included in this section. These parameters should be named in such a way where they can be easily identified, and should include a long description and units parameter, as appropriate.

4 Level 2 Processed Data

Level 2 Processed Data contains binned histograms and calculated bulk microphysical parameters calculated using the Level 1 per-particle data using a fixed time base. These parameters are either one-dimensional (bulk parameters) or two-dimensional (binned histograms), where a reference to time is the principle dimension in both cases. Thus, all parameters in Level 2 use *Time* as their primary dimension, calculated using a fixed time base, and encompassing all particle detection timestamps in the file. In the case of binned data, the second dimension is *Bins*.

5 SPIF Code Definitions

Will complete this section when code is more final.

5.1 Data Chunking

It is quite possible for a translator to read in an entire raw data file for a single probe, parse that into the appropriate (and possibly large) python variables, and pass the resultant dictionary to the NetCDF read/writer module for writing into the SPIF file. This would result in on-the-fly chunking and filtering by the NetCDF library. It is also possible that the raw data files are too numerous or large to be handled all together and so subsets of the data need to be streamed to the read/write module piecemeal. This will also occur if the SPIF file is being written in realtime. This will mean that chunking and filtering may have to be set manually. Thus there may be information passed from the translator to the read/write module that is not stored in the SPIF file but is used to construct the SPIF file.