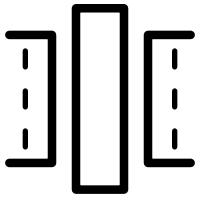
Janus

GUI Software for Processing Thermal-Ion Measurements
from the *Wind* Spacecraft's Faraday Cups

User Guide

Software Version: 0.3.0

Date: November 21, 2016



Janus – GUI Software for Processing Thermal-Ion Measurements from the *Wind* Spacecraft's Faraday Cups

The development of Janus was suppported by NASA Award NNX15AF52G.

Contents

Chapter 1

Installation

1.1 Installing Python and Other Dependencies

The Janus software has hereto been developed on a 64-bit workstation running Ubuntu Linux version 12.04. Though one goal of this project is to make Janus cross-platform, no tests of the code have yet been made on any other systems.

The code for Janus is written exclusively in Python and makes use of several extension modules. Thus, before a user can run Janus, they must install Python and the required modules.

During the development of Janus, version 2.7 of Python was used, but, in principal, version 2.6 should be sufficient. Use of version 3 has not been tested and is not recommended for this version of Janus.

Janus uses Python extension modules to extract data from CDF files, perform statistical analyses and fits, create and manage a GUI in the Qt application framework, and plot data in that GUI. Installation of the following two modules (along with their dependencies) will satisfy all the module dependencies of Janus:

- pyqtplot. This module provides an effective and efficient system for plotting data. Installation instructions are available on the project's website. Alternatively, a user running Ubuntu Linux can simply install the "python-pyqtgraph" package by using Synaptic Package Manager or by executing the following in a terminal:

```
sudo apt-get install python-pyqtgraph
```

- SpacePy. This module was developed at Los Alamos National Laboratory and contains many tools developed specifically for space scientists. Even so, Janus only uses this module to read CDF data files.[1] Installation instructions are available on the project's website.

[1] Future versions of Janus may use a custom CDF reader to avoid superfluous dependencies.

1.2 Extracting and Running Janus

Once its dependencies are satisfied, the Janus code can be extracted and run. The software is commonly distributed as a gzip'ed tar archive with the file name `janus_0.3.0 .tar.gz`. It is also available from the Janus Wind/FC repository on GitHub. The archive and repository each contains a PDF of this user guide, `janus.pdf`, and a Python script, `janus`, that launches the Janus application. On a POSIX system, this script can be executed by opening a terminal emulator, navigating to the software directory, and running the following:

```
./janus
```

Chapter 2

Usage

2.1 Application Window

The Janus application window is divided into into five primary widgets:

- The *Control Widget* is located in the upper left and is used to control the loading and analysis of ion spectra. It also contains a text box which provides the user with status updates.
- The *Faraday Cups Widget* comprises the entire left half of the window and displays the ion spectrum that is currently under analysis.
- The *MFI Widget* is in the center right and displays the magnetic field measurements concurrent with the displayed ion spectrum.
- The *Moments Analysis Widget* is also in the center right and is used to adjust the settings of and to report the results from the moments analysis.
- The *Non-Linear Analysis Widget* is located in the the lower right and is used to adjust the settings of and to report the results from the non-linear analysis.

2.2 Loading Data

2.2.1 Ion Spectra

To request that an ion spectrum be loaded and analyzed, its timestamp is entered into the text box at the top of the left column of the Control Widget. The standard format is as follows:

```
yyyy-mm-dd/hh:mm:ss
```

When entering a timestamp, the user need not adhere strictly to this format. Leading zeros may be omitted (e.g., `03` and `3` are both valid) and substitutions may be made for the non-numeric characters. Likewise, the user may provide the time portion of the timestamp with less precision or omit it entirely.

After the timestamp has been entered, the user may either click on the "Go To Spectrum" button below the text box or simply press the "Enter" key on the keyboard. Once this happens, Janus attempts to retrieve the ion spectrum with the timestamp closest to the one requested. The software uses the date from the requested timestamp to determine which CDF file should be loaded. If this file is not available in the data directory, the software attempts to download it from the CDAWeb FTP server. For more details on the downloading and loading ion spectra (including advanced configuration options), see Section 4.2.

Once an ion spectrum has been loaded, any text in the timestamp text box is replaced with the timestamp of that ion spectrum. A different spectrum can be loaded by again using this text box. Alternatively, jump buttons can be used to quickly browse through a series of spectra. The "<" and ">" buttons respectively allow the user to move from the current spectrum the one that was measured immediately before and after it. Likewise, the "<<" and ">>" buttons respectively load the spectrum measured an hour earlier and later.

The data from a loaded ion spectrum are plotted in the Faraday Cups Widget as black histograms. This widget contains two tabs: one for each Faraday cup. In turn, each tab contains twenty plots: one for each look direction of the corresponding cup. Each plot is labeled with the altitude and azimuth of its corresponding look direction. The vertical axis of each plot is the measured current (in pA) and the horizontal axis is the proton inflow velocity (in km/s) projected along the look direction of the cup.

2.2.2 Magnetic Field Data

Whenever a new ion spectrum is loaded, Janus automatically attempts to load any corresponding magnetic field measurements. The software handles the data files for the magnetic field measurements just like it handles those for ion spectra. Any data file that cannot be located on disk is downloaded from CDAWeb. Section 4.2 presents more information on the downloading and loading of magnetic field data (including advanced configuration options).

Once loaded, the magnetic field data are displayed in the MFI Widget. The first tab of this widget plots the magnetic field (in nT) versus time (in seconds from the timestamp of the ion spectrum). The black curves show the magnitude and negative thereof; the red, green, and blue curves respectively show the x, y, and z vector components in the GSE coordinate system. This widget's second tab provides average values for the loaded magnetic field data.

2.3 Dynamic Analysis Settings

Janus as the ability to make some or all aspects of its data analysis "dynamic": i.e., to automatically rerun some or all of the analysis in response to the user's loading new data or changing settings. The degree to which different elements of the data analysis are dynamic

is controlled by the four check boxes that appear in the Control Widget under the heading "Dynamic."

2.3.1 Moments Analysis

The moments analysis is a relatively elementary method for generating values for the proton bulk parameters. If the "Moments" box is checked, the moments analysis is automatically run whenever a new ion spectrum is loaded. The moments box automatically becomes checked (and the moments analysis is run) if any change is made to the data selection – whether the change is made manually by the user or automatically by the data selection algorithm (see Sections 2.4 and 2.5, respectively).

2.3.2 Initial Guess Generation

Janus can use the results of the moments analysis to generate initial guesses for the bulk parameters of each ion species considered in the non-linear analysis. If the "Init. Guess" box is checked, this process automatically occurs after any run of the moments analysis.

The settings that control the initial guess generation can be adjusted from the "Settings" tab in the Non-Linear Analysis Widget (see Section 2.6.3). If one of these is modified, the "Init. Guess" box automatically becomes checked (and the initial guess generation is run).

As opposed to being derived from the results of the moments analysis, initial guess values can also be specified manual by the user via the "Init. Guess" tab in the Non-Linear Analysis Widget (see Section 2.6.4). If an initial guess value is entered in this manner, the "Init. Guess" box automatically becomes unchecked.

2.3.3 Data Selection

Janus can use the initial guess values to select data for the non-linear fitting. If the "Data Sel." box is checked, this then happens whenever the initial guess is changed – whether the change is made automatically by the initial guess generation algorithm or manually by the user (see Sections 2.6.3 and 2.6.4, respectively).

The settings that control the automatic data selection can be adjusted from the "Settings" tab in the "Non-Linear Analysis" widget (see Section 2.6.3). If one of these is modified, the "Data Sel." box automatically becomes checked (and the data selection algorithm is run).

A user can also use the Faraday Cups Widget to manually modify the data selection for the non-linear fitting (see Section 2.4). If such a change is made, the "Data Sel." box automatically becomes unchecked.

2.3.4 Non-Linear Fitting

The non-linear analysis of an ion spectrum involves using the initial guess to seed a non-linear fit of the selected data points. If the "Non-Linear" box is check, this analysis will be run if any of the following occur:

- The initial guess for the non-linear analysis is changed either automatically by the initial guess generation algorithm or manually by the user (see Sections 2.6.3 and 2.6.4, respectively).

- The data selection for the non-linear analysis has been changed either automatically by the data selection algorithm or manually by the user (see Sections 2.6.3 and 2.4, respectively).

Especially when multiple ion populations are being considered, non-linear fitting can be time-consuming. Thus, a user may choose to leave the "Non-Linear" box unchecked while adjusting the initial guess and data selection for the non-linear analysis.

2.4 Display Settings and Manual Data Selection

In addition to showing the measured data from the ion spectrum, the plots in the Faraday Cups Widget can also display data selections and model response curves. This behavior is controlled by the three check boxes that appear in the Control Widget under the heading "Display." At most, only one of these check boxes can checked at any given time. If none of the boxes is checked, the Faraday Cups Widget displays only the measured data.

If the "Moments" box is checked, the Faraday Cups Widget overlays cyan diamonds indicating the data selection for the moments analysis and blue curves plotting the proton signal inferred therefrom. The displayed data selection is interactive: the user can add or remove any spectral point simply by clicking it. The moments analysis will rerun each time the user changes the data selection in this manner.

If the "Guess/Sel." box is checked, the Faraday Cups Widget shows the selected data points and initial guess curves for the non-linear analysis. Green curves the modeled response from each ion species based on the initial guess values for its bulk parameters, and blue curves show the total modeled response (i.e., the sum of the individual ones). As with the moments analysis, the data selected for the non-linear analysis are marked with cyan diamonds, and the selection can be adjusted with mouse clicks. When the data selection is adjusted in this way, dynamic point selection is automatically disabled (see Section 2.3.3). Regardless, if dynamic non-linear fitting is enabled (see Section 2.3.4), non-linear fitting is automatically run in response to each change in the data selection.

If the "Non-Linear" box is checked, the Faraday Cups Widgets shows the results of the non-linear fitting using the same system of green and blue curves described for the "Guess/Sel." box. Additionally, the data selection for the non-linear analysis is also shown and can be adjusted in the same manner (and with the same results) as described above.

2.5 Moments Analysis

Moments analysis provides a relatively elementary but robust method for inferring values for the proton bulk parameters. It is run automatically if it is set to be dynamic (see Section

2.3.1). Alternatively, the user can manually run the moments analysis by clicking the "Run Moments Analysis" button in the Control Widget. In either case, the Moments Analysis Widget provides a means to control the settings for and to view the numerical results of this analysis.

The first tab of the Moments Analysis Widget specifies the settings for the automatic selection of data for the moments analysis. The algorithm selects a certain number (no less than five) of look directions from each Faraday cup, and then selects a certain number (no less than three) of data points from each of those look directions. Both quantities can be adjusted from this tab. Any such adjustment will automatically set the moments analysis to be dynamic (see Section 2.3.1).

The moments Analysis Widget's second tab displays the values derived for the proton bulk parameters from the moments analysis. For thermal speeds (and the components thereof), the following definition is employed (both in this tab and throughout Janus):

$$w_p \equiv \sqrt{\frac{k_B T_p}{m_p}} \ ,$$

where w_p is the proton thermal speed, k_B is the Boltzmann constant, T_p is the proton temperature, and m_p is the proton mass.

2.6 Non-Linear Analysis

In contrast to the moments analysis, the non-linear analysis is much more versatile and accurate. It is run automatically if it is set to be dynamic (see Section 2.3.4), but a user can also run it manually by clicking the "Run Non-Linear Analysis" button in the Control Widget.

By using a sophisticated non-linear fitting algorithm, Janus is able to consider multiple ion species and populations simultaneously. Additionally, various aspects of each model VDF are adjustable via the Non-Linear Analysis Widget, which also provides the numerical results of the fit.

Janus' non-linear analysis makes a clear distinction between a "species" and a "population." A species refers to all particles of a given type: e.g., the protons or α-particles. A population is a component of a species' total VDF: e.g., the proton core or α-particle beam. Each population should be associated with a species, though each species may have multiple populations. Chapter 3 shows that it allows the analysis to results to be explored much more easily.

2.6.1 Species Properties

The first tab of the Non-Linear Analysis Widget allows the user to see and adjust the ion species in the non-linear analysis. Space is provided for up to four species: one row for each. The first two species are defaulted to protons and α-particles, but the user may elect to change these.

A user specifies a species by providing its name, a symbol for the species (e.g., "p" for protons), and the species mass and charge (relative to the proton). Janus requires that all species have different names and different symbols (though a species name may be identical to its symbol). Additionally, certain strings of characters are prohibited from being either names or symbols (see Section 3.3). The symbol field is primarily provided as a convenience to the user (e.g., to provide more-concise expressions of results; see Section 2.6.5).

2.6.2 Population Properties

The "Populations" tab of the Non-Linear Analysis Widget allows the user to see and adjust the ion populations. Space is provided for up to five populations: one row for each. The default populations are the proton core, proton beam, α-particle core, and α-particle beam, but the user is at liberty to change these.

The check box on the far left of each row indicates whether a given population should be used in the non-linear analysis. No such check box appears in the first row since that population (at a minimum) must be included in the analysis.

Each population must be associated with a species (which is selected from a pull-down menu) and must be given a name and symbol. A population's name and symbol must each be distinct from the names and symbols of all species and of all populations *of its own species*. Populations of different species may have the same name and/or symbol (e.g., the protons and α-particles may both have a population named "core"). Nevertheless, certain strings of characters are prohibited from being either names or symbols (see Section 3.3).

Each population has two additional options. If the "Drift" box is checked, the population is allowed to drift relative to the bulk velocity. The direction of this drift velocity is constrained to be parallel to the magnetic field. Janus does not allow the first population to have drift since it is assumed to be the population that the other populations are drifting relative to. If the "Aniso" box is checked, Janus considers the corresponding population to have temperature anisotropy: i.e., separate temperature components perpendicular and parallel to the background magnetic field. Otherwise, Janus treats the population as having a single, isotropic temperature.

2.6.3 Settings for Initial Guess Generation and Data Selection

The "Settings" tab of the Non-Linear Analysis Widget allows the user to adjust both the settings for the automatic generation of initial-guess values and those for the automatic data selection. Separate settings are provided for each ion population: one row of text boxes for each. When a given population is not in use, it's text boxes are shaded gray.

The initial guess settings appear in the first three columns of text boxes and specify how the results of the moments analysis are used to guess values for ion bulk parameters. The "n/n_m" and "w/w_m" boxes respectively set the ratio of the guessed density and thermal speed relative to those returned by the moments analysis. Even for ion populations which are anisotropic, the initial guess generator always produces isotropic thermal speeds. The "dv/v_m" box sets the ratio of the guessed differential flow (relative to the bulk velocity from

the moments analysis) to the bulk velocity from the moments analysis. The sign convention for the "dv/v_m" values is somewhat unusual: a positive value corresponds to a bulk speed that is faster than that from produced by the moments analysis.[1]

The settings for automatic data selection appear in the last two columns of text boxes and define the domain of phase space associated with each population. The speed values in these text boxes are relative to the initial guess for the bulk velocity (i.e., a value of zero corresponds to the guessed bulk velocity) and are scaled by the initial guess for the thermal speed. The data selection algorithm considers the same look directions used in the moments analysis and selects all data that fall into the projected phase-space domain of at least one population.

2.6.4 Manual Initial Guess Entry

The "Initial Guess" tab of the Non-Linear Analysis Widget specifies the initial guesses for the ion parameter values. If initial guess generation is set to be dynamic (see Section 2.3.2), the text boxes in this tab are populated and updated automatically. Alternatively, values can be directly entered by the user.

The first text box in each row specifies the initial guess for the corresponding population's density (in cm^{-3}).

The next box (if present) specifies the guess for the population's differential flow (in km/s). If the population is not allowed to drift (see Section 2.6.2), this box is hidden.

Finally, either one or two text-boxes are provided for the thermal speed (in km/s). Isotropic populations (see Section 2.6.2) has only single box. Anisotropic populations have two: the first for the perpendicular thermal-speed and the second for the parallel.

The final row of this tab allows the user to manually specify the initial guess of the bulk velocity (in km/s). There text boxes are provided for the x-, y-, and z-components in the GSE coordinate system.

2.6.5 Results

The "Results" tab of the Non-Linear Analysis Widget displays the parameter values returned by the non-linear fitting.

2.7 Automatic Analysis

This feature is currently under development and is therefore disabled in this release.

[1] Elsewhere in the Non-Linear Analysis Widget, the sign of differential flow is taken to indicate parallel versus anti-parallel flow relative to the background magnetic field.

2.8 Outputting Analysis Results

Whenever a non-linear fit is performed on a spectrum, Janus automatically logs the results. Only the results from the most recent analysis of each spectrum are retrained. Janus provides the user with two options for outputting these result: exporting and saving. Each output operation is controlled by a button in the lower-right of the Control Widget.

2.8.1 Exporting

When results are exported, they are outputted to a text file. The formatting of this text file is documented in a header therein. Use of exporting can be somewhat inconvenient because the files tend to become large and there is no official software for reading the data back in for further analysis (e.g., plotting the evolution of density and other parameters with time).

2.8.2 Saving

Janus logs the results of the non-linear fits in a Pyon `series` object (see Section 3.3 for the full description of the Pyon module's `series` class). When the save option is executed, this object is exported into a file (typically with the extension "jns") using Python's built-in Pickle module.

The following example code shows how to recover the Pyon `series` stored in a hypothetical Janus save file named `save.jns`. For convenience, it is assumed that the working directory contains this file, `janus_pyon.py` (which contains the Pyon module's source code), and `janus_const.py` (on which Pyon depends).

```
import pickle
dat = pickle.load( open( 'save.jns', 'rb' ) )
```

After this code is successfully run, the variable `dat` will point to the Pyon *series* restored from the file `save.jns`.

Chapter 3

The Pyon Module

The Pyon (i.e., "Python ion") module was specifically developed for storing results from Janus' non-linear analysis (see Section 2.6) and was structured to accommodate the extreme flexibility that Janus offers users. This Chapter does not document all of Pyon's features; rather, it provides a basic introduction to Pyon and highlights its key features. Readers are encouraged to consult the documentation in the Pyon source code (in the file `janus_pyon.py`) for further details.

3.1 The `spec` and `pop` Classes

Pyon, like Janus' non-linear analysis (see Section 2.6), makes a clear distinction between a "species" and a "population." A species refers to all particles of a given type: e.g., the protons or α-particles. A population is a component of a species' total VDF: e.g., the proton core or α-particle beam. Each population should be associated with a species, though each species may have multiple populations.

The Pyon classes `spec` and `pop` store/manage the properties of species and populations, respectively. One of the attributes of the `pop` class is intended to point to a `spec` object so that each population can be linked to a species. Each classes has two string attributes: one for a name and one for a symbol. As is discussed below, the symbol strings provide users with a convenient way to access stored values in instances of these classes.

It is expected that most users will not use the `spec` and `pop` classes directly. Rather, it is more efficient to use the `plas` class (see Section 3.2), which provides a means for organizing the parameters of all ion species and populations in a plasma.

3.2 The `plas` Class

A `plas` object is able to store the analysis results of a single ion-spectrum. Its attributes include the vector magnetic-field, the vector bulk-velocity, an array of ion species (i.e., instances of the `spec` class), and an array of ion populations (i.e., instances of the `pop` class).

By far, the most sophisticated feature of the `plas` class is its parser, which is invoked by treating a `plas` object like a Python `dict`. This provides users with the easiest way to access values stored in a `plas` object (including in its constituent `spec` and `pop` objects). Additionally, the parser is able to carry out fundamental calculations to return derived parameters (e.g., plasma beta).

As an example, consider a `plas` object `pls` that has two species, protons and α-particles (symbols: `p` and `a`), each of which has both a core and a beam population (symbols: `c` and `b`). The following are examples of calls of the parser with discussions of the results:

- `pls['vec_b0']` returns the three components of the magnetic field.
- `pls['vec_v0']` returns the three components of the bulk velocity.
- `pls['v0_x']` returns the x-component of the bulk velocity.
- `pls['s_v0_x']` returns the uncertainty in the x-component of the bulk velocity.
- `pls['v0']` or `pls['mag_v0']` returns the magnitude of the bulk velocity.
- `pls['n_p_c']` returns the density of the proton core.
- `pls['n_p']` returns the total proton density. The parser automatically searches for all proton populations, and, upon finding the two (i.e., the core and beam) retrieves the density of each and returns their sum.
- `pls['dv_a_c']` returns the drift velocity (parallel to the magnetic field) of the α-particle core. If the α-particle core was not configured to drift, then this returns `0.`, but no error is raised.
- `pls['vec_v_a_c']` returns the three components of the bulk velocity of the α-particle core. The parser automatically calculates this by recursively calling `pls['vec_v0']`, `pls['vec_b0']`, and `pls['dv_a_c']`.
- `pls['w_par_a_b']` returns the parallel thermal-speed of the α-particle beam. If the α-particle beam was not configured to be anisotropic, then this would return it's scalar thermal-speed (without raising an error).
- `pls['w_a_b']` returns the thermal-speed of the α-particle beam. If the α-particle beam was configured to be anisotropic, then the parser computes and returns the scalar thermal speed from the population's perpendicular and parallel thermal-speeds.
- `pls['w_par_a']` returns the total, parallel thermal-speed of the α-particles. The parser automatically searches for all the populations associated with the α-particle species. In its calculations, the parser takes into account not only the populations' different densities and parallel thermal-speeds but also the relative drift among them.
- `pls['t_a']` returns the total, scalar temperature of the α-particles.

- `pls['n_p_k']` returns `None` (since the proton species has not population with the symbol `k`). No error is raised.
- `pls['n_k']` returns `None` (since no species has the symbol `k`). No error is raised.
- `pls['n_x_p']` returns `None` (since density is a scalar). No error is raised.

These examples make clear that the parser automates many of the basic calculations that are frequently encountered in the analysis of ion bulk parameters. Additionally, the parser avoids (as much as possible) raising any errors; instead, it simply returns `None` when it cannot find/compute the requested parameter.

Note that the order of the elements of a key string passed to the parser is irrelevant. For example, `pls['n_p_c']`, `pls['p_n_c']`, and `pls['c_p_n']` all return the same result.

In order for the parser to function correctly, the `plas` class enforces certain restrictions on the names and symbols used for species and populations. The requirements for unique names and symbols are addressed in Sections 2.6.1 and 2.6.2. Additionally, strings such as `v0`, `n`, `beta`, and `time` are forbidden as either names or symbols so that the parser can operate without confusion. The complete list of these "reserved names" is available at the beginning of the Pyon source code (i.e., the file `janus_pyon.py`).

3.3 The `series` Class

Essentially, the Pyon `series` class is an array of `plas` objects. Janus uses a `series` object to log non-linear analysis results: the values returned by the most recent fit of each ion-spectrum are stored. When the "save" operation is executed, this `series` object is copied into a Python Pickle file (see Section 2.8).

The primary advantage of using the `series` class is that it makes use of the `plas` class's parser. For example, suppose that `dat` is a `series` object that contains five `plas` objects. Then, `dat['n_p_b']` returns a five-element array of proton-beam densities. In this context, the advantage of the parser's use of `None` (rather than raising an error) becomes apparent. If the user did not fit a proton beam for all five spectra, the returned array will contain some `None` values, but these can be easily identified and excised (prior to generating plots or making further calculations).

Chapter 4

Advanced Configuration

4.1 Source Code Files

The Python source code for Janus is contained in the `janus` directory. The file `janus.py` contains the "main" Janus class, `janus`. Each of the other Janus classes is contained in a file named `janus_*.py` where `*` is the name of the class; for example, the file `janus_core.py` has the code for the `core` class. A file whose name has one or more upper-case letters contains a customized extension of a PyQt class; for example, the file `janus_event_LineEdit.py` contains the class `event_LineEdit`, which extends the PyQt class `QLineEdit`. The various widgets used in the Janus application window are all instances of the classes specified in `janus_widget_*.py` files. Some source code files merely contain "helper functions" rather than classes (e.g., `janus_time.py`).

4.2 Data Archives

Janus maintains two archives of data files: one for the *Wind*/FC ion spectra and another for the *Wind*/MFI magnetic field data. By default, these collections of files are respectively stored in the directories `janus/data/fc/` and `janus/data/mfi/` (or their equivalents on non-POSIX operating systems).

From the perspective of the Janus code, the the `fc_spec` class creates one instance each of the `fc_arcv` and `mfi_arcv` classes. All data requests are made through these archive classes, which download and read in files as needed and (if requested) delete old files. Whenever an archive reads in a file, it retains those data along with those previously loaded (up to a user-defined limit) for potential later use (to minimize the number of file-reading operations).

Various aspects of each archive class can be adjusted through keywords that can be used at its initialization (see the `__init__` function of the `fc_spec` class). Each archive stores the values of these keywords for later use, but manually changing these values after initialization is strongly discouraged as it can produce errors. The initialization keywords for the archive classes are listed and described below:

- `spec` — Default value: `None`. Usage: pointer to the instance of `fc_spec` that owns the archive. By default, an archive has no access to the instance of `fc_spec` that created it and is thus unable to communicate with the user (see the `verbose` keyword). Such communication is made possible by having `fc_spec` initialize each archive with `spec=self`.

- `buf` — Default value: `3600`. Usage: buffer size in seconds. If the user requests data that is within `buf` seconds of the beginning or end of a given day, the archive loads the data for the previous or next day (respectively) as well as those for the requested day. Setting `buf` to any non-positive value disables this buffer feature.

- `tol` — Default value: `3600` (for `fc_arcv`) or `0` (for `mfi_arcv`). Usage: time tolerance in seconds. When a request is made to `fc_arcv` for an ion spectrum, one is only returned if its timestamp is withing `tol` seconds of the user-requested timestamp. When a request is made to `mfi_arcv` for a range of magnetic field data, that range is automatically padded on both sides by `tol` seconds.

- `use_idl` — Default value: `False`. Usage: indicator of whether IDL data files should be used. By default, each archive attempts to download and read in CDF data files from CDAWeb. If `use_idl` is set to `True`, the archive instead uses IDL data files. Because these IDL files are not available for download, they must already be in the data directory. For `fc_arcv`, the IDL files should be in a custom format (essentially a calibrated version of the `ionspec` format); for `mfi_arcv`, the 20-day `mag` files should be used.

- `use_k0` (only `mfi_arcv`) — Default value: `False`. Usage: indicator of whether `k0` magnetic field data should be used. CDAWeb contains several different types of data files for *Wind*/MFI. By default, `mfi_arcv` uses the `h0` files, but, if `use_k0` is set to `True`, the `k0` files are used instead. Because the `k0` data are sparsely spaced in time, a user who uses `use_k0=True` might also consider using a moderate, positive value for `tol` (e.g., `90`). The value of `use_k0` is disregarded when `use_idl=True` is used.

- `n_file_max` — Default value: `float('infinity')` (i.e., ∞). Usage: maximum number of data files permitted in the data directory. Whenever the archive loads a new data file, a count is made of the files in the data directory that are of the type specified by the `use_idl` and (if applicable) `use_k0` keywords. If this count is found to be larger than `n_file_max`, the excess files with the oldest "access dates" are deleted.

- `n_date_max` — Default value: `40`. Usage: maximum number of days of data permitted to be stored in the archive's memory. Whenever a data request is made to the archive, it makes a count of how many days of data it currently has loaded. If this count is found to be larger than `n_date_max`, the excess days of data that were least-recently loaded are removed from archive's memory.

- `path` — Default value (on a POSIX operating system): `janus/data/fc/` (for `fc_arcv`) or `janus/data/mfi/` (for `mfi_arcv`). Usage: path to the data directory for the archive.
- `verbose` — Default value: `True`. Usage: indicator of whether the archive should message the user. By default, the archive informs the user (via the Control Panel Widget) with status information on the downloading of files and the loading of data. If `verbose` is set to `False`, no such messages are sent. The value of `verbose` is effectively `False` (regardless of its actual value) if a valid pointer is not provided via the `spec` keyword.