# Documentation for `WCSFoF` Code

G. M. Bernstein

*Dept. of Physics & Astronomy, University of Pennsylvania*

garyb@physics.upenn.edu

10 June 2013

## 1. Concepts

- *Point or Detection:* An observation of an object with a position.
- *Match:* A group of detections that are connected by a chain of proximity on the sky (the classic friends-of-friends algorithm). The purpose of `WCSFoF` is to identify all sets of matched detections and write them in an output file structure that transmits the information about the catalogs needed by downstream programs.
- *Affinity:* Each detection is assigned an affinity class. Only objects that have the same affinity are allowed to be friends. The motivation is that galaxy coordinates are not expected to be the same in all filter bands, hence galaxy detections will typically be assigned an affinity by the passband of observation. There is an affinity class `STELLAR` for objects that are expected to have the same coordinates in all exposures.
- *Field:* Each detection must be in exactly one field of the sky. A list of fields must be provided on input. Matching occurs only within fields, not across different fields. A field must have a center, and coordinate matching is done in the gnomonic projection about this center.
- *Extension:* An input catalog of detections. Catalogs are read from FITS binary tables (probably either generated by SExtractor or encapsulating a reference catalog). Each FITS bintable extension is considered a distinct input catalog. Each extension is assigned to one field, and must have some input world coordinate system (WCS) map into celestial coordinates. The input "pixel" coordinates of each detection are mapped to the sky using this WCS and then into the gnomonic system of the field. Each extension's detections are assumed to come from a single device on a single instrument from a single exposure.
- *Attribute:* Any characteristic that we are going to keep track of for every incoming extension, for example the name of its input file, its instrument/device, the name of the table columns holding the $x$ and $y$ coordinates, etc. Attributes are to be read from the input configuration file. See 3.2 for examples.
- *Exposure:* Each extension is assumed to come from one exposure of the sky (a reference catalog would be considered a single exposure). All extensions from a given exposure must

be in the same field. A multi-extension FITS file can contain many extensions (catalogs) for a single exposure. Downstream photometric/astrometric solutions will usually have elements assumed constant for all detections in a common exposure.

- *Instrument:* Each exposure is assumed to have been taken with a single instrument. When astrometric / photometric solutions are derived, elements of the solutions will assumed to be constant for all exposures taken with the same instrument. For example we would want to have a different instrument for each filter of a given camera. We may wish to assign different instruments for different runs of the same camera if we suspect the alignment changed between runs.

- *Device:* An instrument can have multiple devices (*e.g. CCDs*). Each device has its own pixel coordinate system and a distinct WCS. It is assumed that every extension (input catalog) is from a single device of a single exposure with a single instrument.

- *Reference* and *Tag* catalogs: An input catalog (extension) is a *reference* if its position and/or flux information are to be considered known and are not to be re-fit by downstream codes. A *tag* catalog contains information (such as color) that we want associated with a match, but we are not interested in deriving photometric/astrometry solutions for these catalogs. Reference and tag extensions are assigned special instrument ID's.

## 2. Structures

### 2.1. `Point`

A `Point` is an object to be matched. Aside from its coordinate vector, it has three identifying numbers:

- `extensionNumber` identifies which input catalog this came from.

- `objectNumber` identifies the object uniquely in the input catalog.

- `exposureNumber` says which sky exposure this `Point` was observed in.

### 2.2. `PointCat`

The actual matching will occur using the `Catalog` class in *FoF.h*. `Catalog` is a template whose parameters are: (1) the type of the objects to be collected in the catalog, and (2) the number of dimensions $N$ of the matching space. The class being collected must have a method `const getX()` which returns a length-$N$ vector giving its position. In *WCSFoF.cpp* the `PointCat` class is just `typedef fof::Catalog<Point,2>`.

Not worrying right now about how `Catalog` actually works, here are the necessary elements of its interface. It is designed to consider objects within some $N$-dimensional rectangular region. Objects outside the bounds are matched as if they're at the boundary, though not many guarantees about such cases will behave.

- On construction, must be given length-$N$ vectors of the lower and upper bounds of the matching region, plus the distance for which two objects will assumed to be friends for the friends-of-friends algorithm. `getRadius()` returns this matching radius.
- `add(const P& point)` adds a new point to the catalog. A pointer to the input point is saved. The catalog does *not* delete the points when it is destroyed.
- The catalog class can be traversed with iterators as usual in C++.

### 2.3. `Field`

Matches can only occur for objects that are in the same `Field` and friendship is calculated in the coordinate system specific to that field. Each field has a C++ `map` giving a `PointCat` for each affinity name. The method `catalogFor(const string affinity)` returns (a pointer to) the catalog with the given affinity, creating a new catalog if this is a new affinity. Catalogs are destroyed when the `Field` is destroyed.

Each field also has (pointer to) a `SphericalCoords` instance which specifies the map from $(x, y)$ to the celestial sphere in this field. It is destroyed with the `Field`.

### 2.4. `Device`

A `Device` just has a `name` and the (rectangular) bounds of its pixel coordinates.

### 2.5. `Instrument`

An `Instrument` has a `name` and has a vector of the `Device`s that make it up. The class is a `NameIndex` that allows the devices to be indexed either by their numbers or names. The `newDevice()` call adds devices to the `Instrument` and they are destroyed with `Instrument`.

## 3. Inputs

The program is called as

`WCSFoF` ⟨*field specs*⟩ ⟨*exposure specs*⟩ [*parameter file*] [*parameter file*] . . .

The standard input is read as additional parameter specifications. If you have no additional parameters you will need to send an empty file to stdin.

### 3.1. Parameters

Parameters can be specified one per line as ⟨*parameter name*⟩[ =]⟨*value*⟩ [#;] *comment.*

- `matchRadius:` Distance (in arcsec) for friends-of-friends matching algorithm. (Default = 1.)
- `useAffinities:` Force matches to be from the same affinity class (`true`).
- `outName:` Filename for the output FITS tables (*match.cat*).
- `minMatch:` Minimum number of detections for a match to be retained (2).
- `selfMatch:` If false, reject all matches that have more than one detection from the same exposure (`true`).
- `renameInstruments:` A translation table for instrument names. This is a string having the format ⟨*regex1*⟩ = ⟨*replace1*⟩, ⟨*regex2*⟩ = ⟨*replace2*⟩, . . . . Incoming instrument names are checked for matches with any of the regular expressions; if they match one, the name is translated into the replacement test. Examples:
  - `renameInstruments = "i.* = i, r.* = r"` will change anything starting with `i` or `r` to the single-letter names.
  - `renameInstruments "(..).* = \1"` will use just the first two letters of any instrument name. The translated instrument name will be used when looking up an instrument in the input `PixelMapCollection` files.
- `stringAttributes, intAttributes, doubleAttributes:` comma-separated lists of any columns from the input exposure table that are to be passed on to the output tables.

### 3.2. Files

`WCSFoF` requires the names of two input files on the command line. The first is the *field specification* file. Each line of this file describes one field:

```
<Field name> <RA> <Dec> <extent>
```

The field names should be unique and have no spaces. The RA and Dec give the ICRS coordinates of the field center. The extent gives (in degrees) the half-length of the square field that will contain all objects in this field. The program will not properly match objects outside this square (unless they are assigned to another field).

The second input is the name of a FITS file with a single binary table extension containing our *filetable*. Each row of the filetable describes one FITS file that contains one or more extensions that will serve as our input detection catalogs.

The filetable will usually have been prepared by the *parse3.py* preprocessor. The filetable must have the columns specified below. Basically each column of the filetable gives one of the attributes that we will assign to the extensions read for that row of the filetable.

If a cell in the filetable begins with `@`, it is assumed to give the keyword of an entry in the FITS catalog's header. The desired value of the attribute is to be the value of the header card with this keyword. The primary header of the FITS file is searched first, then overridden by any values found in the header of the catalog's binary table extension. An error results if this keyword is not present in the input catalog's header.

The filetable must have these columns:

- `Filename:` (string) the path to the FITS file containing the input catalog(s).
- `Extension:` (int, default = −1) the extension number (zero-indexed) in the FITS file of the binary table detection catalog. If this is a negative integer, then it is assumed that all of the extensions of the FITS file (except extension 0, the primary HDU) are valid binary table catalogs. The code will sense if the FITS file is in the "LDAC" format produced by SExtractor, in which each catalog comes paired with another bintable holding the original image's header information. Note that any time `WCSFoF` needs to look for information in the input catalog headers, it will first look in the primary header of the FITS file, then override this with any values found in the header of the binary table extension (including any LDAC headers).
- `Field:` (string, default `_NEAREST`) the name of the field to which this file's catalogs belong. The special value "`_NEAREST`" can be entered to assign this input file to the field nearest to the `RA` and `Dec` attributes.
- `Exposure:` (string) the name of the exposure to which this catalog(s) belong.
- `Instrument:` (string) the name of the instrument with which the catalog's detections were made. The special values `REFERENCE` and `TAG` indicate the corresponding status for the extension.
- `Device:` (string) the name of the device with which the catalog's detections were made. Note that for multi-extension input files this is almost certainly specified as the value to be found

in a header entry for each extension.

- `RA, Dec:` (strings) the position of the center of this exposure or extension, in the traditional sexagesimal hours/degrees notation, *e.g.* "12:24:30" "-30:04:02.8". Values are not needed if `WCSFile = ICRS` (see below) and the `Field` has been explicitly specified.

- `Select:` (string) specifies the selection criterion for objects from the input catalog to be used as detections. It should be an expression that evaluates to a boolean. For example, `FLAGS == 0 && ERRAWIN_IMAGE<0.05`. Variable names must correspond to columns in the extension's object catalog. Names starting with `@` are taken to be header keywords, and the value associated with the keyword in the extension (or primary) header is used.

- `Star_select:` (string) another boolean expression that determines whether an object is considered a star or a galaxy. Detections passing the star selection are assigned to the affinity class `STELLAR.`

- `Affinity:` (string) the affinity class to which non-stellar detections from this catalog are assigned.

- `xKey, yKey:` (string) the name of the columns in the catalog binary table that give the $x$ and $y$ pixel coordinates of each detection.

- `idKey:` (string, default `_ROW`) the name of the column in the catalog binary table that contains a unique identifying integer for each detection in that catalog. The special value `_ROW` will take the (zero-indexed) row number of the detection in the binary table as its identifier.

- `WCSFile:` (string) This gives the path to a file that, optionally, specifies the WCS for the extension(s) in this row of the filetable. The options here are to enter:

  - The name of a serialized `PixelMapCollection` file. In this case each extension of the catalog file will seek a map in this collection with the name ⟨*exposure name*⟩/⟨*device name*⟩. An error occurs if no map with this name is in the collection.
  - The name of an FITS-header-style ASCII file (as produced by SCAMP), in which we expect to find a WCS specified with FITS conventions. If the input catalog file has multiple extensions, the WCS for each are listed sequentially in the WCS file, seperated by the *END* keyword.
  - Nothing—in which case a WCS will be expected to be specified in the FITS headers of the catalog file.
  - The special value `ICRS`. This means that the $x$ and $y$ coordinates in the catalog are *already* in the ICRS system and no WCS is needed. These coordinates must be in **degrees.**

- Any attribute requested via the parameters `stringAttributes, intAttributes,` or `doubleAttributes` must be present in the filetable.

## 4. Outputs

After `WCSFoF` runs, it has identified all friends-of-friends matches for each affinity in each field. Matches of <`minMatch` detections are discarded. The information gleaned from the input catalogs is output to a multi-extension FITS file (name given by `outName`). This output file has the following extensions.

### 4.1. `Files`

This is a copy of the input filetable.

### 4.2. `Fields`

The extension with HDU name `Fields` is a binary table with one row for each field. In further use, the fields are referred to by their (0-indexed) row number in this table. The columns in this table are

- `Name:` (string) the name of the field.
- `RA, Dec:` (double) the RA and Dec (in **degrees**) of the field center.

### 4.3. `Exposures`

The `Exposures` binary table contains one row for every exposure. In further use, the exposures are referred to by their (0-indexed) row number in this table. The columns in the table are:

- `Name:` (string) the exposure name.
- `RA, Dec:` (double) the coordinates (in **degrees**) of this exposure. Exposures are assigned the sky coordinates of the first extension found to be a member of the exposure.
- `FieldNumber:` (int) the sequence number of the field to which this exposure was assigned.
- `InstrumentNumber:` (int) the sequence number of the instrument used for this exposure.

### 4.4. `Instrument`

An HDU with name "`Instrument`" is produced for each instrument found during the matching. These extensions are given different version numbers to prevent some software from overwriting

them. In the header of this HDU are two cards with keywords `Name` and `Number` giving the name and sequence number assigned to this instrument. This HDU is a binary table for which each row is a device of the instrument. The columns in the table are

- `Name:` (string) the device name.
- `XMin, XMax, YMin, YMax:` (double) the corners of the rectangle that bounds the pixel positions of all detections found on this device.

### 4.5. `Extensions`

The `Extensions` binary table contains one row for every extension from which detections were read. Note this is distinct from the input filetable because one input file on a row of the filetable could have contained many extensions, each of which has its own row in the `Extensions` table. The columns of this table are:

- `Filename, xKey, yKey, idKey:` values read from the input filetable (potentially as evaluated from header keywords).
- Any attributes requested through the parameters `stringAttributes, intAttributes,` or `doubleAttributes` are propagated from the input filetable/headers to the extension table.
- `FileNumber, HDUNumber:` (ints) the (0-indexed) row number in the `Files` extension of the file that holds this extensions data, and the (0-indexed) number of the HDU in that file that holds the catalog.
- `ExposureNumber:` (int) the row of the `Exposures` table that describes the exposure for this extension.
- `DeviceNumber:` (int) the row of the relevant `Instrument` table that describes the device for this extension.
- `WCS:` (string) a serialized `PixelMapCollection` specifying the input WCS used for this extension. Could take the value `ICRS` if the "pixel" coordinates of the objects are actually degree-valued RA and Dec in the ICRS system.

### 4.6. `MatchCatalog`

Finally there are a series of binary tables that each list the matched detections for a given (affinity, field) pair. Each of these HDUs has the name `MatchCatalog`. Each HDU has header fields with keywords `Field, FieldNum,` and `Affinity` which specify the name and sequence number of

this field and the affinity of the objects matched herein. Each row of the table specifies a detection that has been matched.

- `SequenceNumber:` (int) a counter that resets to zero whenever we start a new match. For example if successive rows have values (0, 1, 2, 3, 0, 1, 2) this means that the first 4 form a match and the last 3 form a distinct match.
- `Extension:` (long) the row number in the `Extensions` table of the extension from which this detection originates.
- `Object:` (long) the identification number of this object in the input extension catalog.